OASIS ebXML RegRep Service and Protocols (ebRS) Version 4.0

Draft 4

February 28, 2010

Specification URIs:

Release Notes:

http://wxforge.wx.ll.mit.edu:8080/jira/secure/ReleaseNote.jspa?projectId=10023&styleName=Html&version=10076

This Version:

http://docs.oasis-open.org/regrep/4.0-~~cd3~~**cd4**-draft1/specs/core/regrep-rs.html

http://docs.oasis-open.org/regrep/4.0-~~cd3~~**cd4**-draft1/specs/core/regrep-rs.odt

http://docs.oasis-open.org/regrep/4.0-~~cd3~~**cd4**-draft1/specs/core/regrep-rs.pdf

Previous Version:

http://docs.oasis-open.org/regrep/4.0-~~cd2~~**cd3**-draft1/specs/core/regrep-rs.html

http://docs.oasis-open.org/regrep/4.0-~~cd2~~**cd3**-draft1/specs/core/regrep-rs.odt

http://docs.oasis-open.org/regrep/4.0-~~cd2~~**cd3**-draft1/specs/core/regrep-rs.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 RegRep.

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

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

Table of Contents

Illustration Index

402

Index of Tables

403

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 Error: Reference source not found.

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.

[ebRIM] ebXML Registry Information Model Version 4.0 http://www.oasis-open.org/committees/regrep/documents/4.0/specs/regrep-rim-4.0-cs.pdf

[**RFC 1766]** IETF (Internet Engineering Task Force). RFC 1766: Tags for the Identification of Languages, ed. H. Alvestrand. 1995. http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html

[RFC 2130] IETF (Internet Engineering Task Force). RFC 2130 The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996 http://www.faqs.org/rfcs/rfc2130.html

[RFC 2277] IETF (Internet Engineering Task Force). RFC 2277: IETF policy on character sets and languages, ed. H. Alvestrand. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html

[RFC 2278] IETF (Internet Engineering Task Force). RFC 2278: IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html

[RFC2616] IETF (Internet Engineering Task Force). RFC 2616: Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1* . 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[RFC2965] IETF (Internet Engineering Task Force). RFC 2965: D. Kristol et al. *HTTP State Management Mechanism*. 2000. http://www.w3.org/Protocols/rfc2616/rfc2616.html

[SOAP-MF] SOAP Version 1.2 Part1 – Messaging Framework, April 2007 http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[SOAP-ADJ] SOAP Version 1.2 Part2 – Adjuncts, April 2007 http://www.w3.org/TR/2007/REC-soap12-part2-20070427/

[WSA-CORE] Web Services Addressing – Core 1.0 http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WSA-SOAP] Web Services Addressing – SOAP Binding 1.0 http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/

[WSA-WSDL] Web Services Addressing 1.0 - WSDL Binding, February 2006. http://www.w3.org/TR/ws-addr-wsdl

[WSDL2] Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language http://www.w3.org/TR/wsdl20

[WSS-CORE] WS-Security Core Specification 1.1, February 2006.
http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[WSS-UNT] WS-Security Username Token Profile 1.1, February 2006.
http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf

[WSS-X509] WS-Security X.509 Token Profile 1.1, February 2006.
http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf

[WSS-SAML] WS-Security SAML Token profile 1.1, February 2006.
http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf

[WSS-KRB] WS-Security Kerberos Token Profile 1.1, February 2006.
http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf

[XML] T. Bray, et al. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, October 2000.
http://www.w3.org/TR/REC-xml

[XMLDSIG] XML-Signature Syntax and Processing
http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

1.3 Non-normative References

[IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
http://www.iana.org/

[WSDL] W3C Note. Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[UML] Unified Modeling Language
http://www.uml.org
http://www.omg.org/cgi-bin/doc?formal/03-03-01

[UUID] DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20

[ISO19135] Geographic information -- Procedures for item registration
http://www.iso.org/iso/catalogue_detail.htm?csnumber=32553

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 (Informative)

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 RegRegRep server.
- The *ebXML Registry Services and Protocols (ebRS)* defines the services provided by an ebXML RegRep server and the protocols used by clients of the registry to interact with these services.

2.3 Abstract Protocol

This section describes the types RegistryRequestType, RegistryResponseType and RegistryExceptionType defined within rs.xsd that are the abstract types used by most protocols defined by this specification in subsequent chapters. A typical registry protocol is initiated by a request message that extends RegistryRequestType. In response the registry server sends a response that extends RegistryResponseType. If an error is encountered by the server during the processing of a request, the server returns a fault message that extends the RegistryExceptionType.

2.3.1 RegistryRequestType

The RegistryRequestType is the abstract base type for most requests sent by client to the server.

2.3.1.1 Syntax

```
<complexType name="RegistryRequestType">
  <complexContent>
    <extension base="rim:ExtensibleObjectType">
      <attribute name="id" type="string" use="required"/>
      <attribute name="comment" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>
```

2.3.1.2 Description

- Attribute comment – The comment attribute if specified contains a String that describes the request. A server SHOULD create a Comment instance and associate it with the AuditableEvent(s) for that request using the Generic Comment mechanism (need reference??).
- Attribute id – The id attribute must be specified by the client to uniquely identify a request. Its value SHOULD be a UUID URN like “urn:uuid:a2345678-1234-1234-123456789012”.

2.3.2 RegistryResponseType

The RegistryResponseType is the abstract base type for most responses sent by the server to the client in response to a client request. A global RegistryResponse element is defined using this type which is used by several requests defined within this specification.

2.3.2.1 Syntax

```
<complexType name="RegistryResponseType">
  <complexContent>
    <extension base="rim:ExtensibleObjectType">
      <sequence>
        <element name="RegistryObjectList" type="rim:RegistryObjectListType"
          minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="status" type="rim:objectReferenceType"
        use="required"/>
      <attribute name="requestId" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>
<element name="RegistryResponse" type="tns:RegistryResponseType"/>
```

2.3.2.2 Description

- Element RegistryObjectList – This element contains a sequence of zero or more RegistryObject elements. It is used by requests that return a list of RegistryObject instances. An example is the QueryResponse element that is returned in response to a QueryRequest and contains the list of RegistryObject instances that match the specified query.
- Attribute requestId – This attribute contains the id of the request that returned this QueryResponse.

- Attribute status – This attribute contains the status of the response. Its value MUST be a reference to a ClassificationNode within the canonical ResponseStatusType ClassificationScheme.

2.3.3 RegistryExceptionType

The RegistryExceptionType is the abstract base type for all exception or fault messages sent by the server to the client in response to a client request. A list of all protocol exceptions is available in the Protocol Exceptions appendix.

2.3.3.1 Syntax

```
<complexType name="RegistryExceptionType">
  <attribute name="code" type="string" use="optional"/>
  <attribute name="severity" type="rim:objectReferenceType"
    default="urn:oasis:names:tc:ebxml-regrep:ErrorSeverityType:Error"/>
</complexType>
```

2.3.3.2 Description

- Attribute code – The code attribute value may be used by a server to provide a brief code or identifier for an Exception.
- Attribute severity – The severity attribute value provides a severity level for the exception. Its value MUST reference a ClassificationNode within the canonical ErrorSeverityType ClassificationScheme.

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. Parameterized queries are similar in concept to stored procedures in SQL.

This specification defines a number of canonical queries that are standard queries that MUST be supported by a server. Profiles, 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 QueryDefinition object which is defined by ebRIM. The definition of a QueryDefinition may contain any number of Parameters supported by the query.

3.1.1 Invoking Adhoc Queries

A client may invoke a client specific ad hoc query using the canonical AdhocQuery 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 AdhocQuery 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.

3.2 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

3.2.1 QueryRequest

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

3.2.1.1 Syntax

```
<element name="QueryRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="ResponseOption" type="tns:ResponseOptionType"
            minOccurs="1" maxOccurs="1"/>
          <element name="Query" type="rim:QueryType"
            minOccurs="1" maxOccurs="1" />
        </sequence>
        <attribute name="federated" type="boolean"
          use="optional" default="false"/>
        <attribute name="federation" type="anyURI" use="optional"/>
        <attribute name="format" type="string"
          use="optional" default="application/ebrim+xml"/>
        <attribute ref="xml:lang" use="optional"/>
        <attribute name="startIndex" type="integer" default="0"/>
        <attribute name="maxResults" type="integer" default="-1"/>
        <attribute name="depth" type="integer" default="0"/>
        <attribute name="matchOlderVersions" type="boolean"
          use="optional" default="false"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

3.2.1.2 Example

The following example shows a QueryRequest which gets an object by its id using the canonical GetObjectById query.

```
<query:QueryRequest maxResults="-1" startIndex="0" ...>
    <rs:ResponseOption returnComposedObjects="true"
returnType="LeafClassWithRepositoryItem"/>
    <query:Query queryDefinition="urn:oasis:names:tc:ebxml-
regrep:query:GetObjectById">
        <rim:Slot name="id">
            <rim:ValueList>
                <rim:ValueListItem xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
                    <rim:Value>%danyal%</rim:Value>
                </rim:ValueListItem>
            </rim:ValueList>
        </rim:Slot>
    </query:Query>
</query:QueryRequest>
```

3.2.1.3 Description

- Element ResponseOption - This required element allows the client to control the format and content of the QueryResponse generated by the server in response to this request.
- Element Query - This element identifies a parameterized query and supplies values for its parameters.

- 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.

- Attribute federated – This optional attribute specifies that the server must process this query as a federated query. By default its value is *false*. This value MUST be false when a server routes a federated query to another server. This is to avoid an infinite loop in federated query processing.

- Attribute federation - This optional attribute specifies the id of the target Federation for a federated query in case the server is a member of multiple federations. In the absence of this attribute a server must route the federated query to all registries that are a member of all federations configured within the local server. This value MUST be unspecified when a server routes a federated query to another server. This is to avoid an infinite loop in federated query processing.

- 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” which returns the response in ebRS QueryResponse format.

- Attribute lang - 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.

- Attribute matchOlderVersions – This optional attribute specifies the behavior when multiple versions of the same object are matched by a query. When the value of this attribute is specified as *false* (the default) then a server MUST only return the latest matched version for any object and MUST not return older versions of such objects even though they may match the query. When the alue of this attribute is specified as *true* then a server MUST return all matched versions of all objects.

- Attribute maxResults - This optional attribute specifies a limit on the maximum number of results the client wishes the query to return. If unspecified, the server SHOULD return either all the results, or in case the result set size exceeds a server specific limit, the server SHOULD return a sub-set of results that are within the bounds of the server specific limit. This attribute is described further in the Iterative Queries section.

- 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). This attribute is described further in the Iterative Queries section.

3.2.1.4 Response

This request returns QueryResponse as response.

3.2.1.5 Exceptions

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

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

3.2.2 Element Query

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

3.2.2.1 Syntax

```
<complexType name="QueryType">
  <complexContent>
    <extension base="tns:ExtensibleObjectType">
      <attribute name="queryDefinition"
        type="tns:objectReferenceType" use="required"/>
    </extension>
  </complexContent>
</complexType>
<element name="Query" type="tns:QueryType"/>
```

3.2.2.2 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 rim:Parameter definition within the rim:QueryDefinition definition. The slot value provides a value for the parameter. The slot value's type MUST match the dataType attribute for the rim:Parameter definition within the rim:QueryDefinition. 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.

3.2.3 Element ResponseOption

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

3.2.3.1 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"/>
```

3.2.3.2 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 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 server *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.

3.2.4 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.

3.2.4.1 Syntax

```
<element name="QueryResponse">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryResponseType">
        <attribute name="startIndex" type="integer" default="0"/>
        <attribute name="totalResultCount" type="integer" use="optional"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

3.2.4.2 Example

The following shows a sample response for the example QueryRequest presented earlier.

