OASIS ebXML RegRep Version 4.0 Part 2: Services and Protocols (ebRS)

Committee Specification Draft 01

24 March 2011

Specification URIs:

This Version:

http://docs.oasis-open.org/regrep/regrep-core/v4.0/csd01/regrep-core-rs-v4.0-csd01.odt (Authoritative)
http://docs.oasis-open.org/regrep/regrep-core/v4.0/csd01/regrep-core-rs-v4.0-csd01.pdf
http://docs.oasis-open.org/regrep/regrep-core/v4.0/csd01/regrep-core-rs-v4.0-csd01.html

Previous Version:

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

Latest Version:

http://docs.oasis-open.org/regrep/regrep-core/v4.0/regrep-core-rs-v4.0.odt (Authoritative)
http://docs.oasis-open.org/regrep/regrep-core/v4.0/regrep-core-rs-v4.0.pdf
http://docs.oasis-open.org/regrep/regrep-core/v4.0/regrep-core-rs-v4.0.html

Technical Committee:

OASIS ebXML Registry TC

Chair(s):

Kathryn Breininger, Boeing
Farrukh Najmi, Wellfleet Software

Editor(s):

Farrukh Najmi, Wellfleet Software
Nikola Stojanovic, RosettaNet

Related Work:

This specification replaces or supersedes the OASIS ebXML RegRep 3.0 specifications.

This specification consists of the following documents, schemas, and ontologies:

- Part 0: Overview Document (this document) - provides a global overview and description of all the other parts
- Part 1: Registry Information Model (ebRIM) - specifies the types of metadata and content that can be stored in an ebXML RegRep
- Part 2: Services and Protocols (ebRS) - specifies the services and protocols for ebXML RegRep

- Part 3: XML Schema - specifies the XML Schema for ebXML RegRep
- Part 4: WSDL - specifies the WSDL interface descriptions for ebXML RegRep
- Part 5: XML Definitions - specifies the canonical XML data for ebXML RegRep as well as example XML documents used in the specification

Declared XML Namespace(s):

See Part 0: Overview Document

Abstract:

This document defines the services and protocols for an ebXML RegRep.

A separate document, *OASIS ebXML RegRep Version 4.0 Part 1: Registry Information Model (ebRIM)*, defines the types of metadata and content that can be stored in an ebXML RegRep.

Status:

See Part 0: Overview Document

Citation Format:

When referencing this specification the following citation format should be used:

[regrep-rs-v4.0] *OASIS ebXML RegRep Version 4.0 Part 2: Services and Protocols (ebRS)*. 24 March 2011. OASIS Committee Specification Draft 01. http://docs.oasis-open.org/regrep/regrep-core/v4.0/csd01/regrep-core-rs-v4.0-csd01.odt.

Notices

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

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

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

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

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

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

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

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

Illustration Index

Index of Tables

1 Introduction

All text is normative unless otherwise indicated.

This document specifies the ebXML RegRep service interfaces and the protocols they support. For a general overview of ebXML RegRep and other related parts of the specification please refer to Part 0 [RR-OVERVIEW].

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].

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.

1.1.1 RegistryRequestType

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

1.1.1.1 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>
```

1.1.1.2 Description

- Attribute comment – The comment attribute if specified contains a String that describes the request. A server MAY save this comment within a CommentType instance and associate it with the AuditableEvent(s) for that request as described by [ebRIM].
- 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”.

1.1.2 RegistryResponseType

The RegistryResponseType is the 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.

1.1.2.1 Syntax

```
<complexType name="RegistryResponseType">
  <complexContent>
    <extension base="rim:ExtensibleObjectType">
      <sequence>
        <element name="Exception" type="tns:RegistryExceptionType"
          minOccurs="0" maxOccurs="unbounded"/>
        <element ref="rim:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
        <element ref="rim:ObjectRefList" 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"/>
```

1.1.2.2 Description

- Element ObjectRefList – Contains a sequence of zero or more RegistryObject elements. It is used by requests that return
- Element RegistryObjectList – Contains a sequence of zero or more ObjectRef elements. It is used by requests that return a list of references to RegistryObject instances
- 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. 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:

  - ***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.

1.1.3 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.

1.1.3.1 Syntax

```
<complexType name="RegistryExceptionType">
    <annotation>
        <documentation>Base for all registry exceptions. Based upon
SOAPFault: http://www.w3schools.com/soap/soap_fault.asp</documentation>
    </annotation>
    <complexContent>
      <extension base="rim:ExtensibleObjectType">
        <attribute name="code" type="string" use="optional"/>
        <attribute name="detail" type="string" use="optional"/>
        <attribute name="message" type="string"/>
        <attribute name="severity" type="rim:objectReferenceType"
default="urn:oasis:names:tc:ebxml-regrep:ErrorSeverityType:Error"/>
      </extension>
    </complexContent>
</complexType>
```

1.1.3.2 Description

In addition to the attributes and elements inherited from ExtensibleObjectType this type defines the following attributes and elements:

- Attribute code – The code attribute value may be used by a server to provide an error code or identifier for an Exception.
- Attribute detail – The detail attribute value may be used by a server to provide any detailed information such as a stack trace for an Exception.
- Attribute message – The message attribute value MUST be used by a server to provide a brief message summarizing an Exception.
- Attribute severity – The severity attribute value provides a severity level for the exception. Its value SHOULD reference a ClassificationNode within the canonical ErrorSeverityType ClassificationScheme.

1.2 Server Plugins

Deployments of a 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 as defined by this specification. These standard interfaces are referred to as Service Provider Interfaces (SPI).

Subsequent chapters will specifies various Service Provider Interfaces (SPI) that defines the standard interface for various types of server plugins. These 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 implemented or how it is configured within a server. Nor does it define whether or how, plugin configuration functionality is made discoverable to clients.

2 QueryManager Interface

The QueryManager interface allows a client to invoke queries on the server.

2.1 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 specifying its unique id as well as values for any parameters supported by the query.

A parameterized query MAY be stored in the server as 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.

2.1.1 Invoking Adhoc Queries

A client may invoke a client-specific ad hoc query using a special canonical parameterized query called the 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 query 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.

2.2 Query Protocol

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

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

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

Illustration 1: Query Protocol

2.2.1 QueryRequest

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

2.2.1.1 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>
```

2.2.1.2 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:SlotValue xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <rim:Value>%danyal%</rim:Value>
      </rim:SlotValue>
    </rim:Slot>
  </query:Query>
</query:QueryRequest>
```

2.2.1.3 Description

- Element ResponseOption - This required element allows the client to control the 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 default value is "application/x-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 value 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.

2.2.1.4 Response

This request returns QueryResponse as response.

2.2.1.5 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

2.2.2 Element Query

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

2.2.2.1 Syntax

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

2.2.2.2 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. 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.

2.2.3 Element ResponseOption

A client specifies a ResponseOption structure within a QueryRequest to control the type and structure of results within the corresponding QueryResponse.

2.2.3.1 Syntax

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

2.2.3.2 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 <rim:ObjectRefList> element. 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 <rim:RegistryObjectList> element containing <rim:RegistryObject> elements with xsi:type="rim:RegistryObjectType".
- *LeafClass* - This option specifies that the QueryResponse MUST contain a collection of <rim:RegistryObjectList> element containing <rim:RegistryObject> elements that have an xsi:type attribute that corresponds to leaf classes as defined in [RR-RIM-XSD]. No RepositoryItems SHOULD be included for any rim:ExtrinsicObjectType instance in the <rim:RegistryObjectList> element.
- *LeafClassWithRepositoryItem* - This option is the same as the LeafClass option with the additional requirement that the response include the RepositoryItems, if any, for every rim:ExtrinsicObjectType instance in the <rim:RegistryObjectList> element.

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 a Query that matches rim:OrganizationType instances is asked to return LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume the LeafClass option instead.

2.2.4 QueryResponse

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

2.2.4.1 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>
```

2.2.4.2 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">
  <rim: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>
  </rim:RegistryObjectList>
</query:QueryResponse>
```

2.2.4.3 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.

2.2.5 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.

2.3 Parameterized Query Definition

A parameterized query is defined by submitting a rim:QueryDefinitionType instance to the server using the submitObjects protocol. A detailed specification of the rim:QueryDefinitionType 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.

2.4 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 any 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.

2.4.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
queryExpression	Value is a query expression string in the language specified by the queryLanguage parameter	string		1
queryLanguage	Value is the id of a ClassificationNode within the canonical QueryLanguageScheme ClassificationScheme.	taxonomyElement		1

2.4.2 Query Semantics

- The queryExpression may specify any number of named parameters
- The server MUST use rim:Slot child elements of the rim:Query as named parameters to the query queryExpression
- 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

2.5 Canonical Query: BasicQuery

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

2.5.1 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:RegistryObject[@xsi:type="rim:ClassificationNodeType"]/@path matches specified value When multiple values are specified 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	boolean	false	0..1

	predicates for each parameter			
name	Matches rim:RegistryObject/rim:Name/rim:LocalizedString/@value	string		0..1
objectType	Matches RegistryObjects whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	taxonomyElement		0..1
owner	Matches rim:RegistryObject/@owner. Note that a parameter value of "#@'@#rs:currentUserId()#@'@#" may be used to specify the id of the user associated with the current request	string		0..1
status	Matches RegistryObjects whose status attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	taxonomyElement		0..1

2.5.2 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

2.6 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.

2.6.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
classificationSchemeId	Matches rim:RegistryObject[@xsi:type="rim:ClassificationSchemeType"]/@id. Does not allow wildcards.	string		1

2.6.2 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.

2.7 Canonical Query: FindAssociations

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

2.7.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches Associations whose type attribute references a ClassificationNode where rim:ClassificationNode/@path matches specified value	taxonomyElement		0..1
matchOnAnyParameter	If true then use logical OR between predicates for each parameter	boolean	false	0..1
sourceObjectId	Matches rim:/RegistryObject[@xsi:type="rim:AssociationType"]/@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	taxonomyElement		0..1
targetObjectId	Matches rim:/RegistryObject[@xsi:type="rim:AssociationType"]/@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	taxonomyElement		0..1

	specified value			

2.7.2 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 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

2.8 Canonical Query: FindAssociatedObjects

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

2.8.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches associated RegistryObjects of Association's whose type attribute references a ClassificationNode where rim:ClassificationNode/@path matches specified value	taxonomyElement		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:/RegistryObject[@xsi:type="rim:AssociationType"]/@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	taxonomyElement		0..1
targetObjectId	Matches source RegistryObjects of Associations where the target RegistryObject's id matches rim:/RegistryObject[@xsi:type="rim:AssociationType"]/@targetObject.	string		0..1

	Allows use of “%” wildcard character to match multiple characters. Allows use of “?” wildcard character to match a single character.			
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	taxonomyElement		0..1

2.8.2 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 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

2.9 Canonical Query: GarbageCollector

The canonical query GarbageCollector allows clients to find RegistryObjects that are deemed as garbage by the server.

2.9.1 Parameter Summary

This query specifies no parameters.

2.9.2 Query Semantics

- The server MAY return any objects it considers as garbage or no longer relevant or needed
- The definition of what objects are garbage may be implementation, profile or deployment specific
- The server MUST return the following types of objects
  - Dangling Associations - AssociationType instances that have an unresolvable or null sourceObject or targetObject attribute

2.10 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.

2.10.1 Parameter Summary

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

2.10.2 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

2.11 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.

2.11.1 Parameter Summary

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

	rim:/RegistryObject[@xsi:type="rim:AuditableEventType"]/@timestamp value			

2.11.2 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

2.12 Canonical Query: GetAuditTrailByTimeInterval

The canonical query GetAuditTrailByTimeInterval allows clients to get *all* changes to *all* objects in the server within a specified time interval. This query may be used to keep a client periodically synchronized with changes in the server.

2.12.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
endTime	Specifies the end of the time interval (inclusive) for rim:/RegistryObject[@xsi:type="rim:AuditableEventType"]/@timestamp value	dateTime	5 minutes before current time	0..1
startTime	Specifies the end of the time interval (inclusive) for rim:/RegistryObject[@xsi:type="rim:AuditableEventType"]/@timestamp value	dateTime	Current time	0..1

2.12.2 Query Semantics

- The server MUST return a set of AuditableEvents whose timestamp attribute is within the time interval specified by startTime and endTime parameters. The set is sorted by the timestamp attribute value in descending order (latest first)
- The server MUST only include AuditableEvents whose timestamp is >= startTime parameter value
- The server MUST only include AuditableEvents whose timestamp is <= endTime parameter value

2.13 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.

2.13.1 Parameter Summary

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

2.13.2 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

2.14 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 to query objects hierarchies with parent-child relationships such as the following:

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

2.14.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
depth	Specifies how many levels of descendants to fetch: •depth > 0 implies get descendants upto “depth” levels •depth <= 0 implies get all descendants	integer	1	0..1
exclusiveChildrenOnly	Specifies how to handle children that	boolean	false	0..1

	may have multiple parents: • True value specifies that only children that are not children of any other parent should be returned • false value specifies that children that have other parents should also be matched			
objectType	Specifies the type of object hierarchy for the query	string		0..1
parentId	Specifies the id of the parent object	string		0..1

2.14.2 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 object 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
- If depth parameter is specified then the server MUST also return all descendants upto the specified depth as described by the definition of the depth parameter above
- If exclusiveChildrenOnly is specified with a true value then the server MUST not return any descendants that have multiple parents

2.15 Canonical Query: GetClassificationSchemesById

The canonical query GetClassificationSchemesById allows clients to fetch specified ClassificationSchemes.

2.15.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
id	Matches rim:/RegistryObject[@xsi:type="rim:ClassificationSchemeType"]/@id. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1

2.15.2 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

2.16 Canonical Query: GetRegistryPackagesByMemberId

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

2.16.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
memberId	Matches RegistryPackages that have a RegistryObject as an immediate member where the RegistryObject's id rim:RegistryObject/@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

2.16.2 Query Semantics

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

2.17 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".

2.17.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
subscriptionId	Matches rim:/RegistryObject[@xsi:type="rim:SubscriptionType"]/@id. Wildcards are not allowed.	string		1
startTime	The time since which events should be included in the Notification	xs:dateTime		0..1

2.17.2 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.

2.18 Canonical Query: GetObjectById

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

2.18.1 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

2.18.2 Query Semantics

- The server MUST return the RegistryObjects whose id attribute value matches the specified value of the id parameter.

2.19 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.

2.19.1 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

2.19.2 Query Semantics

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

2.20 Canonical Query: GetReferencedObject

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

2.20.1 Parameter Summary

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

2.20.2 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 invoke the Query within the DynamicObjectRef instance and resolve the reference to the singleton result of the Query and return the matching object.
  - 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 and return the matching object. The id of the remote object is accessible from its canonical URL as the value of the id parameter within the URL.

2.21 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.

2.21.1 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.value	Indexes the value of all slots on all RegistryObjects
repositoryItem	Indexes the text of all text based repository 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

2.21.2 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality

keywords	A space separated list of keywords to search for	string		1

2.21.3 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 beginning 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.

2.22 Canonical Query: 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.

2.22.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
registryPackageIds	A set of IDs of rim:RegistryPackageType instances. Does not allow wildcards.	string		1..*

2.22.2 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.

2.23 Query Functions

A server MAY support any number of 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 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.

2.23.1 Using Functions in Query Expressions

A parameterized query stored as a rim:QueryDefinition instance MAY have a rim:QueryExpression which 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 the 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 be optimized to skip function parsing and evaluation if the special delimiter characters are not present in query expression

The following is an example of a SQL query expression which uses the getClassificationNodes function to match all RegistryObjects that are targets of Association with specified sourceObject and type that is 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 (@# getClassificationNodes("urn:oasis:names:tc:ebxml-
regrep:AssociationType:AffiliatedWith", 0, 2, "false", ",", "${id}") #@) AND
  targetObject.id = a.targetObject@#
```