```
<query:QueryResponse totalResultCount="1" startIndex="0"
status="urn:oasis:names:tc:ebxml-regrep:ResponseStatusType:Success">
  <query:RegistryObjectList>
    <RegistryObject xsi:type="PersonType"
      status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
      objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:Person"
      lid="urn:acme:Person:Danyal" id="urn:acme:Person:Danyal">
      <Name>
        <LocalizedString value="Danyal Najmi" xml:lang="en-US"/>
      </Name>
      <VersionInfo versionName="1"/>
      <PersonName lastName="Najmi" middleName="Idris" firstName="Danyal"/>
    </RegistryObject>
  </query:RegistryObjectList>
</query:QueryResponse>
```

3.2.4.3 Description:

- Element RegistryObjectList (inherited) - This is the element that contains the RegistryObject instances that matched the specified query. A server MUST provide this element in a QueryResponse even if it contains no RegistryObject instances.
- 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. This attribute is described further in the Iterative Queries section.
- Attribute totalResultCount - This optional parameter specifies the size of the complete result set matching the query within the server. When this value is unspecified, the client should assume it is the size of the result set contained within the result. When this value is -1, the client should assume that the number of total results is unknown. In this case the client should keep iterating through the remaining result set for the query until no more results are returned. This attribute is described further in the Iterative Queries section.

3.2.5 Iterative Queries

The QueryRequest and QueryResponse support the ability to iterate over a large result set matching a query by allowing multiple QueryRequest requests to be submitted in succession such that each query requests a different subset of results within the result set. This feature enables the server to handle queries that match a very large result set, in a scalable manner. The iterative query feature is accessed via the startIndex and maxResults parameters of the QueryRequest and the startIndex and totalResultCount parameters of the QueryResponse as described earlier.

A server MUST return a result set whose size is less than or equal to the maxResults parameter depending upon whether enough results are available starting at startIndex.

The iterative queries feature is not a true Cursor capability as found in databases. A server is not required to maintain transactional consistency or state between iterations of a query. Thus it is possible for new objects to be added or existing objects to be removed from the complete result set in between iterations. As a consequence it is possible to have a result set element be skipped or duplicated between iterations. However, a server MUST return the same result in a deterministic manner for the same QueryRequest if no changes have been made in between the request to the server (or servers in case of federated queries).

Note that while it is not required, a server MAY implement a transactionally consistent iterative query feature.

3.3 Parameterized Query Definition

A parameterized query is defined by submitting a <rim:QueryDefinition> object to the server using the submitObjects protocol. A detailed specification of the <rim:QueryDefinition> 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.

3.4 Canonical Query: AdhocQuery

The canonical query AdhocQuery 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.

3.4.1 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

3.4.2 Query Semantics

- The server MUST return a QueryException 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

3.5 Canonical Query: BasicQuery

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

3.5.1 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	string		0..1

	attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value			
status	Matches RegistryObjects whose status attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1

3.5.2 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

3.6 Canonical Query: ClassificationSchemeSelector

The canonical query ClassificationSchemeSelector allows clients to create a Subscription to a remote server to replicate a remote ClassificationScheme. This query may be used as Selector query in the subscription as defined in the object replication feature.

3.6.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
classificationSchemeId	Matches rim:ClassificationScheme/@id. Does not allow wildcards.	string		1

3.6.2 Query Semantics

- The server MUST return the specified ClassificationScheme and all ClassificationNodes that are descendants of that ClassificationScheme.
- The ClassificationNodes MUST NOT be returned as nested elements inside their parent Taxonomy element. Instead they MUST be returned as sibling elements with the RegistryObjectList element of the QueryResponse.

3.7 Canonical Query: FindAssociations

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

3.7.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches Associations where rim:/Association/@type 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 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

3.7.2 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

3.8 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.

3.8.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches associated RegistryObjects where the Association's associationsType, rim:/Association/@type 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 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
- Either sourceObjectId or targetObjectId MUST be specified. If neither are specified then QueryException fault MUST be returned
- Both sourceObjectId and targetObjectId MUST NOT be specified. If both are specified then QueryException 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

3.9 Canonical Query: GetAuditTrailById

The canonical query GetAuditTrailById allows clients to get the change history or audit trail for a RegistryObject whose id attribute value is the same as the value of the id parameter.

3.9.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
endTime	Specifies the end of the time interval (inclusive) for rim:AuditableEvent/@timestamp value	dateTime		0..1
id	Matches rim:RegistryObject/@id.	string		1
startTime	Specifies the end of the time interval (inclusive) for rim:AuditableEvent/@timestamp value	dateTime		0..1

3.9.2 Query Semantics

- The server MUST return a set of AuditableEvents that affected the object with id matching the specified id parameter value. The set is sorted by the timestamp attribute value in descending order (latest first)
- If startTime is specified the server MUST only include AuditableEvents whose timestamp is >= startTime parameter value
- If endTime is specified the server MUST only include AuditableEvents whose timestamp is <= endTime parameter value

3.10 Canonical Query: GetAuditTrailByLid

The canonical query GetAuditTrailByLid allows clients to get the change history or audit trail for all RegistryObjects whose lid attribute value is the same as the value of the lid parameter.

3.10.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
endTime	Specifies the end of the time interval (inclusive) for rim:AuditableEvent/@timestamp value	dateTime		0..1
lid	Matches rim:RegistryObject/@lid.	string		1
startTime	Specifies the end of the time interval (inclusive) for rim:AuditableEvent/@timestamp value	dateTime		0..1

3.10.2 Query Semantics

- The server MUST return a set of AuditableEvents that affected objects with lid matching the specified lid parameter value. The set is sorted by the timestamp attribute value in descending order (latest first)
- If startTime is specified the server MUST only include AuditableEvents whose timestamp is >= startTime parameter value
- If endTime is specified the server MUST only include AuditableEvents whose timestamp is <= endTime parameter value

3.11 Canonical Query: GetChildrenByParentId

The canonical query GetChildrenByParentId allows clients to get the children of a RegistryObject whose Id attribute value is the same as the value specified for the parentId parameter. This query is used within structures with parent-child relationships such as the following:

- ClassificationScheme – Child ClassificationNodes
- Organization – Child Organizations
- RegistryPackage – RegistryPackage Members

3.11.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
objectType	Specifies the type of parent-child hierarchy for the query	string		0..1
parentId	Specifies the id of the parent object	string		0..1

3.11.2 Query Semantics

- If objectType and parentId are both unspecified the server MUST return all RegistryObjects that are not members of a RegistryPackage (root level objects)

- If parentId parameter is unspecified and objectType parameter is specified the server MUST return all root level objects for the parent-child hierarchy identified by the objectType as follows:
  - If objectType parameter value contains the string “ClassificationScheme” the server MUST return all ClassificationSchemes
  - If objectType parameter value contains the string “Organization” the server MUST return all Organizations that are not a member of another Organization (root level Organizations)
  - If objectType parameter value contains the string “RegistryPackage” the server MUST return all RegistryPackages that are not a member of another RegistryPackage (root level RegistryPackages)
- If parentId parameter is specified then the behavior is as follows:
  - If objectType parameter value is unspecified or if its value contains the string “RegistryPackage” the server MUST return all RegistryObjects that are member of a RegistryPackage whose id is the same as the value of the parentId attribute
  - If objectType parameter is specified and its value contains the string “ClassificationScheme” the server MUST return all ClassificationNodes that are children of a TaxonomyElementType instance whose id is the same as the value of the parentId attribute
  - If objectType parameter is specified and its value contains the string “Organization” the server MUST return all Organizations that are members of an Organization whose id is the same as the value of the parentId attribute

3.12 Canonical Query: GetClassificationSchemesById

The canonical query GetClassificationSchemesById allows clients to fetch specified ClassificationSchemes.

3.12.1 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

3.12.2 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

3.13 Canonical Query: GetRegistryPackagesByMemberId

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

3.13.1 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

3.13.2 Query Semantics

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

3.14 Canonical Query: GetMembersByRegistryPackageId

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

3.14.1 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

3.14.2 Query Semantics

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

3.15 Canonical Query: GetNotification

The canonical query GetNotification allows clients to "pull" any pending Notification for a Subscription at a time of their choosing. This is defined in detail under section titled "Pulling Notification on Demand".

3.15.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality

subscriptionId	Matches rim:Subscription/@id. Wildcards are not allowed.	string		1
startTime	The time since which events should be included in the Notification	Xs:dateTime		0..1

3.15.2 Query Semantics

- The server MUST return a Notification with events that affected objects matching the query selector query for the Subscription.
- The server MUST return only those events that have a timestamp later than startTime.

3.16 Canonical Query: GetObjectById

The canonical query GetObjectById allows clients to find RegistryObjects based upon the value of their id attribute.

3.16.1 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		1

3.16.2 Query Semantics

- The server MUST return the any pending Notification for the specified subscription.

3.17 Canonical Query: GetObjectsByLid

The canonical query GetObjectByLid allows clients to find RegistryObjects based upon the value of their lid attribute. It is used to fetch all versions of a logical object without any specific order or relationship among them.

3.17.1 Parameter Summary

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

3.17.2 Query Semantics

- The server MUST return all RegistryObject that whose lid attribute value matches the specified value of the lid parameter.

3.18 Canonical Query: GetReferencedObject

The canonical query GetReferencedObject allows clients to get a RegistryObject that is the target of an rim:objectReferenceType attribute value.

3.18.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
objectReference	Contains the value for an rim:objectReferenceType attribute	string		0..1

3.18.2 Query Semantics

- The server MUST return the RegistryObjectType instance that is being referenced by the specified value for the objectReference parameter.
  - If the objectReference contains the id of a local object that is not a DynamicObjectRef instance then the server MUST return that object.
  - If the objectReference contains the id of a local DynamicObjectRef instance then the server MUST be invoke the Query within the DynamicObjectRef instance and resolve the reference to the singleton result of the Query.
  - If the objectReference contains the canonical URL for a remote object then the server MUST invoke the GetReferencedObject query against the remote server using the id of the remote object as the value of the objectReference parameter. The id of the remote object is accessible from its canonical URL as the value of the id parameter within the URL.

3.19 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.

3.19.1 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

3.19.2 Parameter Summary

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

3.19.3 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.

  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.

3.20 RegistryPackageSelector

The canonical query RegistryPackageSelector allows clients to create a Subscription to a remote server to replicate a remote RegistryPackage as well as all its member objects and the AssociationType instances that relate the members of the RegistryPackage to it. This query MAY be used as Selector query within the Subscription for the replication as defined in the object replication feature.

3.20.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
registryPackageId	Matches rim:Registry/@id.	string		1

	Does not allow wildcards.			

3.20.2 Query Semantics

- The server MUST return the specified RegistryPackageType instance, all RegistryObjectType instances that are members of the specified RegistryPackage as well as all "HasMember" AssociationType instances between the RegistryPackageType instance and its members. that are descendants of that ClassificationScheme.
- The member RegistryObjectType instances MUST NOT be returned as nested elements inside the RegistryPackage. Instead they MUST be returned as sibling elements with the RegistryPackage and Associations within the RegistryObjectList element of the QueryResponse.

3.21 Query Functions

A server MAY support any number of pre-defined functions known as *Query Functions*, that may be used within a query expression or query parameter. Query functions are similar in concept to functions in SQL. Query functions may be used within the query expression of a parameterized query as well as within its invocation parameter values. Query functions enable parameterized queries to use reusable specialized search algorithms to augment their capabilities.

This specification defines a number of canonical functions that are standard functions that MUST be supported by a server. Profiles, implementations and deployments may define additional query functions beyond the canonical functions defined by this specification.

3.21.1 Using Functions in Query Expressions

A parameterized query stored as a rim:QueryDefinition instance MAY have a rim:QueryExpression which defines defines a query expression within its sub-nodes. A client MAY submit a rim:QueryDefinition such that its query expression may use any number of query functions supported by the server any where within the query expression where it is syntactically correct to use value returned by the function.

If a query expression contains one or more function invocations then the query expression MUST delimit the parts of the query expression that are not a function invocation with the leading characters "#@" and trailing characters "@#". This is similar in syntax to a java multi-line comment syntax where a comment is delimited by leading characters "/*" and trailing characters "*/". The delimiters serve the following purposes:

- Allows a parser to recognize the non-function parts of the query expression that MUST be preserved *as is*
- Allows implementations to optimize by skipping function parsing and evaluation if the special delimiter characters are not present in query expression

The following is an example of an SQL query expression which uses the subClassificationNode function to match all RegistryObjects that are targets of Association with specified sourceObject and type that a subnode of AffiliatedWith node upto a depth of 2 levels in the descendant hierarchy. The delimiter characters are in bold font while the function invocations is in bold and italic font below:

```
--example of a query expression with query functions
#@SELECT targetObject.* FROM
RegistryObjectType targetObject, AssociationType a WHERE

  a.sourceObject = :sourceObject AND
  a.type IN (@# subClassificationNode('urn:oasis:names:tc:ebxml-
regrep:AssociationType:AffiliatedWith', 2, ",") #@) AND
```

```
  targetObject.id = a.targetObject@#
```

3.21.2 Using Functions in Query Parameters

A client MAY use query functions supported by a server within parameter values specified when invoking a parameterized query. A client MAY invoke a parameterized query using the Query protocol such that its query parameter values may use any number of query functions supported by the server any where within the query parameter where it is syntactically correct to use value returned by the function.

If a query parameter value contains one or more function invocations then the query expression MUST delimit the parts of the query parameter that are not a function invocation with the leading characters "#@" and trailing characters "@#". If a query parameter value only has function invocations and contains no non-function parts then it must include at least one leading or trailing "#@@#" delimiter token pair to allow optimized parsing and evaluation of query functions only when needed.

The following is an example of a query expression that has no query functions. Its two parameters are shown in bold font:

```
--Following is the query expression within the server
--This time it has no query functions as they are in the query parameters
SELECT targetObject.* FROM
RegistryObjectType targetObject, AssociationType a WHERE

  a.sourceObject = :sourceObject AND
  a.type IN ( :types ) AND
  targetObject.id = a.targetObject
```

The following is an example of invocation of a parameterized query that uses the above query expression and uses the subClassificationNode function from previous example within the value of the *types* parameter. Note the trailing "#@@#" delimiter tokens are present as required.

```
<query:QueryRequest maxResults="-1" startIndex="0" ...>
   <rs:ResponseOption returnComposedObjects="true"
returnType="LeafClassWithRepositoryItem"/>
   <query:Query queryDefinition="urn:acme:ExampleQuery">
      <rim:Slot name="sourceObject">
         <rim:ValueList>
            <rim:ValueListItem xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
               <rim:Value>urn:test:Person:Danyal</rim:Value>
            </rim:ValueListItem>
         </rim:ValueList>
      <rim:Slot name="types">
         <rim:ValueList>
            <rim:ValueListItem xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
               <rim:Value>subClassificationNode('urn:oasis:names:tc:ebxml
-regrep:AssociationType:AffiliatedWith', 2, ',', 'ebrs:null')#@@#</rim:Value>
            </rim:ValueListItem>
         </rim:ValueList>      </rim:Slot>
   </query:Query>
</query:QueryRequest>
```

3.21.3 Function Processing Model

A server MUST meet the following function processing requirements during the processing of a QueryRequest:

- When processing a query expression elements (rim:QueryDefinition/rim:QueryExpression) the server SHOULD NOT perform function processing if the special delimiter sequences of "#@" and "@#" are not found in the query expression
- When processing query invocation parameter elements (query:QueryRequest/query:Query/rim:Slot/rim:ValiueList/rim:ValueListItem) the server SHOULD NOT perform function processing if the special delimiter sequences of "#@" and "@#" are not found in the query expression
- When processing a query expression element if the special delimiter sequences of "#@" and "@#" are found then the server MUST process query expression elements to replace all function invocations with the value returned when the function is invoked with specified parameters
- When processing query invocation parameter elements if the special delimiter sequences of "#@" and "@#" are found then the server MUST process each query parameter element to replace all function invocations with the value returned when the function is invoked with specified parameters
- When invoking a function that has another function invocation as its parameter the inner most functions MUST be invoked first so that the outer function can be invoked with the value returned by the inner function invocation
- When processing a query expression or query parameter the special delimiter characters "#@" and "@#" MUST be removed and the value contained within them MUST be preserved without any change

3.21.4 Function Processor BNF

The following BNF grammar normatively describes the grammar for query expressions and query invocation parameters with embedded function invocations. The ***start*** production describes the grammar for query expressions and query invocation parameters with embedded function invocations.

```
<DEFAULT> SKIP : {
" "
| "\t"
| "\r"
| "\n"
}

<DEFAULT> TOKEN : {
<S_NUMBER: <FLOAT> | <FLOAT> (["e","E"] (["-","+"])? <FLOAT>)?>
| <#FLOAT: <INTEGER> | <INTEGER> ("." <INTEGER>)? | "." <INTEGER>>
| <#INTEGER: (<DIGIT>)+>
| <#DIGIT: ["0"-"9"]>
| <BOOLEAN: "true" | "false">
}

<DEFAULT> TOKEN : {
<S_IDENTIFIER: (<LETTER>)+ (<DIGIT> | <LETTER> | <SPECIAL_CHARS>)*>
| <#LETTER: ["a"-"z","A"-"Z"]>
| <#SPECIAL_CHARS: "_">
| <S_CHAR_LITERAL: "\'" (~["\'"])* "\'" ("\'" (~["\'"])* "\'")*>
| <S_QUOTED_IDENTIFIER: "\"" (~["\n","\r","\""])* "\"">
```