2.23.2 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 the 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 getClassificationNodes 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:SlotValue xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <rim:Value>urn:test:Person:Danyal</rim:Value>
      </rim:SlotValue>
    <rim:Slot name="types">
      <rim:SlotValue xsi:type="StringValueType"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
        <rim:Value>getClassificationNodes("urn:oasis:names:tc:ebxml-
regrep:AssociationType:AffiliatedWith", 0, 2, "false", ",", "$
{id}")#@@#</rim:Value>
      </rim:SlotValue>
    </rim:Slot>
  </query:Query>
</query:QueryRequest>
```

2.23.3 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:SlotValue) 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

2.23.4 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 : {
<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> | <FLOAT> | <INTEGER> | <BOOLEAN> )
```

2.24 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 SHOULD conform to the solutions defined in this section whenever possible.

2.24.1 Specifying a null Value for string Param or Return Value

A function that accepts a string parameter SHOULD treat a value of “rs: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 “rs:null”.

2.25 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 “rs:” 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.

A client MUST specify the parameters of a function in the same order as specified in the table for the function specification.

Table 1 summarizes the canonical functions defined by this specification.

826

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
getClassificationNodes	Returns all ClassificationNode's that are descendants and / or ancestor of the specified reference ClassificationNode and within the specified number of levels as indicated by the ancestorLevels and descendantLevels parameters.

827 *Table 1: Canonical Functions Defined By This Profile*

828 ## 2.25.1 Canonical Function: currentTime

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

830 ### 2.25.1.1 Function Semantics

831 - The server MUST return a string if the query is processed without any exceptions

832 - The value of the string MUST be current time in ISO 8601 format using the UTC time zone. An
833 example of value returned is “2010-02-25T15:22:14.534Z”.

834 ## 2.25.2 Canonical Function: currentUserId

835 This canonical function takes no parameters and returns a string whose value is the id of the user
836 associated with the current RegistryRequest. This specification does not define how user's are managed
837 within the server nor does it define how an id is assigned to a user.

838 ### 2.25.2.1 Function Semantics

839 - The server MUST return a string if the query is processed without any exceptions

840 - The value of the string MUST be “rs:null” if no current user is associated with the
841 RegistryRequest

842 ## 2.25.3 Canonical Function: relativeTime

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

845 ### 2.25.3.1 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

2.25.3.2 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”

2.25.4 Canonical Function: getClassificationNodes

This canonical function takes a reference ClassificationNode's id as parameter and returns all ClassificationNode's that are descendants and/or ancestors of the specified reference ClassificationNode and within the specified number of levels as indicated by the ancestorLevels and descendantLevels parameters.

2.25.4.1 Parameter Summary

Parameter	Description	Data Type
nodeId	Specifies the id of the reference ClassificationNodeType instance	string
ancestorLevels	Specifies how many levels to match ancestors of reference node	integer
descendantLevels	Specifies how many levels to match descendants of reference node	integer
includeSelf	Specifies whether to include the reference ClassificationNodeType instance or not	boolean
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

2.25.4.2 Function Semantics

- The server MUST return a string if the query is processed without any exceptions
- The string MUST be “rs:null” if no ClassificationNode is found that matches the function parameters
- The string MUST consist of a set of substrings separated by the appropriate delimiter character when any ClassificationNode's are found that match the function parameters:
    - 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

- The id of the reference ClassificationNode MUST be included if and only if the includeSelf parameter value is true
- A ancestorLevels value of N where N > 0 matches all ClassificationNodes upto the Nth level ancestors of the reference ClassificationNode. A value of 1 matches the immediate parents of the reference ClassificationNode while a value of 2 matches the parents and grandparents of the reference ClassificationNode. A value of -1 matches all ancestors of the reference ClassificationNode
- A descendantsLevels value of N where N > 0 matches all ClassificationNodes upto the Nth level descendants of the reference ClassificationNode. A value of 1 matches the immediate children of the reference ClassificationNode while a value of 2 matches the children and grandchildren of the reference ClassificationNode. A value of -1 matches all descendants of the reference ClassificationNode
- A template value of “rs:null” is implicitly equivalent to a template value of “${id}”

2.26 Query Plugins

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

2.26.1 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.

3 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.

3.1 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 a SubmitObjectsRequest message to the LifecycleManager endpoint.

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

Illustration 2: SubmitObjects Protocol

3.1.1 SubmitObjectsRequest

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

3.1.1.1 Syntax

```
<simpleType name="mode">
  <restriction base="NCName">
    <enumeration value="CreateOrReplace"/>
    <enumeration value="CreateOrVersion"/>
    <enumeration value="CreateOnly"/>
  </restriction>
</simpleType>

<element name="SubmitObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element ref="rim:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
        </sequence>
        <attribute name="checkReferences" type="boolean" use="optional"
          default="false"/>
```

```
        <attribute name="mode" type="tns:mode" use="optional"
          default="CreateOrReplace"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

3.1.1.2 Description

- Element RegistryObjectList - Specifies a set of RegistryObject instances that are being submitted to the server. The RegistryObjects in the list may be new objects being submitted to the server or they may be current objects already existing in the server.
- Attribute checkReferences – Specifies the reference checking behavior expected of the server
  - true - Specifies that a server MUST check submitted objects and make sure that all references via reference attributes and slots to other RegistryObjects are resolvable. If a reference does not resolve then the server MUST return UnresolvedReferenceException
  - false (default) – Specifies that a server MUST NOT check submitted objects to make sure that all references via reference attributes and slots to other RegistryObjects are resolvable. If a reference does not resolve then the server MUST NOT return UnresolvedReferenceException
- Attribute mode – 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 does not exist, server MUST create it as a new object. If an object already exists, the server MUST return an ObjectExistsException fault message

3.1.1.3 id and lid Requirements

Table 2 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

959 *Table 2: Requirements for id and lid During SubmitObjects Protocol*

960

961 3.1.1.4 Returns

962 This request returns a RegistryResponse.

963 3.1.1.5 Exceptions

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

3.1.2 Audit Trail Requirements

- The server MUST create a single AuditableEvent object as follows:
  - If RegistryObjects were created by the request, it contain a single Action sub-element with eventType *Created* for all the RegistryObjects created during processing of the request
  - If RegistryObjects were updated by the request, it contain a single Action sub-element with eventType *Updated* for all the RegistryObjects updated during processing of the request
- The server SHOULD create AuditableEvents *after* successfully processing the request in a separate transaction from the request

3.1.3 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>
```

3.2 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 enables *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

3.2.1 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 update action that modifies each target object. Update actions may insert a node within a target object, delete an existing node from a target object or update an existing node within the target object. A node in the context of the UpdateObjects protocol is defined to be an XML DOM node (typically an element or an attribute).

3.2.1.1 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 ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />

          <!-- Specifies how to update selected objects -->
          <element name="UpdateAction" type="tns:UpdateActionType"
           minOccurs="1" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="checkReferences" type="boolean" use="optional"
          default="false"/>
        <attribute name="mode" type="tns:mode" use="optional"
          default="CreateOrReplace"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

3.2.1.2 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

- Attribute checkReferences – Specifies the reference checking behavior expected of the server
  - true - Specifies that a server MUST check updated objects and make sure that all references via reference attributes and slots to other RegistryObjects are resolvable. If a reference does not resolve then the server MUST return UnresolvedReferenceException
  - false (default) – Specifies that a server MUST NOT check updated objects to make sure that all references via reference attributes and slots to other RegistryObjects are resolvable. If a reference does not resolve then the server MUST NOT return UnresolvedReferenceException
- Attribute mode – Specifies the semantics for how the server should handle RegistryObjects being updated in the server:
  - CreateOrReplace (default) - If an object does not exist, server MUST return ObjectNotFoundException. If an object already exists, server MUST update the existing object without creating a new version
  - CreateOrVersion - If an object does not exist, server MUST return ObjectNotFoundException. If an object already exists, server MUST create a new version of the existing object before applying the requested update action
  - CreateOnly – This mode does not apply to UpdateObjectsRequest. If specified, server MUST return an InvalidRequestException

3.2.1.3 Returns

This request returns a RegistryResponse.

3.2.1.4 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.

3.2.2 UpdateAction

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

3.2.2.1 Syntax

```
<complexType name="UpdateActionType">
  <annotation>
    <documentation xml:lang="en">
    </documentation>
  </annotation>
  <sequence>
    <!-- Value for attribute or element -->
    <element name="ValueHolder" type="rim:ValueType"
      minOccurs="0" 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 node 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>
```

3.2.2.2 Description

- Element Selector – Is a QueryExpressionType that contains the expression that identifies a 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 MUST be specified using the RegistryObject being updated as the context node.

  An XPATH selector expression may select an attribute or an element relative to the RegistryObject context node. 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.