```
| <OPENPAREN: "(">
| <CLOSEPAREN: ")">
| <COMMA: ",">
| <COLON: ":">
| <DELIMITED_TEXT: "#@" (~["@"])* "@#">
}

start ::= ( textOrFunctionCall )+ <EOF>
text ::= ( ( <DELIMITED_TEXT> ) )
textOrFunctionCall ::= ( text | FunctionCall )
FunctionCall ::= FunctionReference <OPENPAREN> FunctionArgumentList
<CLOSEPAREN>
FunctionReference ::= <S_IDENTIFIER> <COLON> <S_IDENTIFIER>
FunctionArgumentList ::= FunctionArgument ( <COMMA> FunctionArgument )*
FunctionArgument ::= ( FunctionCall | <S_CHAR_LITERAL> |
<S_QUOTED_IDENTIFIER> | <S_NUMBER> | <BOOLEAN> )
```

3.22 Common Patterns In Query Functions

This section defines some commonly occurring patterns in query functions and defines some common solutions to addressing these patterns. Profiles are SHOULD conform to the solutions defined in this section whenever possible.

3.22.1 Specifying a null Value for string Param or Return Value

A function that accepts a string parameter SHOULD treat a value of “ebrs:null” as a null string. A null string is a string whose value is unspecified.

When a function returns a “string” type it SHOULD return a null value string as the canonical value “ebrs:null”.

3.23 Canonical Functions

This section defines a set of standard canonical functions that MUST be supported by all servers. A client MAY use these functions within a query expression or within the value of a parameter to a parameterized query. A server MUST process the functions according to their behavior as specified in this section. The function processing model is specified in Function Processing Model.

A client MUST use the “ebrs:” namespace prefix when using a canonical function defined by this profile. Profiles of this specification MAY define their own canonical functions as well as a standard namespace prefix to be used with these functions.

Table 3 summarizes the canonical functions defined by this specification.

Function Name	Semantics
currentTime	Returns the current time in ISO 8601 format
currentUserId	Returns the id of the user associated with the current RegistryRequest
relativeTime	Returns a time in the future or past, relative to the current time where the offset period is determined by specified parameter
subClassificationNodes	Returns descendant classification nodes of specified node up to specified depth
superClassificationNodes	Returns ancestor classification nodes of specified node up to specified depth

Table 3: Canonical Functions Defined By This Profile

3.23.1 Canonical Function: currentTime

This canonical function takes no parameters and returns the current time associated with the server.

3.23.1.1 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The value of the string MUST be current time in ISO 8601 format using the UTC time zone. An example of value returned is “2010-02-25T15:22:14.534Z”.

3.23.2 Canonical Function: currentUserId

This canonical function takes no parameters and returns a string whose value is the id of the user associated with the current RegistryRequest. This specification does not define how user's are managemed within the server not does it define how an is assigned to a user.

3.23.2.1 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The value of the string MUST be “ebrs:null” if no current user is associated with the RegistryRequest

3.23.3 Canonical Function: relativeTime

This canonical function takes a string parameter in the format specified by xs:duration that specify a time offset period and returns a time in the future or past relative to the current time by the specified period.

3.23.3.1 Parameter Summary

Parameter	Description	Data Type
duration	A duration of time in the format as specified by the duration type defined by XML Schema duration type. The duration format supports negative or positive durations so this function may be used to return a time relative to current in the future or the past.	duration

3.23.3.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The format of the duration parameter MUST conform to the format as specified by the duration type defined by XML Schema duration type otherwise the server MUST return InvalidRequestException
- The value of the string MUST be a time in ISO 8601 format that is offset by the specified period in the future relative to the current time. An example of value returned is "2010-02-25T15:22:14.534Z"

3.23.4 Canonical Function: subClassificationNode

This canonical function takes an ClassificationNode's id as parameter and returns all ClassificationNode's that are descendants of the specified ClassificationNode and within the specified number of generations as indicated by the depth parameter.

3.23.4.1 Parameter Summary

Parameter	Description	Data Type
classificationNodeId	The value of this parameter specifies the id of a ClassificationNodeType instance	string
depth	Specifies how many generations deep to match descendants	integer
delimiter	The value of this parameter specifies the delimiter string to be used as separator between the tokens representing the ids matched by the function	string
template	The value of this parameter specifies a template to contain each id returned by the function. The template may contain one or more occurances of template parameter string "${id}" as placeholder for the id of a matched ClassificationNode	string

3.23.4.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The string MUST be "ebrs:null" if no ClassificationNode is found that is a descendant of specified ClassificationNode within the specified depth or if specified ClassificationNode does not exist
- The string MUST consist of a set of substrings separated by the appropriate delimiter character when any ClassificationNode's are found that are descendant of specified ClassificationNode within the specified depth:
  - There MUST be a substring for each ClassificationNode matched by the function
  - Each substring MUST conform to the specified template such that all occurrences of ${id} are replaced by the id of a ClassificationNode matched by the function
-
- A depth of N where N > 0 matches the Nth generation descendants of the specified ClassificationNode. For example a depth of 1 matches the immediate children of the specified

ClassificationNode while a depth of 2 matches the grandchildren of the specified ClassificationNode

- A depth of -1 matches all descendants of the specified ClassificationNode
- A template value of “ebrs:null” is implicitly equivalent to a template value of “${id}”

3.23.5 Canonical Function: superClassificationNode

This canonical function takes an ClassificationNode's id as parameter and returns all ClassificationNode's that are ancestors of the specified ClassificationNode and within the specified number of generations as indicated by the depth parameter.

3.23.5.1 Parameter Summary

Parameter	Description	Data Type
classificationNodeId	The value of this parameter specifies the id of a ClassificationNodeType instance	string
depth	Specifies how many generations deep to match ancestors	integer
delimiter	The value of this parameter specifies the delimiter string to be used as separator between the tokens representing the ids matched by the function	string
template	The value of this parameter specifies a template to contain each id returned by the function. The template may contain one or more occurrences of template parameter string “${id}” as placeholder for the id of a matched ClassificationNode	string

3.23.5.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The string MUST be “ebrs:null” if no ClassificationNode is found that is a ancestor of specified ClassificationNode within the specified depth or if specified ClassificationNode does not exist
- The string MUST consist of a set of substrings separated by the appropriate delimiter character when any ClassificationNode's are found that are ancestors of specified ClassificationNode within the specified depth:
  - There MUST be a substring for each ClassificationNode matched by the function
  - Each substring MUST conform to the specified template such that all occurrences of ${id} are replaced by the id of a ClassificationNode matched by the function
- A depth of N where N > 0 matches the Nth generation ancestors of the specified ClassificationNode. For example a depth of 1 matches the immediate parents of the specified ClassificationNode while a depth of 2 matches the grandparents of the specified ClassificationNode
- A depth of -1 matches all ancestors of the specified ClassificationNode
- A template value of “ebrs:null” is implicitly equivalent to a template value of “${id}”

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 LifecycleManager 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

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">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="RegistryObjectList" type="rim:RegistryObjectListType"
```

```
            minOccurs="0" maxOccurs="1"/>
          <element name="ObjectRefList" type="rim:ObjectRefListType"
            minOccurs="0" maxOccurs="1" />
        </sequence>
        <attribute name="mode" use="optional" default="CreateOrReplace">
          <simpleType>
            <restriction base="NCName">
              <enumeration value="CreateOrReplace"/>
              <enumeration value="CreateOrVersion"/>
              <enumeration value="CreateOnly"/>
            </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 server. The RegistryObjects in the list may be brand new objects being submitted to the server or they may be current objects already existing in the server.
- Attribute mode – 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
  - CreateOrVersion - If an object does not exist, server MUST create it as a new object. If an object already exists, server MUST not alter the existing object and instead it MUST create a new version of the existing object using the state of the submitted object
  - CreateOnly - If an object already exists, the server MUST return an ObjectExistsException fault message

4.1.1.3 id and lid Requirements

Table 4 defines the requirements for id and lid attribute values for RegistryObjectType instances that are submitted via the SubmitObjects protocol.

Mode / Requirements	ID Requirements	LID Requirements
CreateOrReplace	• MUST be specified by client or else server MUST return InvalidRequestException • If id does not exists, server MUST create new object using that id (create) • If id exists, server MUST replace existing object matching that id (update)	• MUST be specified by client or else server MUST return InvalidRequestException
CreateOrVersion	• MUST be specified by client or else server MUST return InvalidRequestException • If id does not exists and lid does not exist, server MUST create new object using that id (create) • If id does not exists and lid exists, server MUST throw InvalidRequestException (otherwise multiple root level versions would become possible) • If id exists, server MUST create a new version of existing object matching that id (version)	• MUST be specified by client or else server MUST return InvalidRequestException
CreateOnly	• MAY be specified by client • If unspecified Server MUST generate UUID URN • If id does not exists, server MUST create new object using that id (create) • If id exists, server MUST return ObjectExistsException	• MUST be specified by client or else server MUST return InvalidRequestException • MUST NOT exist or else server MUST return ObjectExistsException

Table 4: Requirements for id and lid During SubmitObjects Protocol

4.1.1.4 Returns

This request returns a RegistryResponse.

4.1.1.5 Exceptions

- A server MUST return an UnsupportedCapabilityException fault message if the request contains a type that is an extension of types defined by ebRIM and if the server cannot support such extension.

4.1.2 RegistryResponseType and RegistryResponse

The RegistryResponseType the base type for all response element. The RegistryResponse element is of type RegistryResponseType and is used as response message for several protocols.

4.1.2.1 Syntax

```
<complexType name="RegistryResponseType">
  <complexContent>
    <extension base="rim:ExtensibleObjectType">
      <sequence>
        <element name="Exception" type="tns:RegistryExceptionType"
          minOccurs="0" maxOccurs="unbounded"/>
        <element name="RegistryObjectList" type="rim:RegistryObjectListType"
          minOccurs="0" maxOccurs="1"/>
      </sequence>
      <attribute name="status" type="rim:objectReferenceType" use="required"/>
      <attribute name="requestId" type="anyURI" use="optional"/>
    </extension>
  </complexContent>
</complexType>
<element name="RegistryResponse" type="tns:RegistryResponseType"/>
```

4.1.2.2 Description

- Element Exception – One or more Exception elements are used to represent exception encountered during a request that allows areturn status of PartialSuccess
- Element RegistryObjectList – This element is used to return a list of RegistryObjectType instances if specified by the request.
- Attribute requestId – This attribute contains the if of the request that generated this response. It is used for correlating responses to requests
- Attribute status - This attribute is used to indicate the status of the request. The value of the status attribute MUST be a reference to a ClassificationNodeType instance within the canonical ResponseStatusType ClassificationScheme. A server MUST support the status types as defined by the canonical ResponseStatusType ClassificationScheme. The canonical ResponseStatusType ClassificationScheme may be extended by adding additional ClassificationNodes to it.

  The following canonical values are defined for the ResponseStatusType ClassificationScheme:

  - ~~***Success*** - This status specifies that the request was successful.~~
  - ***Failure*** - This status specifies that the request encountered a failure. This value MUST never be returned since a server MUST indicate failure conditions by returning an appropriate fault message.
  - ***PartialSuccess -*** This status specifies that the request was partially successful. Certain requests such as federated queries allow this status to be returned.
  - ***Success*** - This status specifies that the request was successful.
  - ***Unavailable*** – This status specifies that the response is not yet available. This may be the case if this RegistryResponseType represents an immediate response to an asynchronous request where the actual response is not yet available.
-

- ~~The id and lid MUST match the id and lid for an existing object within the server. A server MUST return an ObjectNotFoundException if this rule is violated by client.~~

- ~~The following rules apply to the processing of SubmitObjectsRequest when the mode is "CreateOrReplace":~~

~~The client MUST specify a value for the id attribute. A server MUST return an InvalidRequestException if this rule is violated by client.":~~

~~The client MUST specify a value for the id attribute. A server MUST return an InvalidRequestException if this rule is violated by client.ReplaceOnly~~

~~The following rules apply to the processing of SubmitObjectsRequest for all values of the mode attribute:~~

- ~~The client MUST specify a value for the lid attribute. A server MUST return an InvalidRequestException if if this rule is violated by client.~~

~~The following rules apply to the processing of SubmitObjectsRequest when the mode is "CreateOnly":~~

- ~~There MUST NOT be any existing object in the server with the same lid attribute value as a submitted object. A server MUST return an ObjectExistsException if this rule is violated by client.~~

- ~~The client MAY specify a value for the id attribute.~~

    - ~~If id attribute value is specified it MUST NOT be the same as id attribute value of an existing object in the server. A server MUST return an ObjectExistsException if this rule is violated by client.~~

    - ~~If id attribute value is unspecified then the server MUST set it to a server generated universally unique id value. A server generated universally unique id value MUST conform to the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID]. An example of such a value is:~~

      ~~urn:uuid:a2345678-1234-1234-123456789012~~

    - ~~If id is unspecified for a RegistryObjectType instance then that instance cannot be the target of a reference by other objects in same SubmitObjectsRequest.~~

~~The following rules apply to the processing of SubmitObjectsRequest when the mode is " via SubmitObjects protocol. or replaced the requirements for id and lid attribute values for RegistryObjectType instances that are submittedThis section specifies~~

~~id and lid Requirements~~

4.1.3 Audit Trail Requirements

A server MUST create AuditableEvents *after* successfully processing a SubmitObjectsRequest within the same 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*~~Updated for all the RegistryObjects ~~replaced~~updated during processing of a SubmitObjectsRequest.

4.1.4 Sample SubmitObjectsRequest

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

```
<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 InvalidRequestException fault message if the client attempts to update the id, lid or objectType attribute of a RegistryObject.

Illustration 3: UpdateObjects Protocol

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~~node within a target object, delete an existing ~~fragment~~node from a target object or update an existing ~~fragment~~node within the target object. A node is defined in the context of the UpdateObjects protocol to be an XML element or an attribute.

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

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

4.2.1.1 Syntax

```
<element name="UpdateObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <!-- Query and ObjectRefList select objects to update -->
          <element name="Query" type="rim:QueryType" minOccurs="0" maxOccurs="1" />
          <element name="ObjectRefList" type="rim:ObjectRefListType"
            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 Query - 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 server. A server MUST use all objects that are referenced by this element in addition to any other objects identified by the Query 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

- A server MUST return an UnsupportedCapabilityException fault message if the request contains a type that is an extension of types defined by ebRIM and if the server cannot support such extension.

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="ValueHolder" 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 fragmentnode from
  target document.
  -->
  <attribute name="mode" use="required">
    <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~~node 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 QueryException 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. The scope of the XML document that is processed by the XPATH expression is the RegistryObjectType instance. A server MUST implicitly support the standard namespace prefixes used by RegRep schemas (rim:, query:, rs:, lcm:, spi:) as a notational convenience. These standard namespace prefixes should map to the latest version of the specification supported by the server.

An XPATH selector expression may select an attribute or an element. If it selects an attribute then the ValueHolder element should use a ValueType subtype for a primitive type (instead of AnyValueType) that corresponds to the primitive type for the attribute (e.g. StringValueType). The ValueHolder/Value element's content shall contain the attribute value.

- Element ValueHolder - 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 ~~V~~value provided by ValueHolder MUST be added to the target object. If the selector targets a repeated element (maxOccurs > 1), the ~~fragment~~node MUST be added at the end. If the selector targets a non-repeated element (maxOccurrs = 1) that already exists, the resource MUST generate an InvalidRequestException with a fault detail of ~~Fragment~~NodeAlreadyExistsException. If the selector targets an existing item of a repeated element, the ~~V~~value provided by ValueHolder MUST be added before the existing item.
  - Update – Indicates that the ~~fragment~~node identified by selector MUST be replaced by value ~~the~~by the ~~specified~~ ValueHolder in its place~~. If the selector resolves to nothing then this fragment element does not result in any change to the target object.~~ . If the selector resolves to nothing then there should be no change to the target object.
  - Delete - indicates that the ~~fragment~~node 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

The following example shows an UpdateObjectsRequest which updates the Name element within a Person instance with the Name element specified by the Value element within UpdateAction. The Selector element uses an XPATH expression to select the Name element ~~fragment~~node within Person objects. The target objects that are updated are chosen by the ObjectRefList element. The target objects could also have been chosen by a Query element.

```
<UpdateObjectsRequest ...>
  <ObjectRefList>
    <rim:ObjectRef id="urn:acme:person:Danyal"/>
  </ObjectRefList>
  <UpdateAction mode="Update">
    <Value xsi:type="rim:AnyValueType">
      <rim:Name>
        <rim:LocalizedString xml:lang="en-US" value="Danny"/>
      </rim:Name>
    </Value>
    <Selector xsi:type="rim:StringQueryExpressionType"
      queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:XPath">

      <rim:Value>/rim:Person/rim:Name</rim:Value>
    </Selector>
  </UpdateAction>
</UpdateObjectsRequest>
```

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

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">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element name="ObjectRefList" type="rim:ObjectRefListType"
            minOccurs="0" maxOccurs="1" />
        </sequence>
        <attribute name="deletionScope" type="rim:objectReferenceType"
          use="optional" default="urn:oasis:names:tc:ebxml-
regrep:DeletionScopeType:DeleteAll"/>
      </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 server 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 Query - 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 server. A server MUST remove all objects that are referenced by this element in addition to any other objects identified by the Query element.

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 RemoveObjectsRequest

The following is a sample RemoveObjectsRequest to remove an Object by its id.