- Attribute mode – This attribute specifies the semantics for how the server should update target objects:

  - Insert - Indicates that the value provided by ValueHolder MUST be added to the target object. If the selector targets a repeated element (maxOccurs > 1), the 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 NodeAlreadyExistsException. If the selector targets an existing item of a repeated element, the value provided by ValueHolder MUST be added before the existing item.

- Update – Indicates that the node identified by selector MUST be replaced by value by the ValueHolder in its place. If the selector resolves to nothing then there should be no change to the target object.
- Delete - indicates that the node identified by selector MUST be deleted from the target object if it is present.

3.2.3 Audit Trail Requirements

- The server MUST create a single AuditableEvent object as follows:
  - If RegistryObjects were updated by the request, it contain a single Action sub-element with eventType Updated for all the RegistryObjects updated during processing of the request
- The server SHOULD create AuditableEvents *after* successfully processing the request in a separate transaction from the request

3.2.4 Sample UpdateObjectsRequest

The following example shows an UpdateObjectsRequest which updates the Name element within a PersonType instance with the Name element specified by the Value element within UpdateAction. The Selector element uses an XPATH expression to select the Name element node within the Person objects identified as target of update in the ObjectRefList. The context node of the XPATH expression is the RegistryObject element for the PersonType instance. The target objects could also have been chosen by a Query element.

```
<UpdateObjectsRequest ...>
  <rim:ObjectRefList>
    <rim:ObjectRef id="urn:acme:person:Danyal"/>
  </rim: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:Name</rim:Value>
    </Selector>
  </UpdateAction>
</UpdateObjectsRequest>
```

3.3 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 a RemoveObjectsRequest message to the LifecycleManager endpoint.

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

Illustration 4: RemoveObjects Protocol

3.3.1 RemoveObjectsRequest

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

3.3.1.1 Syntax

```
<element name="RemoveObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
        </sequence>
        <attribute name="checkReferences" type="boolean" use="optional"
          default="false"/>
        <attribute name="deleteChildren" type="boolean" use="optional"
          default="false"/>
        <attribute name="deletionScope" type="rim:objectReferenceType"
          use="optional" default="urn:oasis:names:tc:ebxml-
regrep:DeletionScopeType:DeleteAll"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

3.3.1.2 Description

- Attribute checkReferences – Specifies the reference checking behavior expected of the server
  - true - Specifies that a server MUST check objects being removed and make sure that there are no references to them from other objects via reference attributes and slots. If a reference exists then the server MUST return ReferencesExistsException
  - false (default) – Specifies that a server MUST NOT check objects being removed to make sure that there are no references to them from other objects via reference attributes and slots. If a reference exists then the server MUST NOT return ReferencesExistsException
- Attribute deleteChildren – This attribute specifies whether or not to delete children of the objects being deleted according to the following behavior:
  - false – Specifies the server MUST NOT delete the children of objects that are specified to be deleted

- true – Specifies the server MUST delete children of objects being deleted if and only if those children are not children of any other parent objects

- 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 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.

3.3.1.3 Returns:

This request returns a RegistryResponse.

3.3.1.4 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.

3.3.2 Audit Trail Requirements

- The server MUST create a single AuditableEvent object as follows:

  - If RegistryObjects were removed by the request, it contain a single Action sub-element with eventType Deleted for all the RegistryObjects removed during processing of the request

- The server SHOULD create AuditableEvents *after* successfully processing the request in a separate transaction from the request

3.3.3 Sample RemoveObjectsRequest

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

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

4 Version Control

This section describes the version control features of the ebXML RegRep.

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:

- There is a single root node for the tree
- 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 and is represented in the version tree as a child node of the node representing the parent version node for that version

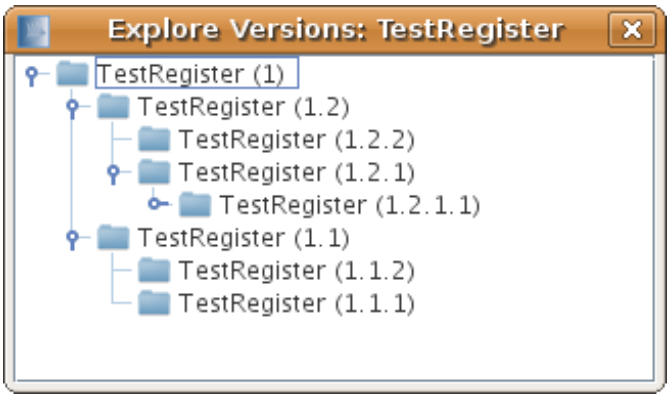


Illustration 5: A visual example of a version tree

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 a RegistryObject 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.

4.1 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 that are separate from the lifecycle of their parent objects. Some example of such composed types are:

- ClassificationType
- ExternalIdentifierType
- ExternalLinkType
- ServiceEndpointType

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

4.2 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].

4.3 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. All versions of a logical RegistryObject have the same "lid" attribute value. 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.

4.4 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.

4.5 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.

4.5.1 Versioning of RegistryObjectType

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

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

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:

- A server MUST create a new version of a version-controlled, non-composed RegistryObjectType instance in the following cases:
  - An existing object is replaced using the submitObjects protocol with mode of CreateOrVersion
  - An existing object is updated using the updateObjects protocol with mode of CreateOrVersion
- A server MUST NOT create a new version of a composed RegistryObjectType instance when it is updated.
- 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.

4.5.2 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 existing ExtrinsicObjectType instance and assign it a new unique versionName within its VersionInfo element when either the ExtrinsicObjectType instance or its RepositoryItem are updated using the submitObjects or updateObjects protocol and the mode is CreateOrVersion
  - 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.
  - A server MUST create a new version of the RepositoryItem for an existing ExtrinsicObjectType instance and assign it a new unique versionName within the ContentVersionInfo element when the RepositoryItem is updated using the submitObjects or updateObjects protocol and the mode is CreateOrVersion

4.6 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.

4.7 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 reference the new version of the RegistryPackage rather than the older version

4.8 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

4.9 Inter-version 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).

- A server MUST automatically link each new version in the version tree for a RegistryObject to its predecessor using an Association between the two versions
- The type attribute value of the Association MUST reference the canonical AssociationType “Supersedes”
- The sourceObject attribute value of the Association MUST reference the new version
- The targetObject attribute value of the Association MUST reference the old version

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

4.10 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

4.11 Locking and Concurrent Modifications

This specification does not define 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 model. An implementation may choose to support a locking model in an implementation specific manner. A future specification may address these capabilities.

4.12 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.

5 Validator Interface

The Validator interface allows the validation of objects published to the server. The interface may be used by clients to validate objects already published to the server or may be used by the server to validate objects during the processing of the submitObjects or updateObjects protocol

A server MUST implement the Validator interface as an endpoint. The Validator interface validates objects using Validator Plugins specific to the type of object being validated.

5.1 ValidateObjects Protocol

The ValidateObjects protocol is initiated by sending an ValidateObjectsRequest message to the Validator endpoint.

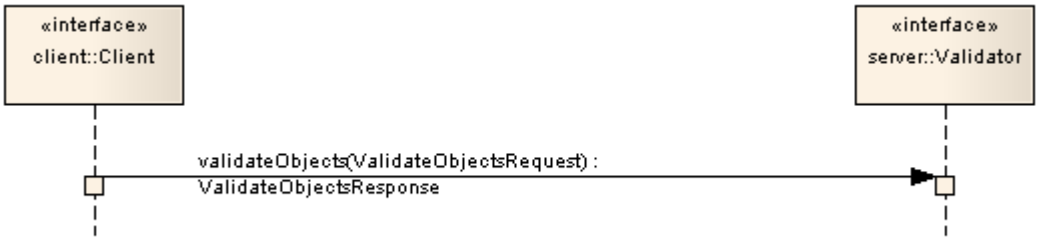


Illustration 6: ValidateObjects Protocol

The Validator endpoint sends an ValidateObjectsResponse back 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.

5.1.1 ValidateObjectsRequest

The ValidateObjectsRequest message initiates the validateObjects protocol and specifies the objects that need to be validated.

5.1.1.1 Syntax

```
<element name="ValidateObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element ref="rim:ObjectRefList" 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>
```

5.1.1.2 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 ...>
  <rim:ObjectRefList>
    <rim:ObjectRef id="urn:acme:wsdl:purchaseOrder.wsdl"/>
  </rim:ObjectRefList>
</ValidateObjectsRequest>
```

5.1.1.3 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. This element MAY be specified by server when sending the request to the Validator plugin if the Validator plugin requires an invocation control file. It SHOULD NOT be specified by the client.
- 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. This element is typically used when a client initiates the validateObjects protocol.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST validate all objects that are contained in this element. This element is typically used when a server initiates the validateObjects protocol during the processing of a submitObjects or updateObjects protocol request or when it is delegating a client initiated validateObjects protocol request to a Validator plugin.
- Element Query - Specifies a query to be invoked. A server MUST validate all objects that match the specified query. This element is typically used when a client initiates the validateObjects protocol.

5.1.1.4 Response

This request returns ValidateObjectsResponse as response.

5.1.1.5 Exceptions

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

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

5.1.2 ValidateObjectsResponse

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

5.2 Validator Plugins

Validator plugins allow a server to use specialized extension modules to validate specific types of objects during the processing of a SubmitObjectsRequest, UpdateObjectsRequest or 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.

5.2.1 Validator Plugin Interface

A Validator plugin implements the Validator interface. The server's Validator endpoint SHOULD 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 <spi: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 <spi: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.

5.2.2 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.

6 Cataloger Interface

The Cataloger interface allows a client to catalog or index objects already in the server. The interface may be used by clients to catalog objects already published to the server or may be used by the server to catalog objects during the processing of the submitObjects or updateObjects protocol .

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 cataloged.

6.1 CatalogObjects Protocol

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

The CatalogObjects protocol is initiated 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.

6.1.1 CatalogObjectsRequest

The CatalogObjectsRequest message initiates the catalogObjects protocol and specifies the objects that need to be cataloged.

6.1.1.1 Syntax

```
<element name="CatalogObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element name="Query" type="rim:QueryType"
            minOccurs="0" maxOccurs="1" />
          <element ref="rim:ObjectRefList" 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>
```

6.1.1.2 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 ...>
  <rim:ObjectRefList>
    <rim:ObjectRef id="urn:acme:wsdl:purchaseOrder.wsdl"/>
  </rim:ObjectRefList>
</CatalogObjectsRequest>
```

6.1.1.3 Description

- Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the cataloging 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.
- 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 is typically used when a client initiates the catalogObjects protocol.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST catalog all objects that are contained in this element. This element is typically used when a server initiates the catalogObjects protocol during the processing of a submitObjects or updateObjects protocol request or when it is delegating a client initiated catalogObjects protocol request to a Cataloger plugin.
- Element Query - Specifies a query to be invoked. A server MUST catalog all objects that match the specified query. This element is typically used when a client initiates the catalogObjects protocol.

6.1.1.4 Response

This request returns CatalogObjectsResponse as response.

6.1.1.5 Exceptions

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

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

6.1.2 CatalogObjectsResponse

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

6.1.2.1 Syntax

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

6.1.2.2 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">
  <rim: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
        name="urn:oasis:names:tc:ebxml-
regrep:profile:wsdl:slot:targetNamespace">
        <rim:SlotValue xsi:type="rim:StringValueType">
          <rim:Value>urn:acme:Service:PurchaseOrder</rim:Value>
        </rim:SlotValue>
      </rim:Slot>
      <rim:RepositoryItem>...binary encoded content...</rim:RepositoryItem>
    </rim:RegistryObject>
  </rim:RegistryObjectList>
</CatalogObjectsResponse>
```

6.1.2.3 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

- MUST have its targetObject attribute reference a cataloged RegistryObjectType instance generated by the cataloger

- A cataloger SHOULD assign the same accessControlPolicy to cataloged objects as their source object. A cataloger MAY use a different strategy for assigning access control policy to cataloged objects.

- 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 source object is updated without creating a new version.

6.2 Cataloger Plugins

Cataloger plugins allow a server to use specialized extension modules to catalog specific types of objects during the processing of a SubmitObjectsRequest, UpdateObjectsRequest or 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.

6.2.1 Cataloger Plugin Interface

A Cataloger plugin implements the Cataloger interface. The server's Cataloger endpoint SHOULD 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 <spi: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 <spi: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.

6.2.2 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 cataloged.