```
<lcm:RemoveObjectsRequest ...>
    <lcm:ObjectRefList>
        <ObjectRef id="urn:acme:Person:Danyal"/>
    </lcm:ObjectRefList>
</lcm:RemoveObjectsRequest>
```

5 Version Control

This section describes the version control features of the ebXML RegRep. This feature is based upon [DeltaV]. The ebXML RegRep version control feature defines a simplified façade that provides a small subset of [DeltaV] functionality.

Versioning of a RegistryObjectType instance is the process of updating the object in such a way that the original instance remains unchanged while a new instance is created as a new version of the original instance. Any specific version of an object may itself be versioned. Thus in general the versions of an object form a tree structure referred to as the Version Tree for that object.

A *Version Tree* for an object is defined to be a tree structure where:

- The root is the original version
- Each non-root node in the tree is a version of the object
- Each version is created from a parent version represented by the parent node of the node for that version

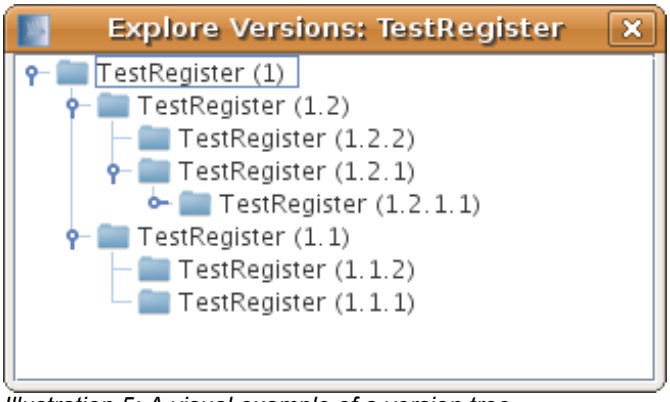

Illustration 5: A visual example of a version tree

Illustration 5 visualizes the version tree concept. In this non-normative example the object TestRegister has 8 versions. Each node's version is identified by the parenthesized string suffix like “(1.2.2)”. Version 1 is the original version. Version 1 was versioned twice to create versions 1.1 and 1.2. Version 1.1 was versioned twice to create versions 1.1.1 and 1.1.2. Version 1.2 was versioned twice to create versions 1.2.1 and 1.2.2. Version 1.2.1 was versioned once to create version 1.2.1.1. Note that this example uses a version naming convention for ease of understanding only. This specification does not prescribe a specific version naming convention for server to use when assigning version names.

The terms “logical object” or “logical RegistryObject” are used to refer to all version of an object in a version independent manner. The terms “object version” or “RegistryObject version” are used to refer to a specific version of the logical object. The terms “RegistryObject instance” and “RegistryObjectType instance” imply a specific object version.

Illustration 5 visualizes a single logical object TestRegister with 8 object versions.

5.1 Version Controlled Resources

Version controlled resources are resources that support versioning capability.

All repository items in an ebXML RegRep are implicitly version-controlled resources as defined by section 2.2.1 of [DeltaV]. No explicit action is required to make them a version-controlled resource.

Instances of RegistryObjectType types are also implicitly version-controlled resources. The only exceptions are those sub-types of RegistryObjectType that are composed[1] types and their instances do not have independent lifecycles from their parent objects. Some example of such composed types are:

- ClassificationType
- ExternalIdentifierType
- ExternalLinkType
- ServiceEndpointType

A server MAY limit specific non-composed types from being version-controlled resources based upon server specific policies.

5.2 Versioning and Id Attribute

Each object version of a logical RegistryObject is a unique object and as such has its own unique value for its id attribute as defined by [ebRIM].

5.3 Versioning and Lid Attribute

A RegistryObject instance MUST have a *Logical ID (LID)* defined by its "lid" attribute to identify the logical RegistryObject of which it is a version. Note that this is in contrast with the "id" attribute that MUST be unique for each version of the same logical RegistryObject. A client may refer to the logical RegistryObject in a version independent manner using its LID.

5.4 Version Identification for RegistryObjectType

A RegistryObjectType instance MUST have a VersionInfo element whose type is the VersionInfoType type defined by ebRIM. The VersionInfo element identifies the version information for that RegistryObjectType instance. The versionName attribute of the VersionInfo element identifies the version name for a specific version of a logical object. A server MUST not allow two versions of the same logical object to have the same versionName attribute value within its VersionInfo element.

5.5 Version Identification for RepositoryItem

When a RegistryObject is an ExtrinsicObject with an associated repository item, the version identification for the repository item is distinct from the version identification for the ExtrinsicObject.

An ExtrinsicObject that has an associated repository item MUST have a contentVersionInfo element whose type is VersionInfoType defined by ebRIM. The contentVersionInfo attributes identifies the version information for that repository item instance.

5.5.1 Versioning of RegistryObjectType

This section describes the versioning of all RegistryObjectType types with the exception of ExtrinsicObjectType which is defined in a separate section.

The following rules apply to versioning of all RegistryObjectType instances that are not instances of ExtrinsicObjectType type. It assumes that versioning is enabled for such RegistryObjectType types:

---

[1] Composed object types are identified in class diagrams in [ebRIM] as classes with composition or "solid diamond" relationship with a RegistryObject type.

- A server MUST create a new version of a version-controlled, non-composed RegistryObjectType instance when it is updated or replaced.
- A server MUST NOT create a new version of a composed RegistryObjectType instance when it is updated or replaced.
- When creating a new version for a non-composed RegistryObjectType instance, a server MUST create new logical objects for any composed logical objects within the new version of the composed object. Any such new logical object for composed objects MUST have a new server generated universally unique id and lid attribute.

5.5.2 Versioning of ExtrinsicObjectType

The ExtrinsicObjectType type requires special consideration for versioning because it may have an associated RepositoryItem which is versioned independently from the ExtrinsicObjectType instance.

The following rules apply to versioning of ExtrinsicObjectType instances assuming that a server has versioning enabled for the ExtrinsicObjectType type:

- A server MUST create a new version of an ExtrinsicObjectType instance and assign it a new unique versionName within its VersionInfo element when either the ExtrinsicObjectType instance or its RepositoryItem are updated or replaced.
- A server MUST create a new version of the RepositoryItem for an ExtrinsicObjectType instance and assign it a new unique versionName within the ContentVersionInfo element when the RepositoryItem is updated or replaced.
- A server MUST create a new version of an ExtrinsicObjectType instance and assign it a new unique versionName within its VersionInfo element when the previous version had a RepositoryItem and the new version does not have one (RepositoryItem was deleted).
- A server MUST create a new version of an ExtrinsicObjectType instance and assign it a new unique versionName within its VersionInfo element when the previous version did not have RepositoryItem and the new version has one (RepositoryItem was added). In such cases the server MUST also create a new version of the RepositoryItem and assign it a new unique versionName within the ContentVersionInfo element.

5.6 Versioning and References

An object reference from a RegistryObjectType instance references a specific version of the referenced RegistryObjectType instance. When a server creates a new version of a referenced RegistryObjectType instance it MUST NOT move references from other objects from the previous version to the new version of the referenced object. Clients that wish to always reference the latest versions of an object MAY use the “dynamic reference” defined in ebRIM feature to always reference the latest version.

A special case is when a SubmitObjectsRequest contains an object that is being versioned by the server and the request contains other objects that reference the object being versioned. In such case, the server MUST update all references within the submitted objects to the object being versioned such that those objects now reference the new version of the object being created by the request.

5.7 Versioning of RegistryPackages

When a server creates a new version of a RegistryPackageType instance, it MUST implicitly make all members of the old version also be members of the new version. This requires that the server MUST make a copy of all HasMember Associations in which the old version of the RegistryPackage is the sourceObject as follows:

- The copied Associations MUST be new versions of their original Association (MUST have the same lid)
- The sourceObject of the copied Associations MUST be reference the new version of the RegistryPackage rather than the older version

5.8 Versioning and RegistryPackage Membership

A RegistryPackage MUST NOT contain more than version of the same logical object as its member.

- A server MUST return an InvalidRequestException fault message if a client attempts to publish more than one version of the same logical object as member of the same RegistryPackage instance

5.9 Versioning and Audit Trail

The canonical EventType ClassificationScheme used by the Audit Trail feature defines an Updated event type and then defines a Versioned event type as a child of the Updated event type ClassificationNode. The semantic are that a Versioned event type is specialization of the Updated event type.

The following rules apply when creating audit trail:

- A server MUST use the Updated event type in the AuditableEvent when it updates a RegistryObject without creating a new version.
- A server MUST use the Versioned event type in the AuditableEvent when it creates a new version of a logical RegistryObject.
- A server MUST NOT use the Created event type in the AuditableEvent when it creates a new version of a logical RegistryObject.

5.10 Inter-versions Association

Each RegistryObject node in the version tree of a logical object except for the root version MUST be linked to the RegistryObject node in the version tree that was its immediate predecessor (previous version).

-
- ~~Sometimes it may be necessary to explicitly indicate which version supersedes another version for the same object. This is especially true when two versions are siblings branch roots of the version tree for the same object.Within any single branch within the version tree for an object any given version implicitly supersedes the version immediately prior to it.~~ A ~~client~~server ~~MAY~~MUST automatically link each new version in the version tree for a RegistryObject to its predecessor ~~specify an~~using an Association between the two versions.
- The type attribute value of the Association MUST reference the canonical AssociationType "Supersedes"
- ~~two versions of an object within the objects version tree using the canonical AssociationType "Supersedes" to indicate that the any~~The sourceObject ~~supersedes the target targetObject within the Association.~~attribute value of the Association MUST reference the new version
- The targetObject attribute value of the Association MUST reference the old version~~A client MUST NOT specify an Association between two versions of an object using the canonical~~

~~AssociationType "Supersedes" if the sourceObject is an earlier version within the same branch in the version tree than the targetObject as this violates the implicit "Supersedes" association between the two version.~~

-

Note that this section is functionally equivalent to the predecessor-set successor-set elements of the Version Properties as defined by [DeltaV].

5.11 Version Removal

Specific versions of a logical object MAY be deleted using the RemoveObjects protocol by specifying the version by its unique id.

- A server MAY allow authorized clients to remove specified versions of a RegistryObject
- A server MAY prune older versions of RegistryObjects based upon server specific administrative policies in order to manage storage resources
- When a non-leaf version within a version tree is deleted, a server MUST implicitly delete the entire version sub-tree under that non-leaf version such that no versions created directly or indirectly from the specified remain in the registry

5.12 Locking and Concurrent Modifications

This specification does not define a workspace feature with explicit checkin and checkout capabilities as defined by [DeltaV]. A server MAY support such features in an implementation specific manner.

This specification does not prescribe a locking or branching model. An implementation may choose to support an optimistic (non-locking) model. Alternatively or in addition, an implementation may support a locking model that supports explicit checkout and checkin capability. A future specification may address these capabilities.

5.13 Version Creation

The server manages creation of new version of a version-controlled resource automatically. A server that supports versioning MUST implicitly create a new version for the resource if an existing version of the resource is updated via a SubmitObjectsRequest or UpdateObjectsRequest when the mode attribute value is CreateOrVersion. A server MUST update the existing version of a resource without creating a new version when the mode attribute is set to CreateOrReplace.

6 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.

6.1 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.

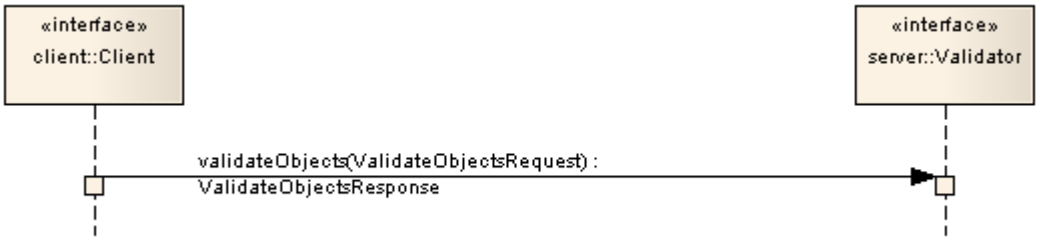

Illustration 6: ValidateObjects Protocol

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.

6.1.1 ValidateObjectsRequest

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

6.1.1.1 Syntax

```
<element name="ValidateObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element name="ObjectRefList" type="rim:ObjectRefListType"
            minOccurs="0" maxOccurs="1" />
          <element name="OriginalObjects" type="rim:RegistryObjectListType"
            minOccurs="1" maxOccurs="1"/>
          <element name="InvocationControlFile"
            type="rim:ExtrinsicObjectType"
            minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

6.1.1.2 Example

The following example shows a client request to validate a specified WSDL file. It assumes that the server will be configured with a Validator plugin for WSDL files. It also assumes that the server will specify OriginalObjects and InvocationControlFile elements when it relays the request to the appropriate Validator plugin.

```
<spi:ValidateObjectsRequest ...>
  <spi:ObjectRefList>
    <rim:ObjectRef id="urn:acme:wsdl:purchaseOrder.wsdl"/>
  </spi:ObjectRefList>
</ValidateObjectsRequest>
```

6.1.1.3 Description

- 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.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the server. 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 Query - Specifies a query to be invoked. A server MUST validate all objects that match the specified query.

6.1.1.4 Response

This request returns ValidateObjectsResponse as response.

6.1.1.5 Exceptions

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

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

6.1.2 ValidateObjectsResponse

Currently ValidateObjectsResponse is a simple extension to RegistryResponseType and does not define additional attributes or elements.

7 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.

7.1 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.

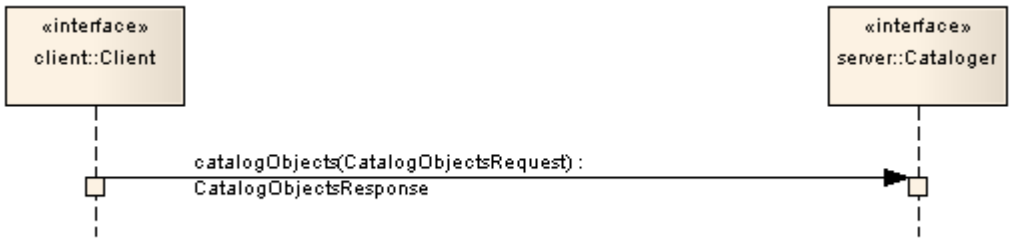

Illustration 7: CatalogObjects Protocol

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

7.1.1 CatalogObjectsRequest

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

7.1.1.1 Syntax

```
<element name="CatalogObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element name="ObjectRefList" type="rim:ObjectRefListType"
            minOccurs="0" maxOccurs="1" />
          <element name="OriginalObjects" type="rim:RegistryObjectListType"
            minOccurs="0" maxOccurs="1"/>
          <element name="InvocationControlFile"
            type="rim:ExtrinsicObjectType"
            minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

7.1.1.2 Example

The following example shows a client request to catalog a specified WSDL file. It assumes that the server will be configured with a Cataloger plugin for WSDL files. It also assumes that the server will specify OriginalObjects and InvocationControlFile elements when it relays the request to the appropriate Cataloger plugin.

```
<spi:CatalogObjectsRequest ...>
  <spi:ObjectRefList>
    <rim:ObjectRef id="urn:acme:wsdl:purchaseOrder.wsdl"/>
  </spi:ObjectRefList>
</CatalogObjectsRequest>
```

7.1.1.3 Description

- Element Query - Specifies a query to be invoked. A server MUST catalog all objects that match the specified query. This element MAY be specified by client when sending the request to the server.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the server. A server MUST catalog all objects that are referenced by this element. This element MAY be specified by client when sending the request to the server.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST catalog all objects that are contained in this element. This element MAY be specified by server when sending the request to the Cataloger plugin. It SHOULD NOT be specified by the client.
- 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. This element MAY be specified by server when sending the request to the Cataloger plugin if the Cataloger plugin requires an an invocation control file. It SHOULD NOT be specified by the client.

7.1.1.4 Response

This request returns CatalogObjectsResponse as response.

7.1.1.5 Exceptions

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

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

7.1.2 CatalogObjectsResponse

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

7.1.2.1 Syntax

```
<element name="CatalogObjectsResponse">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryResponseType">
      </extension>
    </complexContent>
  </complexType>
```

```
</element>
```

7.1.2.2 Example

The following example shows a CatalogObjectsResponse sent by a server to the client in response to a CatalogedObjectRequest. It shows that the Cataloger augmented the Original object with a new Slot that catalogs the target namespace used by the WSDL file.

```
<CatalogObjectsResponse status="urn:oasis:names:tc:ebxml-
regrep:ResponseStatusType:Success">
  <RegistryObjectList>
    <rim:RegistryObject xsi:type="rim:ExtrinsicObjectType"
      mimeType="text/xml"
      status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
      objectType="urn:oasis:names:tc:ebxml-
regrep:ObjectType:RegistryObject:ExtrinsicObject:XML:WSDL"
      lid="urn:acme:wsdl:purchaseOrder.wsdl"
      id="urn:acme:wsdl:purchaseOrder.wsdl">
      <rim:Slot
        dataType="urn:oasis:names:tc:ebxml-regrep:DataType:String"
        name="urn:oasis:names:tc:ebxml-
regrep:profile:wsdl:slot:targetNamespace">
        <rim:ValueList>
          <rim:ValueListItem xsi:type="rim:StringValueType">
              <rim:Value>urn:acme:Service:PurchaseOrder</rim:Value>
          </rim:ValueListItem>
        </rim:ValueList>
      </rim:Slot>
      <rim:RepositoryItem>...binary encoded content...</rim:RepositoryItem>
    </rim:RegistryObject>
  </RegistryObjectList>
</CatalogObjectsResponse>
```

7.1.2.3 Description

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

- Element RegistryObjectList (Inherited) – 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.
  - A cataloger MUST create AssociationType instance between the source object for the catalogObjects operation (specified by OriginalObjects element in CatalogRequest) and each of the cataloged RegistryObjectType instances generated by the cataloger. Each such AssociationType instance
    - MUST have its type attribute reference the canonical AssociationType “urn:oasis:names:tc:ebxml-regrep:AssociationType:HasCatalogedMetadata”,
    - MUST have its sourceObject attribute reference the source object for the catalogObjects operation and
    - MUST have its targetObject attribute reference a cataloged RegistryObjectType instance generated by the cataloger.

- A server MUST delete all cataloged metadata generated by a cataloger when the source object is deleted.
- A server MUST update all cataloged metadata generated by a cataloger when the same version of the source object is updated.

8 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 [WSDL2, WSDL1] specification.

A server may implement these interfaces as external web services invoked by the server using [SOAP-MF, SOAP-ADJ] 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.

8.1 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.

8.1.1 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.

8.2 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.

8.2.1 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:Query> 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:Query> 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.

8.2.2 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.

8.3 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.

8.3.1 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:Query> 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:Query> 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.

8.3.2 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.

9 Subscription and Notification

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.

9.1 Server Events

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

9.1.1 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. A server SHOULD perform such pruning by removing the older AuditableEvents first.

9.2 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 based upon the notificationOption within the DeliveryInfo for the subscription.

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

9.3 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.

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

9.3.1 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.

9.3.2 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.

9.3.3 Subscription Expiration

Each subscription MAY define a startTime and endTime attribute which determines the period within which a Subscription is valid. If startTime is unspecified then a server MUST set it to the time of submission of the subscription. If endTime is unspecified then the server MUST assume the subscription is valid at any time since startTime inclusively.

Outside the bounds of the valid period, a Subsription MAY exist in an expired state within the server. A server MAY remove an expired Subscription at any time.

A server 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.

9.3.4 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:QueryType> 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 algorithm illustrated by the following pseudo-code fragment:

```
//Get objects that match selector query
List<RegistryObjectType> objectsOfInterest =
  getObjectsMatchingSelectorQuery(selectorQuery);

if (objectsOfInterest.size() > 0) {

  //Now get AuditableEvents that affected objectsOfInterest
  //MUST not include AuditableEvents that have already been delivered
  //to this subscriber
  List<RegistryObjectType> eventsOfInterest =
    getEventsOfInterest(objectsOfInterest);

  if (eventsOfInterest.size() > 0) {
    //Now create Notification on objectsOfInterest.
    //Notification will include eventsOfInterest that only include objects
    //that are affected by the event and are also in objectsOfInterest
    NotificationType notification = createNotification(
      objectsOfInterest, eventsOfInterest);

    //Now send notification using info in DeliveryInfo
    sendNotification(notification);
  }
}
```

- Objects of interest MUST be those objects that match the selector query for the subscription
- Events of interest MUST be events that have never been delivered in a notification for this subscription
- Events of interest MUST have affected at least one object of interest
- Events of interest MUST have contain all objects of interest (or references to them) that were affected by the event

- Events of interest MUST NOT contain an object or reference to an object that is not an object of interest

9.4 Event Delivery

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

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

A server MUST NOT deliver the same event more than once for the same Subscription to the same endpoint.

9.4.1 Notification Option

A client MAY specify a notificationOption attribute in DeliveryInfo element of a Subscription. The notificationOption attribute specifies how the client wishes to be notified of events. This attribute controls whether the Event within a Notification contains complete RegistryObjectType instances or only ObjectRefType instances. It is defined in detail in ebRIM.

9.4.2 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.

9.4.3 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.

9.4.3.1 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:~~Subscription~~DeliveryInfo:email~~n:~~NotificationFormatter~~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.

9.5 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 or REST. A server MUST implement a NotificationListener interface as an endpoint if it supports the object replication feature as this endpoint will be used by remote servers to deliver Notification of changes to replicated objects.

9.6 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 a Notification message to the NotificationListener endpoint registered within the Subscription for which the Notification is being delivered.

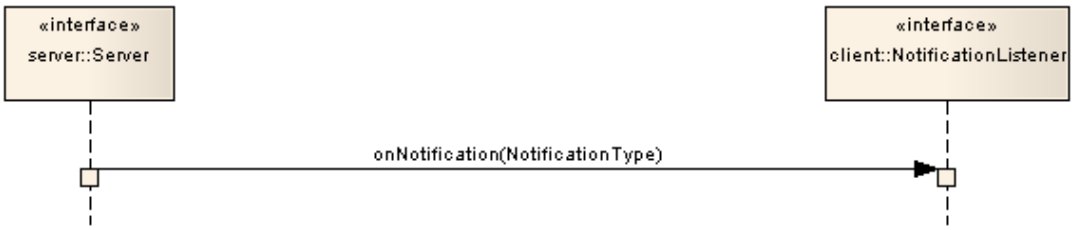

Illustration 8: Notification Protocol

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

9.6.1 Notification

The Notification message is sent by client to the NotificationListener interface deliver an event notification for a subscription. It is a one-way request patteern and produces no response. The syntax and semantics of the Notification message is described in detail in ebRIM.

9.7 Pulling Notification on Demand

A client MAY “pull” Notifications for a Subscription by invoking the GetNotification canonical query. A client MAY specify a startTime since which it wishes to include events within the pulled Notification. If client does not specify a startTime then all events since the last “push” delivery to that client's NotifyTo endpoint will be included in the Notification. If Subscription does not define any “push” delivery for that client's NotifyTo endpoint then a client MUST use startTime parameter to avoid getting the same events within the Notification returned by the GetNotification query.

Pulling a Notification leaves the Notification intact on the server for any potential pushing of the Notification to endpoints defined in DeliveryInfo elements of the Subscription.

9.8 Deleting a Subscription

A client MAY terminate a Subscription with a server 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.

10 Multi-Server Features

This chapter describes features of ebXML RegRep that involve more than one ebXML RegRep server instances. These features include:

- Remote Object Reference – Allows references between objects residing in different servers
- Object Replication – Allows replication of objects residing in a remote server to a local server
- Federated Queries – Allows queries that execute against, and return results from multiple servers

10.1 Remote Objects Reference

A RegistryObject in one ebXML RegRep server MAY contain a reference to a RegistryObject in *any* other ebXML RegRep server that is compatible with ebXML RegRep specifications of the same major version number as ~~the the~~the source server. Remote object reference feature does not require the local and remote servers to be part of the same federation. Remote object refernces are described in detail in [ebRIM].

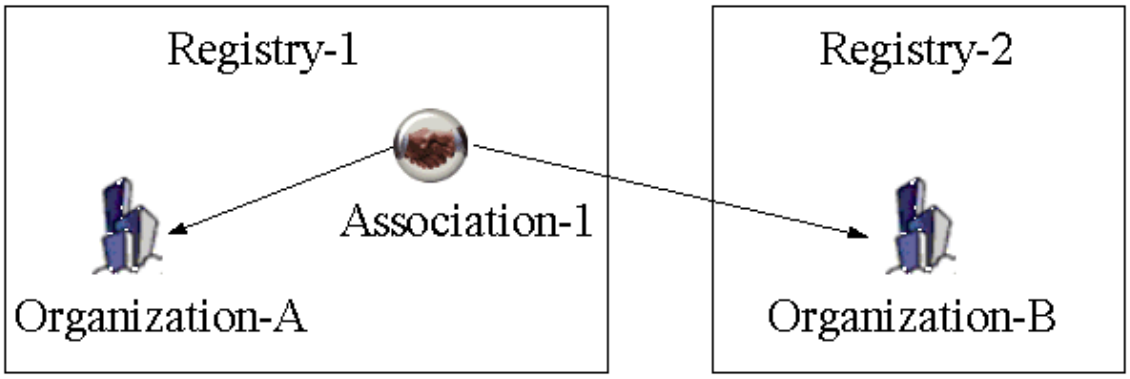

Illustration 9: Remote Object Reference

10.2 Local Replication of Remote Objects

RegistryObjects within a server MAY be replicated in another server. A replicated copy of a remote object is referred to as its replica. The remote object MAY be an original object or it MAY be a replica. A replica from an original is referred to as a first-generation replica. A replica of a replica is referred to as a second-generation replica (and so on).

A server that replicates a remote object locally is referred to as the local server for the replication. The server that contains the remote object being replicated is referred to as the remote server for the replication.

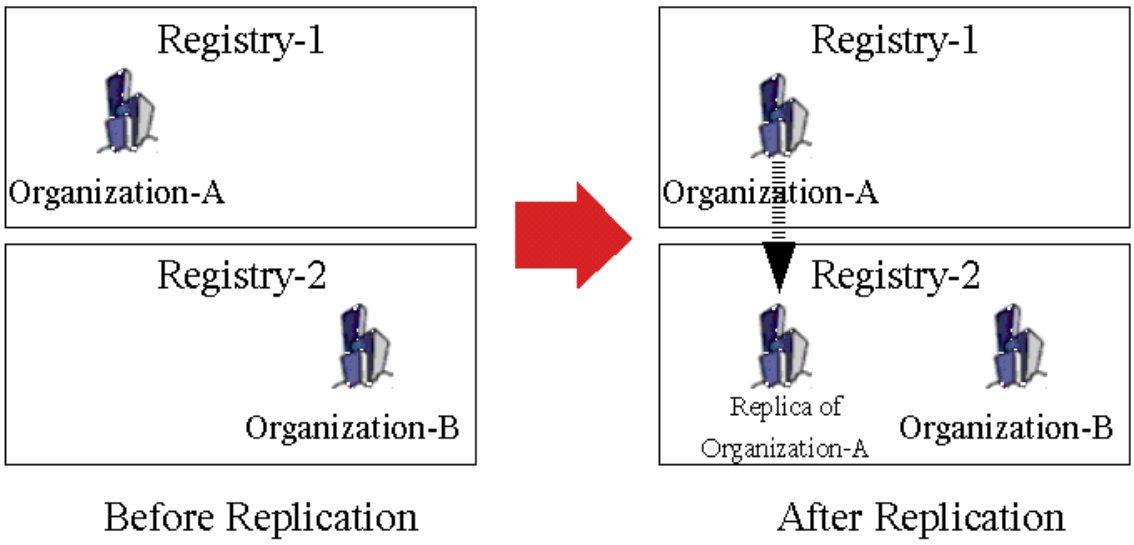

Illustration 10: Local Replication of Remote Objects

The following rules govern replication of remote objects:

- A server MUST treat local replicas of remote objects the same as local objects within the Query protocol.
- A client MUST NOT perform update operations via SubmitObjects and UpdateObjects operations on a local replica of a remote object.
- A server MUST return an InvalidRequestException fault message if a client attempts to update a replica via SubmitObjects and UpdateObjects operations.
- A server MUST delete a replica if a client uses RemoveObjects operation to remove the replica.
- Object replication capability is orthogonal to the server federation capability. Objects MAY be replicated from any server to any other server without any requirement that the registries belong to the same federation.

10.2.1 Creating Local Replica and Keeping it Synchronized

Replication feature is based upon the Subscription and Notification feature. A local replica of a remote objects is created as follows:

- A client submits a Subscription to the remote server on behalf of the local server.
  - The subscription is published like any other RegistryObjectType instance using the Submit Objects protocol with the LifecycleManager endpoint of the remote server.
  - This typically requires that the client is registered with the remote server and can authenticate with it.
- The Subscription defines a Selector query that matches one or more objects that need to be replicated from remote server to local server.
  - Selector query may match any number of objects using any selection criteria supported by the query.

- The Subscription specifies the address of a NotificationListener endpoint implemented by the local server where the remote server may send Notifications regarding the objects that need to be replicated.
- When the remote server send the first Notification to the local server, the local server creates local replicas for the remote objects in the Notification.
  - A server MUST NOT create a local replica for an object if a local object exists with the same id. In such case the server MUST return a ObjectExistsException fault message.
- Whenever the remote server send subsequent Notifications to the local server for the same Subscription, the local server synchronizes the local replica with the remote object.
  - A server MUST delete a local replica when its source object is deleted at the remote server.
  - A server MUST NOT delete a local object whose id matches the id of a remote object when a notification arrives regarding the deletion of the remote object. In such case the server MUST return a InvalidRequestException fault message.

A server MUST use standard QueryManager interface to read the state of a remote object. No new APIs are needed to read the state of a remote object. No prior registration or contract is needed for a server to read the state of a remote object if that object is readable by anyone, as is the case with the default access control policy.

Once the state of the remote object has been read, a server MAY use server specific means to create a local replica of the remote object.

A server MUST set a Slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:home” on a local replica. The value of the Slot MUST be a StringValueType that specifies the base URL of the home server for the remote object that is the source of the local replica. A server MUST NOT set a Slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:home” on a local object within its home server. The presence of this slot distinguished a local replica of a remote object from a local object.

10.2.2 Removing a Local Replica

An authorized client can remove a local replica in the same manner as removal of local objects using the standard RemoveObjects protocol.

10.2.3 Removing Subscription With Remote Server

An authorized client can remove the Subscription at the remote server that was created on behalf of the local server using the standard RemoveObjects protocol.

10.3 Registry Federations

A server federation is a set of ebXML RegRep servers that have voluntarily agreed to form a loosely coupled union. Such a federation may be based on common business interests or membership in a community-of-interest. Registry federations enabled clients to query the content of their member servers using federated queries as if they are a single logical server.

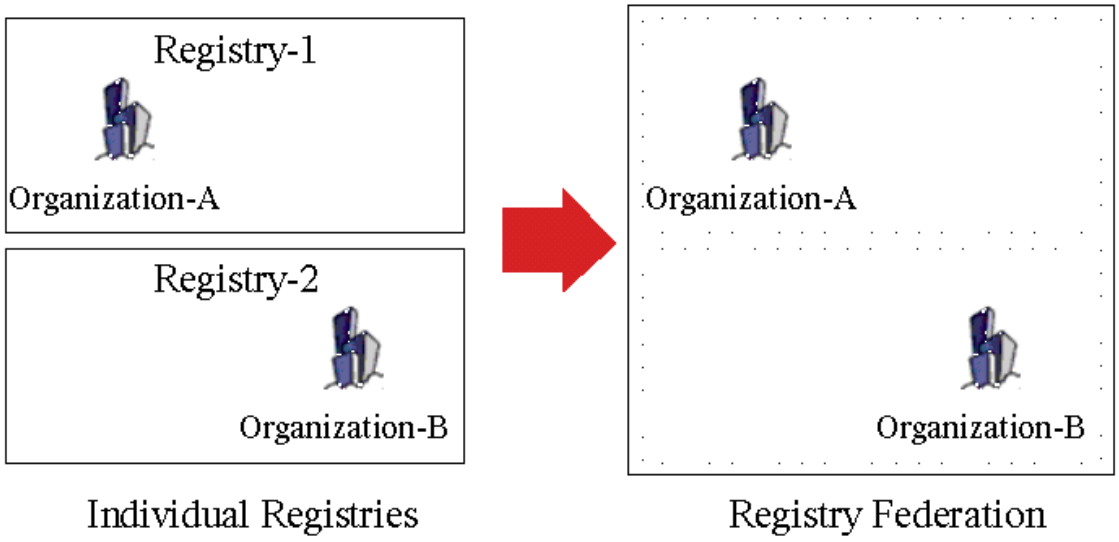

Illustration 11: Registry Federations

10.3.1 Federation Configuration

A deployment MAY configure a set of related ebXML RegRep servers as a Federation using the Registry and Federation classes defined in detail by [ebRIM]. Instances of these classes and the associations between these instances describe a federation and its members.

The Federation information model is summarized here as follows:

- A Federation instance represents a set of ebXML RegRep servers
- A Registry instance represents a server that is a member of the Federation.
- An Association instance with type of *HasFederationMember* represents membership of the server in the federation. This Association links the Registry instance and the Federation instance as shown in illustration below.

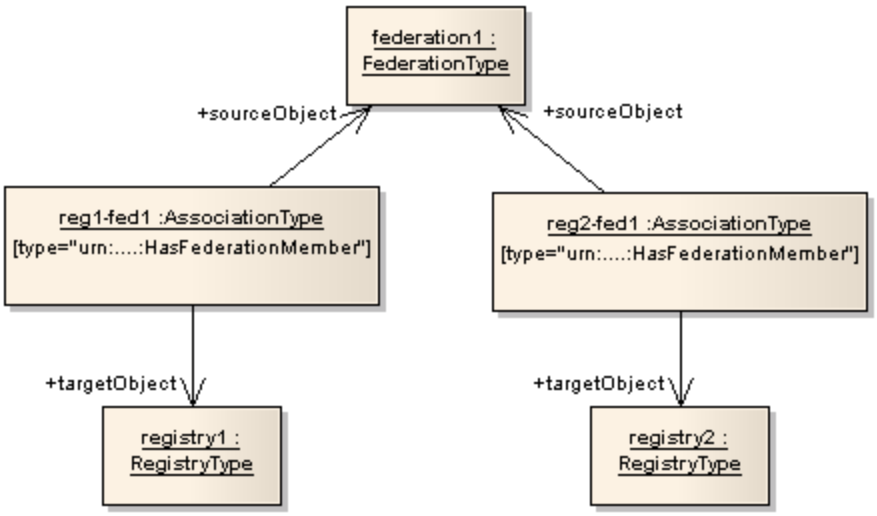

Illustration 12: Federation Configuration Example

10.3.1.1 Creating a Federation

The following rules govern how a federation is created:

- A Federation is created by submitting a Federation instance to a server using the SubmitObjects protocol
- The server where the Federation is created is referred to as the federation home
- A federation home MAY contain multiple Federation instances

10.3.1.2 Joining a Federation

The following rules govern how a server joins a federation:

- Each server SHOULD have exactly one Registry instance within its home server.
- A server's Registry instance SHOULD never change its home server
- A server MAY request to join an existing federation by submitting an instance of an Association that associates the Federation instance as sourceObject, to its Registry instance as targetObject, using a type of *HasFederationMember*. The home server for the Association and the Federation objects MUST be the same.

10.3.1.3 Leaving a Federation

The following rules govern how a server leaves a federation:

A server MAY leave a federation at any time by removing its *HasFederationMember* Association instance that links it with the Federation instance. This is done using the standard RemoveObjects protocol.

10.3.1.4 Dissolving a Federation

The following rules govern how a federation is dissolved:

- A federation is dissolved using the standard RemoveObjects protocol against the Federation's home server and removing its Federation instance
- The removal of a Federation instance is governed by Access Control Policies like any other RegistryObject

10.3.2 Local Vs. Federated Queries

A client MAY query a federation as a single unified logical server. A QueryRequest sent by a client to a federation member MAY be local or federated depending upon the value of the federated attribute of the QueryRequest.

10.3.2.1 Local Queries

When the federated attribute of QueryRequest has the value of *false* (default) then the query is a local query.

A local QueryRequest is only processed by the server that receives the request.

10.3.2.2 Federated Queries

When the *federated* attribute of QueryRequest has the value of *true* then the query is a federated query.

A federation member MUST route a federated query received by it to all other federation member registries on a best attempt basis.

If an exception is encountered while dispatching a query to a federation member ~~is not reachable for any reason then it MAY be silently ignored.~~the server MUST return a QueryResponse as follows:

- The status of the QueryResponse MUST reference the canonical “PartialSuccess” ClassificationNode within the canonical ResponseStatusType ClassificationScheme
- The QueryResponse MUST have a set of Exception sub-elements of tyep rs:RegistryExceptionType, one for each exception encountered while dispatching a query to a remote server

When a server routes a federated query to a federation member that is a RegistryType instance then it MUST set the federated attribute value of the QueryRequest to *false* and the *federation* attribute value to null to avoid infinite loops.

A federated query operates on data that belongs to all members of the target federation.

When a client submits a federated query to a server and no federations exist in the server, then the server MUST treat it as a local query.

The following rules apply to the treatment of iterative queries when the query is federated:

- A server MUST return a result set whose size is less than or equal to the maxResults parameter depending upon whether enough results are available within the scope of servers in the federation, starting at startIndex.
- A server MUST return the same result in a deterministic manner for the same federated QueryRequest if no changes have been made in between the request to the federation member servers and their collective state.
- A server MAY choose any implementation specific algorithm to select results from its federation members for each iteration of an iterative query as long as the algorithm is deterministic and repeatably produces the same results for the same set of federation members and their collective

state. For example a server MAY use a sequential algorithm that gets as many results from each of its server sequentially until it satisfies the maxResults parameter or until there are no more results. Alternatively, a server MAY use a parallel algorithm that balances the amount of data retrieved from each of its federation members.

10.3.3 Local Replication of Federation Configuration

A federation member is required to locally cache the federation configuration metadata in the Federation home server for each federation that it is a member of. A server SHOULD use the replication feature for this.

The federation member MUST keep the cached federation configuration metadata synchronized with the master copy in the Federation home.

10.3.4 Time Synchronization Between Federation Members

Federation members are not required to synchronize their system clocks with each other. However, each Federation member SHOULD keep its clock synchronized with an atomic clock server within the latency described by the replicationSyncLatency attribute of the Federation.

11 Registration Procedures

Issue 96: we need a declarative way to express regproc workflow (e.g. use a ~~BPEL~~BPMN mapped to OWL-S description)??

This chapter describes the registration procedures feature set that supports the governance of RegistryObjects within the server. Registration procedures provide a controlled process for submitting a proposal for changes to a collection of RegistryObjects, review of the change proposal and approval or rejection of the changes proposed. The registration procedures feature set is based upon the concepts defined by [ISO19135].

11.1 Registers and Governance

Governance of RegistryObjects within the server is achieved by a set of governance policies. Governance policies are assigned to a set of RegistryObjects indirectly by assigning them to a specialized RegistryPackage called a Register as defined by [ebRIM]. Such Register policies implicitly govern all members of a Register as well as all descendant members within a Register hierarchy.

The following requirements describe governance of RegistryObjects using registration procedures:

- A RegistryObject MUST be within the membership hierarchy of a Register in order to be governed by registration procedures
- A RegistryObject MUST NOT be within the membership hierarchy of more than one Register. A server MUST return InvalidRequestException if a RegistryObject is published within the membership hierarchy of a new Register when it is already within the membership hierarchy of an existing Register
- A RegistryObject MAY opt out of being governed by registration procedures by not belonging to the membership hierarchy of any Register

11.1.1 Change Proposal Review Process - Overview

Changes to a Register and its members are described by a change proposal and managed by a change proposal process. The change proposal is represented as a new version of a Register that is submitted to a change proposal review process. The change proposal review process consists of a set of activities performed by various persons, services and organizations designated to be responsible for the governance of a Register. These activities are as follows:

- Creating a draft change proposal
- Submitting a change proposal for review and approval
- Accepting a change proposal for review