An XSLT file used as InvocationControlFile with the Canonical XML Cataloger MUST meet the following constraints:

- Support an ExtrinsicObject as primary input
- Support an XML RepositoryItem for the ExtrinsicObject object as a secondary input
- The secondary input is specified using an <xsl:param> with name “repositoryItem” and with value that is the id of the ExtrinsicObject for which it is a RepositoryItem

A server MUST implement the Canonical XML Cataloger with the following constraints:

- Uses an XSLT processor with the XSLT file specified as InvocationControlFile
- Specifies the ExtrinsicObject being cataloged as the primary input to the XSLT processor
- Specifies the RepositoryItem for the ExtrinsicObject object being cataloged by setting the parameter named “repositoryItem” with a value that is the id of the ExtrinsicObject for which it is a RepositoryItem
- Resolves references to the RepositoryItem via the $repositoryItem parameter value within the XSLT file specified as InvocationControlFile

7 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.

7.1 Server Events

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

7.1.1 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.

7.2 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].

7.3 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:SubscriptionType instance to the server using the standard SubmitObjects protocol.

Details for the rim:SubscriptionType are defined in [ebRIM].

7.3.1 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.

7.3.2 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 SHOULD return a QuotaExceededException in the event that an authorized user submits more Subscriptions than allowed by their server-specific quota.

7.3.3 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 choose a default value based on its policies.

Outside the bounds of the valid period, a Subscription 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.

7.3.4 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 have affected at least one object of interest
- Events of interest MUST 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

7.4 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].

7.4.1 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.

7.4.2 Delivery to NotificationListener Web Service

If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-regrep:endPointType:soap”, then the server MUST use the specified address as the web service endpoint URL to deliver the Notification to. The target web service in this case MUST implement the NotificationListener interface.

7.4.3 Delivery to Email Address

If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-regrep:endPointType:rest”, then the server MUST use the specified address as the email address to deliver the Notification via email. This specification does not define how a server is configured to send Notifications via email.

7.4.4 Delivery to a NotificationListener Plugin

If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-regrep:endPointType:plugin”, then the server MUST use the specified address as a Notification plugin identifier and deliver the Notification via local call to the plugin. This specification does not define how a server is configured for Notification plugins.

7.4.4.1 Processing Email Notification Via XSLT

A client MAY specify an XSLT style sheet within a DeliveryInfo element to process a Notification prior to it being delivered to an email address. The XSLT style sheet MAY be specified using a Slot in DeliveryInfo element where the Slot's name is “urn:oasis:names:tc:ebxml-regrep:rim:DeliveryInfo:emailNotificationFormatter” 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.

7.5 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.

7.6 Notification Protocol

A server sends a Notification to an endpoint 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.

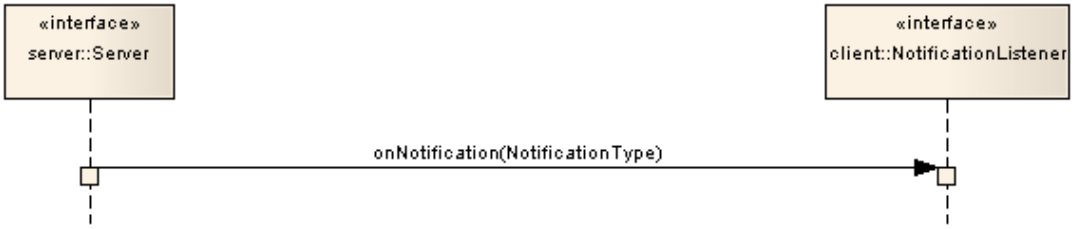


Illustration 8: Notification Protocol

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

7.6.1 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 pattern and produces no response. The syntax and semantics of the Notification message is described in detail in ebRIM.

7.7 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 MUST 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.

7.8 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.

8 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

8.1 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 a compatible version number as the source server. Remote object reference feature does not require the local and remote servers to be part of the same federation. Remote object references are described in detail in [ebRIM].

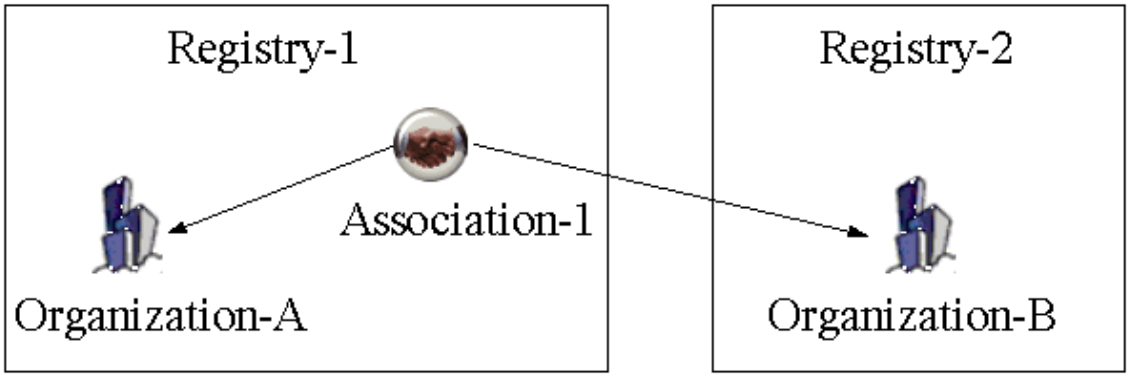


Illustration 9: Remote Object Reference

8.2 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.