- Reviewing and subsequently approving or rejecting a change proposal
- Withdrawing a change proposal, ~~F~~fixing issues identified and resubmitting ~~a rejected~~an updated change proposal

A default change proposal review process is defined by this specification. A server MUST support the default change proposal review process. A server MAY support additional change proposal review processes. Error: Reference source not found shows the default change proposal review process.

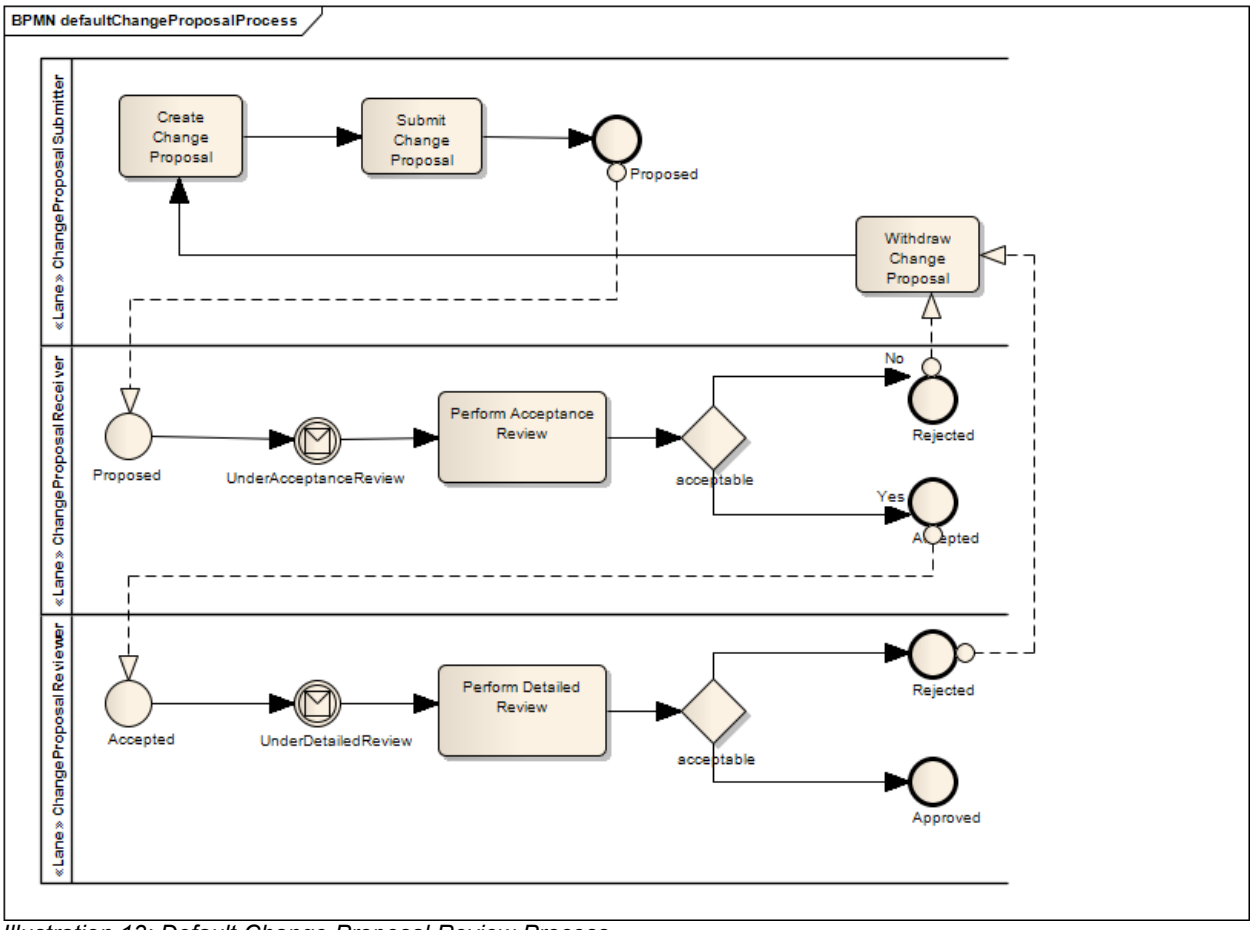

Illustration 13: Default Change Proposal Review Process

~~The change proposal process and its activities are described in detail in subsequent sections.~~

The default change proposal review process and its activities are described in detail in subsequent sections.

11.2 Governance Roles

The governance of a Register is the responsibility of one or more subjects (typically persons ~~and~~or services) that are affiliated with or operated by one or more organizations. This specification describes various governance roles played by organizations, persons and services with respect to a Register.

Illustration 14: Organization and Subject Roles Responsible for Governing a RegisterAssignment of Change Proposal Submitter and Submitting Organization Roles

~~Illustration 16 shows the various register governance roles assigned to organizations, persons and services.~~ These roles are described in detail in subsequent sections and are assigned using the standard SubmitObjects, UpdateObjects registry protocols. ~~Illustration 16 shows the various register governance roles assigned to organizations, persons and services. These roles are described in detail in subsequent sections and are assigned using the standard SubmitObjects, UpdateObjects registry protocols.~~ Note that any role played by a person ~~in the figure m~~may also be played by a service.

11.2.1 Organizational Roles

One or more organizations are responsible for the governance of a Register. Each organization may play one or more roles within the governance process as described in subsequent sections. Each such organization is represented by an OrganizationType instance as defined in [ebRIM].

11.2.1.1 Register Owner

The Register Owner role ~~belongs~~is assigned to the Organization that is ultimately responsible for a Register. Typically this role belongs to the Organization that creates the Register. The Register Owner Organization may choose to play all other organizational roles for the governance of a Register or it MAY delegate some or all of these additional roles to one or more Organizations.

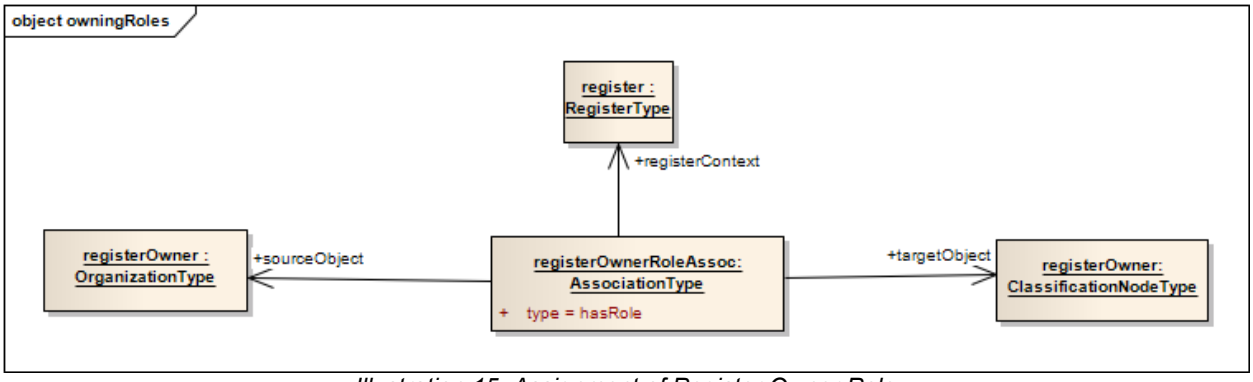

Illustration 15: Assignment of Register Owner Role

The Register Owner role MAY be assigned to a~~Register Register~~n Organization using an Association as follows:

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme
- The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Register Owner organization
- The targetObject attribute value of Association MUST reference ~~rRegistethe~~ the canonical "RegisterOwner" ClassificationNode within the canonical OrganizationRole ClassificationScheme
- The ~~type attribute value of~~ Association MUST specify a Register context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:registerContext which references a Register~~reference the canonical "RegisterOwner" ClassificationNode within the canonical OrganizationRole ClassificationScheme~~
- ~~MUST~~

11.2.1.2 Submitting Organization

The Register Owner for a Register MAY designate one or more Organizations (~~MAY~~ includeing itself) as Submitting Organizations for the Register. A Submitting Organization for a Register is authorized to submit change proposals for the Register.

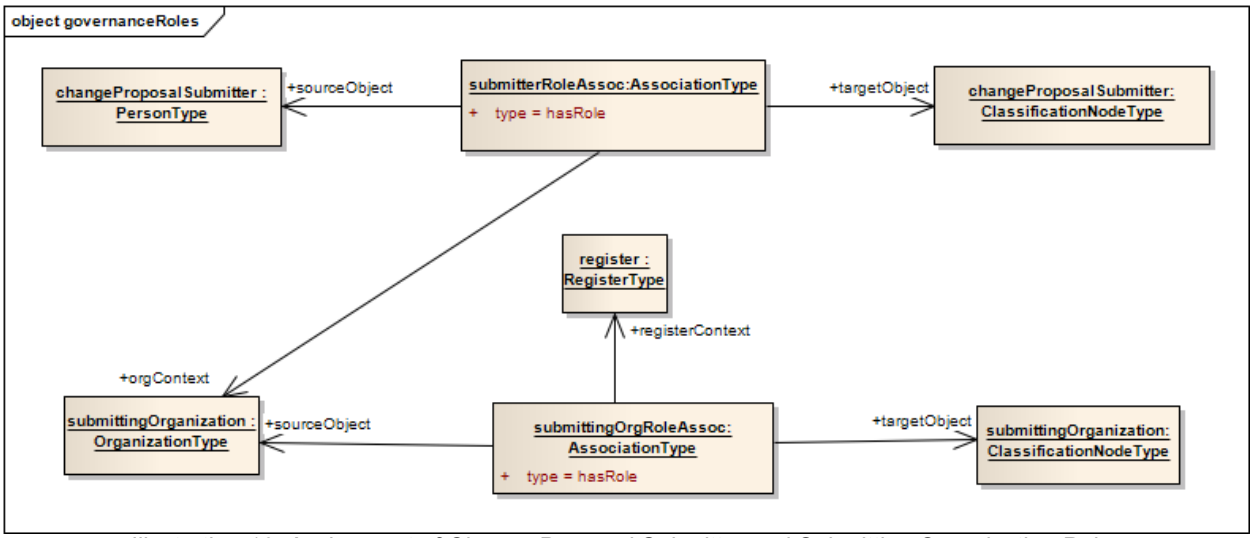

Illustration 16: Assignment of Change Proposal Submitter and Submitting Organization Roles

A Submitting Organization role MAY be assigned to a~~Register~~ n Organization using an Association as follows (illustrated by lower association in Illustration 16):

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme
- The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Submitting Organization

- The targetObject attribute value of Association MUST reference the canonical "SubmittingOrganization" ClassificationNode within the canonical OrganizationRole ClassificationScheme

- The Association MUST specify a Register context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:registerContext which references a Register

- ~~The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Submitting Organization~~

- ~~The targetObject attribute value of Association MUST reference the Register~~

~~The type attribute value of Association MUST reference the canonical "SubmittingOrganization" ClassificationNode within the canonical OrganizationRole ClassificationScheme~~

11.2.1.3 Register Manager

The Register Owner for a Register MUST designate exactly one Organization (MAY designate itself) as the Register Manager for the Register. The Register Manager for a Register receives change proposals for a Register and is responsible for checking the change proposal for validity, completeness and compliance with organizational policies. The Register Manager does not perform a detailed technical review of the change proposal.

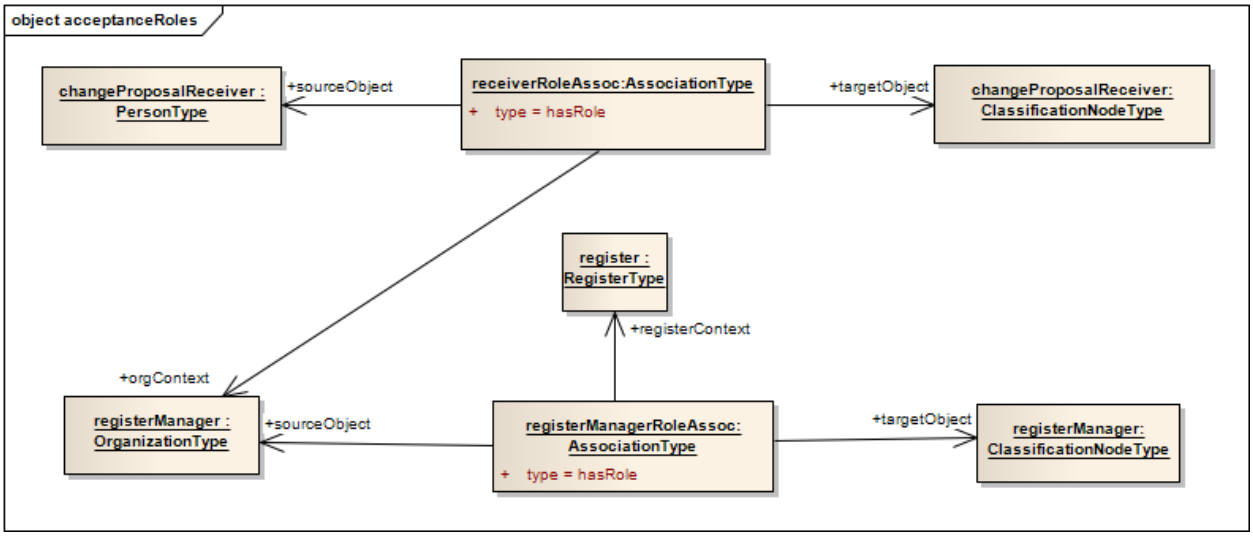

Illustration 17: Assignment of Change Proposal Receiever and Register Manager Roles

The Register Manager role MAY be assigned to a~~Register~~ n Organization using an Association as follows (illustrated by lower association in Illustration 17):

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme

- The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Register Owner organization

- The targetObject attribute value of Association MUST reference the canonical "RegisterManager" ClassificationNode within the canonical OrganizationRole ClassificationScheme

- The Association MUST specify a Register context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:registerContext which references a Register~~The targetObject attribute value of Association MUST reference the RegisterThe type attribute value of Association MUST reference the canonical "RegisterManager" ClassificationNode within the canonical OrganizationRole ClassificationScheme~~

-

~~The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Register Manager~~

-

11.2.1.4 Control Body

The Register Owner for a Register MUST designate exactly one Organization (may designate itself) as the Control Body for the Register. The Control Body for a Register is responsible for doing the detailed technical review of a change proposal once it has been accepted by the Register Manager. The Control Body MUST accept or reject the change proposal or specific changes within the change proposal.

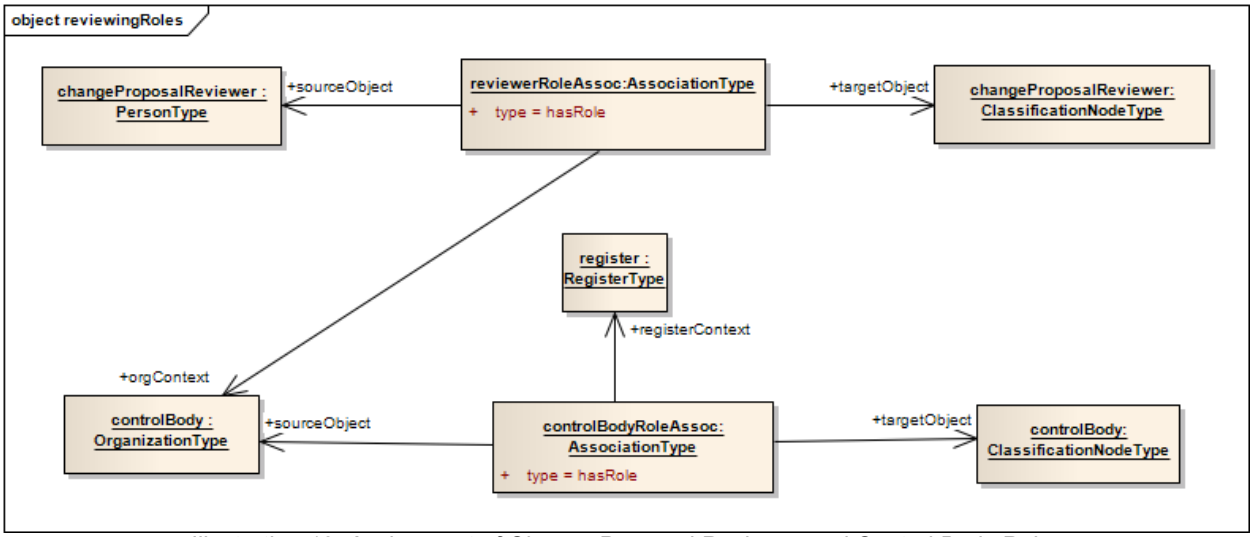

Illustration 18: Assignment of Change Proposal Reviewer and Control Body Roles

The Control Body role MAY be assigned to a~~Register~~ n Organization using an Association as follows (illustrated by lower association in Illustration 18):

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme

- The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Control Body organization

- The targetObject attribute value of Association MUST reference the canonical "ControlBody" ClassificationNode within the canonical OrganizationRole ClassificationScheme

- The Association MUST specify a Register context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:registerContext which references a Register

- ~~The sourceObject attribute value of Association MUST reference the OrganizationType instance representing the Control Body~~
- ~~The targetObject attribute value of Association MUST reference the Register~~

~~The type attribute value of Association MUST reference the canonical "ControlBody" ClassificationNode within the canonical OrganizationRole ClassificationScheme~~

11.2.2 Subject Roles

One or more subjects affiliated with a governing Organization for a Register are responsible for specific governance tasks for a Register. A subject is typically a person though they may also be a service.

Each subject may play one or more roles within the governance process as described in subsequent sections. Each such subject is typically represented by a PersonType or ServiceType instance as defined in [ebRIM].

11.2.2.1 Change Proposal Submitter

The Change Proposal Submitter role is assigned to subjects affiliated with a Submitting Organization for a Register who are authorized to submit change proposals for that register.

The Change Proposal Submitter role MAY be assigned to a subject using an Association as ~~follows~~described below and as shown in upper Association in Illustration 16:

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme
- The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or ServiceType) representing the Change Proposal Submitter
- The targetObject attribute value of Association MUST reference the canonical "ChangeProposalSubmitter" ClassificationNode within the canonical SubjectRole ClassificationScheme
- The Association MUST specify an Organization context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:organizationContext which references an OrganizationType with role Submitting Organization for some Register
- ~~The Association MUST have a canonical slot "urn:oasis:names:tc:ebxml-regrep:rim:Role:context" whose value MUST reference a Submitting Organization for the RegisterThe targetObject attribute value of Association MUST reference the canonical "ChangeProposalSubmitter" ClassificationNode within the canonical SubjectRole ClassificationScheme~~
- ~~The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme~~
- ~~The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or ServiceType)~~
-
-

11.2.2.2 Change Proposal Receiver

The Change Proposal Receiver role is assigned to subjects affiliated with the Register Manager for a Register who are authorized to perform acceptance review for change proposals received for that register. They may accept or reject the change proposal as a whole based upon the findings of the review.

The Change Proposal Receiver role MAY be assigned to a subject using an Association as described below and as shown in upper Association in Illustration 17:

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme
- The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or ServiceType) representing the Change Proposal Receiver
- The targetObject attribute value of Association MUST reference the canonical "ChangeProposalReceiver" ClassificationNode within the canonical SubjectRole ClassificationScheme
- The Association MUST specify an Organization context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:organizationContext which references an OrganizationType with role Register Manager for some Register

~~The Change Proposal Receiver role MAY be assigned to a subject using an Association as follows:~~

- ~~The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or ServiceType)~~
- ~~The targetObject attribute value of Association MUST reference the canonical "ChangeProposalReceiver" ClassificationNode within the canonical SubjectRole ClassificationScheme~~
- ~~The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme~~
- ~~The Association MUST have a canonical slot "urn:oasis:names:tc:ebxml-regrep:rim:Role:context" whose value MUST reference the Register Manager Organization for the Register~~

11.2.2.3 Change Proposal Reviewer

The Change Proposal Reviewer role is assigned to subjects affiliated with the Control Body for a Register who are authorized to perform a detailed technical review of a change proposal once it has been accepted by the Register Manager. They may approve or reject the change proposal as a whole based upon the findings of the review. They may also approve or reject specific changes within a change proposal. A change proposal with any rejected changes is considered to be rejected as a whole.

The Change Proposal Reviewer role MAY be assigned to a subject using an Association as described below and as shown in upper Association in Illustration 18:

- The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme
- The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or ServiceType) representing the Change Proposal Reviewer
- The targetObject attribute value of Association MUST reference the canonical "ChangeProposalReviewer" ClassificationNode within the canonical SubjectRole ClassificationScheme

- The Association MUST specify an Organization context via a Slot name urn:oasis:names:tc:ebxml-regrep:RoleAssociation:organizationContext which references an OrganizationType with role Control Body for some Register

~~The Change Proposal Reviewer role MAY be assigned to a subject using an Association as follows:~~

- ~~The sourceObject attribute value of Association MUST reference the subject instance (typically PersonType or serviceType)~~
- ~~The targetObject attribute value of Association MUST reference the canonical "ChangeProposalReviewer" ClassificationNode within the canonical SubjectRole ClassificationScheme~~
- ~~The type attribute value of Association MUST reference the canonical "HasRole" ClassificationNode within the canonical AssociationType ClassificationScheme~~
- ~~The Association MUST have a canonical slot "urn:oasis:names:tc:ebxml-regrep:rim:Role:context" whose value MUST reference the Control Body Organization for the Register~~

11.3 Default Change Proposal Review Process

This section provides a detailed specification for the default change proposal review process introduced earlier. Note that various activities in the change proposal process uses the standard SubmitObjects, UpdateObjects and RemoveObjects registry protocols. These various activities are also restricted to specific roles as defined by the applicable access control policy.

11.3.1 Creating a Draft Change Proposal

A change proposal describes changes to a Register including the addition, update and removal of RegistryObjects within the membership hierarchy of the Register. An authorized client MAY create a change proposal as follows:

- A change proposal MUST be created by publishing a new version of a Register using the SubmitObjects or UpdateObjects protocol with mode attribute of "CreateOrVersion"
- The VersionInfo/@userVersionName attribute of the Register SHOULD be set by the client to a new value that provides a meaningful name for the new version of the Register
- The status attribute of the newly created version of the Register representing a change proposal MUST reference the canonical "~~Draft~~Submitted" ClassificationNode within the canonical StatusType ClassificationScheme
- New RegistryObjects MAY be added within the draft Register's membership hierarchy of the Register using the SubmitObjects protocol.
- ~~A server MUST NOT permit new members to be added to~~
- ~~A server MUST automatically set the status of such new objects to reference the canonical "Draft" ClassificationNode within the canonical StatusType ClassificationScheme~~
- Existing RegistryObjects within the draft Register's membership hierarchy MAY be updated as new versions of the objects using the SubmitObjects or UpdateObjects protocol with mode attribute of "CreateOrVersion". The status of such new versions MUST reference the canonical "~~Draft~~**Submitted**" ClassificationNode within the canonical StatusType ClassificationScheme
- Existing RegistryObjects within the Register's membership hierarchy MAY be removed from the Register's membership hierarchy. This MUST be done using the RemoveObjects protocol to

remove the HasMember Association between a RegistryObject and its parent RegistryPackageType instance within the Register hierarchy

- A Comment SHOULD be associated with the Register that describes the changes being proposed in narrative form. This comment is used by the Change Proposal Acceptor / Reviewer during the acceptance and review processes. The Comment MUST be attached to the Register as described in [ebRIM].

11.3.2 Submitting a Change Proposal

The change proposal submission activity is performed as follows:

- Based upon the default Register Access Control Policy, the client MUST have the authentication credentials of a subject with ChangeProposalSubmitter role for the Register for which change proposal is being submitted
- ~~Based upon the default Register Notification Policy, a~~ A draft change proposal MUST be submitted for the review by changing the status of the Register version to reference the canonical "Proposed" ClassificationNode within the canonical StatusType ClassificationScheme
- A server MUST return InvalidRequestException if a Request attempts any changes other than status changes to be made to a Register or RegistryObjects in its membership hierarchy unless the status of the Register is Submitted. New versions of the Register MAY be created and new references to the Register MAY be created as they do not change the Register version.
- A client MUST not submit a request that combines status change to the Register with any other changes to the Register attributes and elements. A server MUST return InvalidRequestException if a client request mixes status change with any other changes to the Register attributes and elements.

11.3.3 Accepting a Draft Change Proposal

The change proposal acceptance activity is performed as follows:

- Based upon the **default Register Notification Policy**, when a change proposal is submitted the server MUST send a notification to all subjects that have the role of Change Proposal Receiver within an organization that has the role of Registry Manager for the Register
- Any one of the Change Proposal Receivers receiving the notification of the change proposal MAY perform the activity of acceptance review of the change proposal. This specification does not prescribe how ~~this activity is assigned to a specific subject~~Change Proposal Receivers decide which of them should perform the activity. Any one of them may take on the activity and notify others by changing the status of the Register to reference the canonical "UnderAcceptanceReview" ClassificationNode within the canonical StatusType ClassificationScheme
- The Change Proposal Receiver then reviews the change proposal for compliance with basic acceptance criteria established by the Register Manager. This specification does not prescribe any specific acceptance criteria
- As an outcome of the acceptance review activity, the Change Proposal Receiver MUST either accept it or reject the change proposal in its entirety as follows:
  - Acceptance of the change proposal for review MUST be done by changing the status of the Register version to reference the canonical "~~UnderReview~~Accepted" ClassificationNode within the canonical StatusType ClassificationScheme

- Rejection of the change proposal MUST be done by changing the status of the Register version to reference the canonical “Rejected” ClassificationNode within the canonical StatusType ClassificationScheme. A Comment SHOULD be associated with the Register that describes the reasons for rejection and suggestions for improvement in narrative form. The Comment MUST be attached to the Register as described in [ebRIM]

11.3.4 Reviewing a Draft Change Proposal

The change proposal ~~acceptance~~review activity is performed as follows:

- Based upon the **default Register Notification Policy**, when a change proposal is accepted for review the server MUST send a notification to all subjects that have the role of Change Proposal Reviewer within an organization that has the role of Control Body for the Register
- Any one of the Change Proposal Reviewers receiving the notification of the change proposal acceptance MAY perform the activity of reviewing the change proposal. ~~This specification does not prescribe how this activity is assigned to a specific subject~~This specification does not prescribe how Change Proposal Reviewers decide which of them should perform the activity. Any one of them may take on the activity and notify others by changing the status of the Register to reference the canonical “UnderReview” ClassificationNode within the canonical StatusType ClassificationScheme
- The Change Proposal Reviewer then performs a detailed technical review of the proposed changes. Such review is content and domain-specific. This specification does not prescribe any specific review criteria
- As an outcome of the detailed technical review activity, the Change Proposal Receiver MUST either approve the change proposal in its entirety or approve specific changes in the change proposal individually and reject the change proposal as a whole. This is done as follows:
  - Approval of the entire change proposal MUST be done by changing the status of the Register version to reference the canonical “Approved” ClassificationNode within the canonical StatusType ClassificationScheme
  - When a Register is approved at the Register level, a server MUST implicitly set the status of all RegistryObjects within a Register's member hierarchy to to reference the canonical “Approved” ClassificationNode within the canonical StatusType ClassificationScheme
  - Rejection of the change proposal MUST be done by changing the status of the Register version to reference the canonical “Rejected” ClassificationNode within the canonical StatusType ClassificationScheme.
    - A Comment SHOULD be associated with the Register that describes the reasons for rejection and suggestions for improvement in narrative form. The Comment MUST be attached to the Register as described in [ebRIM].
    - The Change Proposal Receiver SHOULD approve as many of the changes in the change proposal as possible by explicitly setting the status of all RegistryObjects within a Register's member hierarchy to to reference the canonical “Approved” ClassificationNode within the canonical StatusType ClassificationScheme

11.4 Register Policies

This section describes the various policies used to govern a register and specifies a default policy for each category. In the absence of more specific custom policies, a server MUST enforce the default Register policies as specified in this section.

11.4.1 Access Control Policy (ACP)

Access to a Register and the RegistryObjects in its membership hierarchy may be controlled by an access control policy assigned to the Register. The Register access control policy controls access to the Register as well as RegistryObjects in its membership hierarchy.

An access control policy MAY be explicitly assigned to a Register as using the canonical slot "urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:accessControlPolicy" as described in ebRIM.

11.4.2 Default Register Access Control Policy

In the absence of an explicitly assigned access control policy a default access control policy is defined for a Register. ~~The default access control policy for a Register is described as follows:~~

~~It supports all the rules specified for the global default access control policy for RegistryObjects as described in [ebRIM]~~

~~It applies to the Register as well as all RegistryObjects within the Register membership hierarchy~~

- ~~A authenticated client with authentication credentials that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register is allowed to:~~
  - ~~Update or version the Register~~
  - ~~Update, version or remove RegistryObjects from the Registers membership hierarchy~~
- ~~A authenticated client with authentication credentials that are assigned the canonical subject role of ChangeProposalReceiver within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register is allowed to:~~
  - ~~Set the status of the Register to the canonical StatusType of "Accepted" or "Rejected"~~
  - ~~Attach Comments to the Register~~
- ~~A authenticated client with authentication credentials that are assigned the canonical subject role of ChangeProposalReviewer within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register is allowed to:~~
  - ~~Set the status of the Register to the canonical StatusType of "Approved" or "Rejected"~~
  - ~~Set the status of any RegistryObject in the Register's membership hierarchy to the canonical StatusType of "Approved" or "Rejected"~~
  - ~~Attach Comments to the Register~~

~~Attach Comments to any RegistryObject in the Register's membership hierarchy~~Table 5 below summarizes the default access control policy for a Register:

Action / Role	Registry Guest	Registry Administrator	Content Owner	Change Proposal Submitter	Change Proposal Receiver	Change Proposal Reviewer
Accept Change Proposal		✓			✓	
Add Member To Register		✓	✓	✓		
Approve		✓				✓
Create		✓	✓	✓		
Delete		✓	✓	✓		
Deprecate		✓	✓	✓		
Read	✓	✓	✓	✓	✓	✓
Reference		✓	✓	✓	✓	✓
Reject Change Proposal		✓			✓	✓
Submit Change Proposal		✓		✓		
Start Acceptance Review		✓			✓	
Start Detailed Review		✓				✓
Update		✓	✓	✓		
Version		✓	✓	✓		
Withdraw Change Proposal		✓		✓		

Table 5: Default Access Control Policy for Registers

- A server MUST permit an action for each role checked in table above for that action
- A server MUST NOT permit an action for each role not checked in table above for that action

11.4.3 Register Notification Policy

The Register notification policy defines which subjects (typically Person or Service) should be notified when specific events in the lifecycle of a Register and RegistryObjects in its membership hierarchy transpire. Typically a notification policy is defined by publishing a Subscription object. The Register notification policy is used to trigger workflow activities within the change proposal process.

A notification policy MAY be explicitly assigned to a Register as using the canonical slot "urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:notificationPolicy". The value of this slot MUST reference the Subscription object representing the notification policy.

11.4.4 Default Register Notification Policy

In the absence of an explicitly assigned notification policy a default notification policy is defined for a Register. The default notification policy for a Register is described ~~as follows:~~by the following table.

Register Event	Notify Change Proposal Submitter	Notify Change Proposal Receiver	Notify Change Proposal Reviewer
Register version created	✓		
Register version deleted	✓		
Register status set to “Proposed”	✓	✓	
Register status set to “UnderAcceptanceReview”	✓	✓	
Register status set to “Accepted”	✓		✓
Register status set to “Rejected”	✓		
Register status set to “UnderReview”	✓		✓
Register status set to “Approved”	✓		

Table 6: Default Register Notification Policy

- A server MUST deliver a notification for a Register event to each role checked in table above for that Register event
- A server MUST NOT deliver notifications according to the default notification policy if there is an explicit notification policy assigned to the Register
- When delivering notifications, a server MUST deliver an email notification to subjects that are Persons and a web service notification to subjects that are services~~When a Register is created the server MUST send a notification to all subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register~~
- ~~When a Register is deleted the server MUST send a notification to all subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register~~
- ~~When a Register's status is changed to reference the canonical “Proposed” StatusType ClassificationNode the server MUST send a notification to:~~
  - ~~All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register~~
  - ~~All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReceiver within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register~~
- ~~When a Register's status is changed to reference the canonical “UnderReview” StatusType ClassificationNode the server MUST send a notification to:~~

- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register
- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReceiver within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register
- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReviewer within the context of an Organization that has the canonical organization role of ControlBody within the context of the Register

- When a Register's status is changed to reference the canonical "Rejected" StatusType ClassificationNode the server MUST send a notification to:
  - All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register
- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReceiver within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register
- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReviewer within the context of an Organization that has the canonical organization role of ControlBody within the context of the Register
- When a Register's status is changed to reference the canonical "Approved" StatusType ClassificationNode the server MUST send a notification to:
  - All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalSubmitter within the context of an Organization that has the canonical organization role of SubmittingOrganization within the context of the Register
  - All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReceiver within the context of an Organization that has the canonical organization role of RegisterManager within the context of the Register
- All subjects (e.g. Person or Service) that are assigned the canonical subject role of ChangeProposalReviewer within the context of an Organization that has the canonical organization role of ControlBody within the context of the Register
- When a subject has multiple endpoints to receive notifications on as is the case in email endpoints, this specification does not specify how a server determines which endpoint of a subject to send notifications to.

Issue 115: Do we need to specify Validation and Cataloging policies for Registers??

11.4.5 Register Policy Inheritance

A Register's membership hierarchy MAY contain several levels represented by sub-Registers or sub-RegistryPackages. This section specifies how Register policies are inherited by objects within their membership hierarchy. The following requirements are defined for Register policy inheritance:

- A server MUST NOT permit a RegistryObject to be within the membership hierarchy of more than one Registers. This is to avoid policy conflicts that arise if multiple Register policies were to govern a RegistryObject
- A server MUST enforce the Register policies on a RegistryObject that are explicitly defined on their nearest Register ancestor within their Register membership hierarchy

- A sub-Register MAY define an explicit Register policy to override any explicitly defined Register policies it inherits from an ancestor Register. In this case the ancestor Register's policies do not apply to the Register or its descendant RegistryObjects within its subtree of the membership hierarchy

12 Security Features

This chapter describes the security features of ebXML RegRep. A glossary of security terms can be referenced from [RFC 2828]. This specification incorporates by reference the following specifications:

- **[WSS-CORE]** WS-Security Core Specification 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
- **[WSS-UNT]** WS-Security Username Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf
- **[WSS-X509]** WS-Security X.509 Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf
- **[WSS-SAML]** WS-Security SAML Token profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
- **[WSS-KRB]** WS-Security Kerberos Token Profile 1.1, February 2006. http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf

12.1 Message Integrity

A server MUST provide for message integrity to ensure that client requests and server responses are not tampered with during transmission (man-in-the-middle attack).

12.1.1 Transport Layer Security

A server MUST support HTTP/S protocol for *all* ebXML RegRep protocols defined by this specification. HTTP/S protocol support SHOULD allow for both SSL and TLS as transport protocols.

12.1.2 SOAP Message Security

A server MUST support soap message security for *all* ebXML RegRep protocols defined by this specification when those protocols are bound to SOAP.

SOAP message security MUST conform to [WSS-CORE].

The [WSS-CORE] has several profiles for supporting various types of security tokens in a standard manner. A server MUST support at least one of the following types of security token:

- Username tokens as specified by [WSS-UNT]
- X509 Certificate tokens as specified by [WSS-X509T]
- SAML tokens as defined by [WSS-SAMLT]
- Kerberos tokens as specified by [WSS-KRBT]

12.2 Message Confidentiality

A server SHOULD support encryption of protocol messages. as defined section 9 of [WSS-CORE] as a mechanism to support confidentiality of *all* ebXML RegRep protocols defined by this specification when those protocols are bound to SOAP.

12.3 User Registration and Identity Management

A server MUST provide a user registration mechanism to register and manage authorized users of the server. A server MUST also provide an identity management mechanism to register and manage the security tokens associated with registered users. This specification does not define how a server provides user registration and identity management mechanisms.

12.4 Authentication

A server MUST support authentication of the client requests based on the security tokens provided by the client. This specification does not specify the mechanism used by a server to authenticate client requests. Server implementations MAY use any means to provide authentication capability.

12.5 Authorization and Access Control

A server MUST control access by client to resources it manages based upon:

- The access control policy associated with each resource.
- The action the client is performing
- The identity associated with the client as well as any roles assigned to that identity

A server MUST provide an access control and authorization mechanism based upon chapter titled "Access Control Information Model" in [ebRIM]. This model defines a default access control policy that MUST be supported by the server. In addition it also defines a binding to [XACML] that allows fine-grained access control policies to be defined.

12.6 Audit Trail

A server MUST keep a journal or audit trail of all operations that result in changing the state of its resources. This provides a basic form of non-repudiation where a client cannot repudiate that it performed actions that are logged in the Audit Trail.

A server MUST create an audit trail as part of the same transaction as the request that affected to state of server resources. A server MUST create this audit trail using AuditableEvent instances as define by the chapter title "Event Information Model" of [ebRIM].

Details of how a server maintains an Audit Trail of client requests is described in the chapter title "Event Information Model" of [ebRIM].

13 Native Language Support (NLS)

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

13.1 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].

13.2 NLS and Registry Protocol Messages

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

A 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 server 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 client and server 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].

13.3 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>
    <element name="LocalizedString" type="tns:LocalizedStringType"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
```

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

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

Examples of such elements are the “Name” and “Description” elements of the RegistryObject class defined by [ebRIM].

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

The 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 ...>
      <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.

13.3.1 Language of a LocalizedString

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

13.3.2 Character Set of RegistryObject ~~*LocalizedString*~~

The character set used by a ~~locale specific String (LocalizedString)~~ RegistryObjects is defined by the charset attribute within the *Content-Type* mime header for the XML document containing the RegistryObject.~~ as shown below:~~

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

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 server.

13.3.3 ~~Language of *LocalizedString*~~

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

13.4 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.

13.4.1 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"
```

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 server.

13.4.2 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 server. 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 client asking a preferred language for messages from server, could be functionality for a future revision of this document.

14 REST Binding

This chapter specifies a minimal REST binding for the QueryManager interface. This binding will be referred to as Core REST binding. Additional, more detailed REST bindings such as binding for ATOM, ATOM Pub, Open Search etc. will be defined by separate specifications. These additional specification will also provide a RESTFul interface to the LifecycleManager interface.

14.1 Query Protocol REST Binding

A server MUST implement a REST Binding for the Query Protocol of the Query Manager interface as specified in this section. This binding allows a client to invoke any parameterized query supported by the server in a RESTful manner.

The URL pattern or template for the parameterized query invocation is as follows:

```
#Template URL for parameterized query invocation
<server base url>/search?queryId=<the query id>&{<param-name>=<param-value>}*
```

core The following example shows the use of the FindObjectsByIdAndType canonical query using the REST binding.

```
#Get RegistryObject with id: urn:acme:pictures:danyal.jpg
GET http://acme.com/myregistry/search?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectById&id=urn:acme:pictures:danyal.jpg
```

14.1.1 Parameter queryId

The queryId parameter MUST specify the id of a parameterized stored query while zero or more additional parameters MAY provide parameter name and value pairs for parameters supported by the query. If the queryId is unspecified then it implicitly specifies the value “urn:oasis:names:tc:ebxml-regrep:query:FindObjectById” as the default queryId.

14.1.2 Query Specific Parameters

A parameterized query MAY define any number of query specific parameters. A client MAY specify values for these parameters MAY as additional options to the URL. For example, the **id=urn:acme:pictures:danyal.jpg** part in example URL above supplies a value for the id attribute defined by the FindObjectsByIdAndType query.

In addition to query-specific parameters, every query invocation URL MUST also support one or more universal canonical query parameters. These are described in subsequent sections

14.1.3 Canonical Query Parameter: depth

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find objects matching specifies keywords and also return
#related objects reachable by up to 10 levels of references
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&depth=10
```

14.1.4 Canonical Query Parameter: format

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find 10 resources by keywords using en-us language and ebRS format
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-
us&format=application/ebrs+xml
```

14.1.5 Canonical Query Parameter: federated

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Perform a federated query across members of all configured
federations
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true
```

14.1.6 Canonical Query Parameter: federation

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Perform a federated query across members of specified federation
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true&fed
eration=urn:acme:federation:acme-partners
```

14.1.7 Canonical Query Parameter: getRepositoryItem

This canonical query parameter indicates whether the resource addressed by the URL is the repository item associated with the RegistryObjectType instance that matches the specified id. If specified with a value of “true” then the server MUST return the repository item (if any) associated with the RegistryObjectType instance that matches the specified id.

```
//Get RepositoryItem associated with
```

```
//ExtrinsicObjectWith id urn:acme:pictures:danyal.jpg
GET http://acme.com/myregistry/search?
id=urn:acme:pictures:danyal.jpg&getRepositoryItem=true
```

14.1.8 Canonical Query Parameter: matchOlderVersions

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find objects matching specified name and include older versions of
matched objects if they match
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:BasicQuery&name=TestRegister1&matchOlderVersionsOnQuery=true
```

14.1.9 Canonical Query Parameter: startIndex

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find 10 resources by keywords starting at index 30
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10&star
tIndex=30
```

14.1.10 Canonical Query Parameter: lang

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find resources by keywords using en-us language
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-us
```

14.1.11 Canonical Query Parameter: maxResults

This canonical query parameter represents the same named attribute and associated semantics as defined for Query Request.

```
#Example: Find 10 resources by keywords
/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10
```

14.1.12 Query Response

The response document returned by the Query Protocol REST binding MUST be a QueryResponse document. If the format parameter value is unspecified or if it is specified as “application/ebrs+xml” then the response document must have query:QueryResponse element as its root element.

14.2 Canonical URL

The canonical URL is an HTTP GET URL that MAY be used to reference or access RegistryObjectType instance in a RESTful manner. While a RegistryObjectType instance may be accessed via HTTP GET in a variety of ways, the canonical URL provides a simple universally supported means to access the object via HTTP GET. A server MUST provide access to its RegistryObjectType instances and repository items via canonical URLs as defined in sections below. Access to such resources MUST be controlled by the applicable access control policies associated with these resources as defined by ebRIM under chapter titled Access Control Information Model.

14.3 Canonical URL for RegistryObjects

The canonical URL for RegistryObjectType instances conform to the following rules:

- The URL represents the invocation of the canonical GetObjectById query
- The queryId SHOULD NOT be specified
- The id query parameter MUST be specified
- The id query parameter's value MUST be exact and not a pattern containing wildcard characters
- Other query parameters MUST NOT be specified

The following are examples of valid canonical URLs for RegistryObjectType instances. Note that for readability we do not encode special characters in the id attribute value.

```
//Get RegistryObject with associated with
//ExtrinsicObjectWith id: urn:acme:pictures:danyal.jpg
GET http://acme.com/myregistry/search?id=urn:acme:pictures:danyal.jpg

//Get RegistryObject with associated with
//ExtrinsicObjectWith id: http://www.acme.com/pictures/danyal.jpg
GET http://acme.com/myregistry/search?
id=http://www.acme.com/pictures/danyal.jpg
```

14.4 Canonical URL for Repository Items

The canonical URL for repository items conform to the following rules:

- The URL represents the invocation of the canonical GetObjectById query
- The queryId SHOULD NOT be specified
- The id query parameter MUST be specified
- The id query parameter's value MUST be exact and not a pattern containing wildcard characters
- The getRepositoryItem parameter MUST be specified with a value of “true”

- Other query parameters MUST NOT be specified

The following are examples of valid canonical URLs to access Repository Items. Note that for readability we do not encode special characters in the id attribute value.

```
//Get RepositoryItem associated with
//ExtrinsicObjectWith id urn:acme:pictures:danyal.jpg
GET http://acme.com/myregistry/search?
id=urn:acme:pictures:danyal.jpg&getRepositoryItem=true

//Get RepositoryItem associated with
//ExtrinsicObjectWith id http://www.acme.com/pictures/danyal.jpg
GET http://acme.com/myregistry/search?
id=http://www.acme.com/pictures/danyal.jpg&getRepositoryItem=true
```

15 SOAP Binding

This chapter specifies the requirements for SOAP Binding that a regrep server or client must adhere to. The normative definition of service endpoint, protocols and their SOAP binding is contained within the WSDL 1.1 definition avilable at http://www.oasis-open.org/committees/regrep/documents/4.0/ wsdl/1.1. A WSDL 2.0 definition is also available at http://www.oasis-open.org/committees/regrep/documents/4.0/ wsdl/2.0.

The following additional requirements are defined by this specification for the SOAP binding:

- A server MUST use WS-Addressing SOAP Headers when sending a Notification message to a SOAP endpoint as defined here.

15.1 WS-Addressing SOAP Headers

The following rules apply to a server when sending a Notification message to a SOAP endpoint for the NotificationListener.

- Use of WS-Addressing SOAP headers MUST conform to [WSA-SOAP].
- A server MUST set the content of the wsa:MessageID element to a unique id. A server SHOULD generate a universally unique id value that 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).*
- A server MUST set the wsa:ReplyTo SOAP header element
  - The wsa:Address elements content MUST be set to the base URL for the server.
- A server MUST set the content of the wsa:To element to the SOAP endpoint URL where the message is being sent to.
- A server MUST set the content of the wsa:Action element to the value of the soapAction attribute of the soap:operation element for the operation defined for the SOAP binding for the interface's WSDL .

The following example shows a SOAP message containing a Notification intended for a NotificationListener SOAP endpoint.

```
<env:Envelope>
  <env:Header>
    <wsa:MessageID>
    urn:uuid:3e79348f-d696-4fac-a015-a4bae0bf83c5
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://www.acme.com/regrep</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://www.client.com/notificationListener</wsa:To>
    <wsa:Action>urn:oasis:names:tc:ebxml-
regrep:wsdl:NotificationListener:bindings:4.0:NotificationListener:onNotificat
ion</wsa:Action>
  </env:Header>
  <env:Body>
     <rim:Notification .../>
  </env:Body>
</env:Envelope>
```

Appendix A. Acknowledgments

The following individuals have contributed significantly towards the creation of this specification and are gratefully acknowledged

Contributors:

- Rob Atkinson, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia
- Simon Cox, Commonwealth Scientific and Industrial Research Organisation (CSIRO), Australia
- Mark Ford, MIT Lincoln Labs
- Lydia Gietler, Danish Ministry of the Environment
- Brett Levasseur, MIT Lincoln Labs
- Alissandro Triglia, OSS Nokalva
- Aleksei Valikov, Disy Informationssysteme GmbHAlissandro Triglia, OSS Nokalva
-

Appendix B. Revision History

[optional; should not be included in OASIS standards]

Appendix C. Protocol Exceptions

This appendix defines the standard exception that may be returned by various protocols defined in this specification. These exceptions MUST be returned as SOAP fault messages in the SOAP binding for the protocols. Implementations SHOULD provide relevant details regarding the exception within the Detail element of the fault.

XSD Element Name	Description
AuthenticationException	Generated by server when a client sends a request with authentication credentials and the authentication fails for any reason.
AuthorizationException	Generated by server when a client sends a request to the server for which it is not authorized.
CatalogingException	Generated by server when a problem is encountered during the processing of a CatalogObjectsRequest.
InvalidRequestException	Generated by server when a client sends a request that is syntactically or semantically invalid.
ObjectExistsException	Generated by the server when a SubmitObjectsRequest attempts to create an object with the same id as an existing object and the mode is “CreateOnly”.
ObjectNotFoundException	Generated by the server when a QueryRequest expects an object but it is not found in server.
QueryException	Generated by server when when a problem is encountered during the processing of a QueryRequest.
QuotaExceededException	Generated by server when a a request exceeds a server specific quota for the client.
ReferencesExistException	Generated by server when a RemoveObjectRequest attempts to remove a RegistryObject while references to it still exist.
TimeoutException	Generated by server when a the processing of a request exceeds a server specific timeout period.
UnresolvedReferenceException	Generated by the server when a request references an object that cannot be resolved within the request or to an existing object in the server.
UnsupportedCapabilityException	Generated by server when when a request attempts to use an optional feature or capability that the server does not support.
ValidationException	Generated by server when a problem is encountered during the processing of a ValidateObjectsRequest.