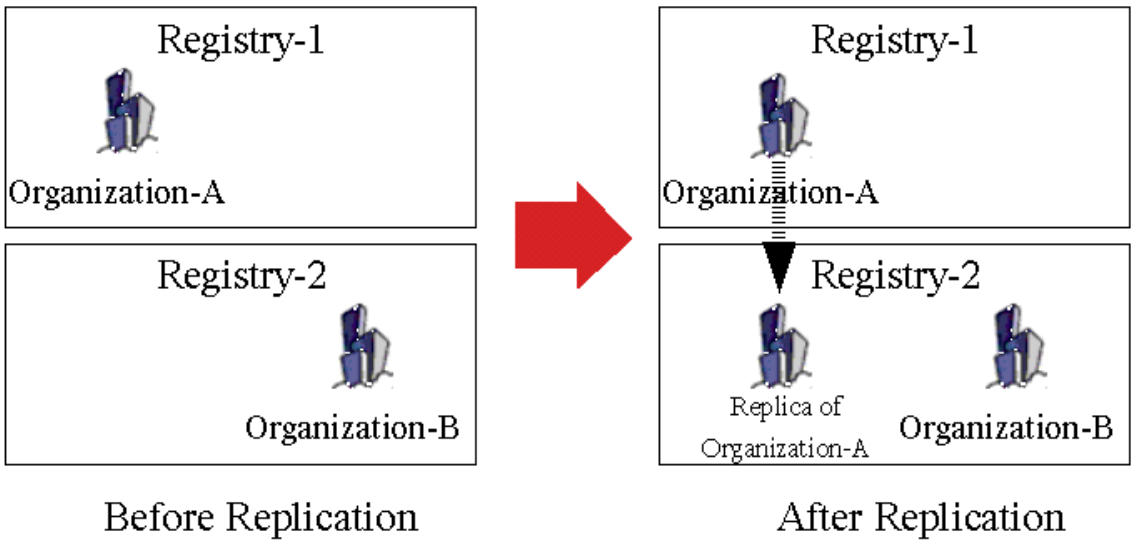


Illustration 10: Local Replication of Remote Objects

The following rules govern replication of remote objects:

- A server MUST match local replicas of remote objects in the same manner 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.
- Objects MAY be replicated from any server to any other server without any requirement that the registries belong to the same federation.

8.2.1 Creating Local Replica and Keeping it Synchronized

Replication feature relies upon the Subscription and Notification feature to keep replicas synchronized with changes to the remote object. 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.

- The local server uses the selector query for the subscription to PULL the initial copy of the remote object(s)
    - 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 an ObjectExistsException fault message.
- Whenever the remote server send 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 that is not a replica of a remote object if a notification arrives regarding the deletion of a remote object with the same id as the local object. In such case the server MUST return an InvalidRequestException fault message.

A server MUST use standard QueryManager interface 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.

8.2.2 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.

8.2.3 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 with the remote server.

8.3 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.

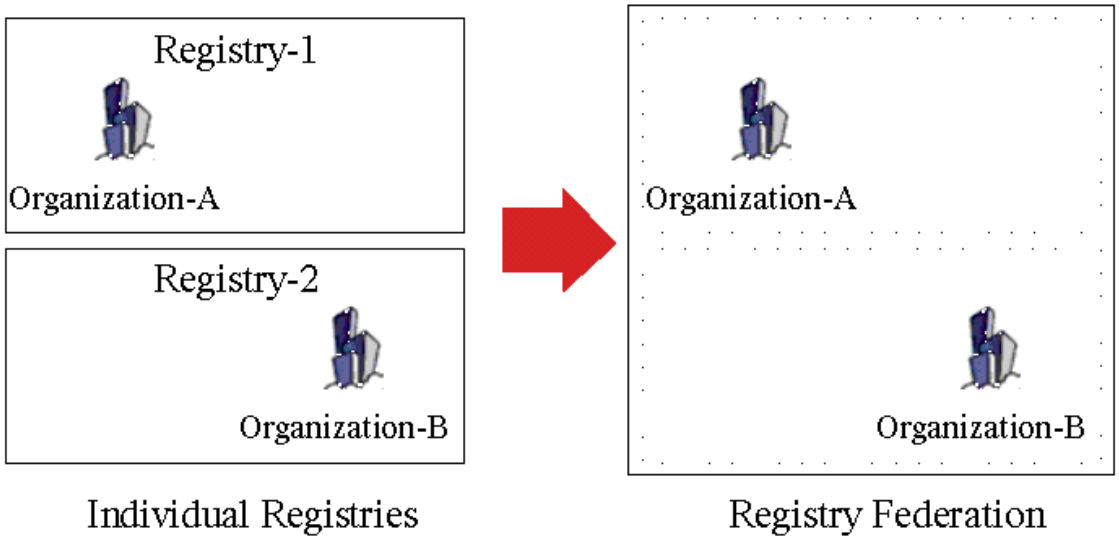


Illustration 11: Registry Federations

8.3.1 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 described in [ebRIM].

8.3.1.1 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

8.3.1.2 Joining a Federation

The following rules govern how a server joins a federation:

- Each server SHOULD have exactly one local RegistryType instance. Each server MAY have multiple remote RegistryType instances
- A server MAY join an existing federation by submitting an instance of an Association that associates the Federation instance as sourceObject, to the Registry instance representing the server as targetObject, using a type of *HasFederationMember*. The home server for the Association and the Federation objects MUST be the same
- A Federation (child federation) MAY join an existing federation (parent federation) by submitting an instance of an Association that associates the Federation instance representing the parent federation as sourceObject, to the Federation instance representing the child federation as

targetObject, using a type of *HasFederationMember*. The home server for the Association and the parent Federation objects MUST be the same

8.3.1.3 Leaving a Federation

The following rules govern how a server leaves a federation:

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

8.3.1.4 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 FederationType instance
- The removal of a FederationType instance is governed by Access Control Policies like any other RegistryObject

8.3.2 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.

8.3.2.1 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.

8.3.2.2 Federated Queries

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

A server MUST route a federated query received by it to all servers that are represented by RegistryType instances in the membership tree of the federation(s) that is the target of the federated query on a best attempt basis.

If an exception is encountered while dispatching a query to a federation member 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 type 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 server 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 is distributed across all the 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.

8.3.3 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 locally caching the Federation configuration.

The federation member MUST keep the cached federation configuration synchronized with the original object in the Federation home.

8.3.4 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.

9 Governance Features

This chapter specifies how a server supports governance of RegistryObjects.

Governance is defined as the enforcement of business processes and policies defined by a Community of Practice, that guide, direct, and control how its members collaborate to achieve its business goals.

Within this specification, governance is defined as the enforcement of collaborative business processes and policies defined by a Community of Practice to manage the end-to-end life cycle of RegistryObjects within the server. Such collaborative business processes will be referred to as “governance collaborations”.

The remainder of this chapter specifies:

- Scope of governance collaborations
- How governance collaborations are represented,
- How representations of governance collaborations are assigned to RegistryObjects, and
- How a server uses the representation of governance collaborations assigned to a RegistryObjects to govern them

9.1 Representing a Governance Collaboration

This specification makes use of BPMN 2.0[1] [BPMN2] to represent business collaborations that govern RegistryObjects as follows:

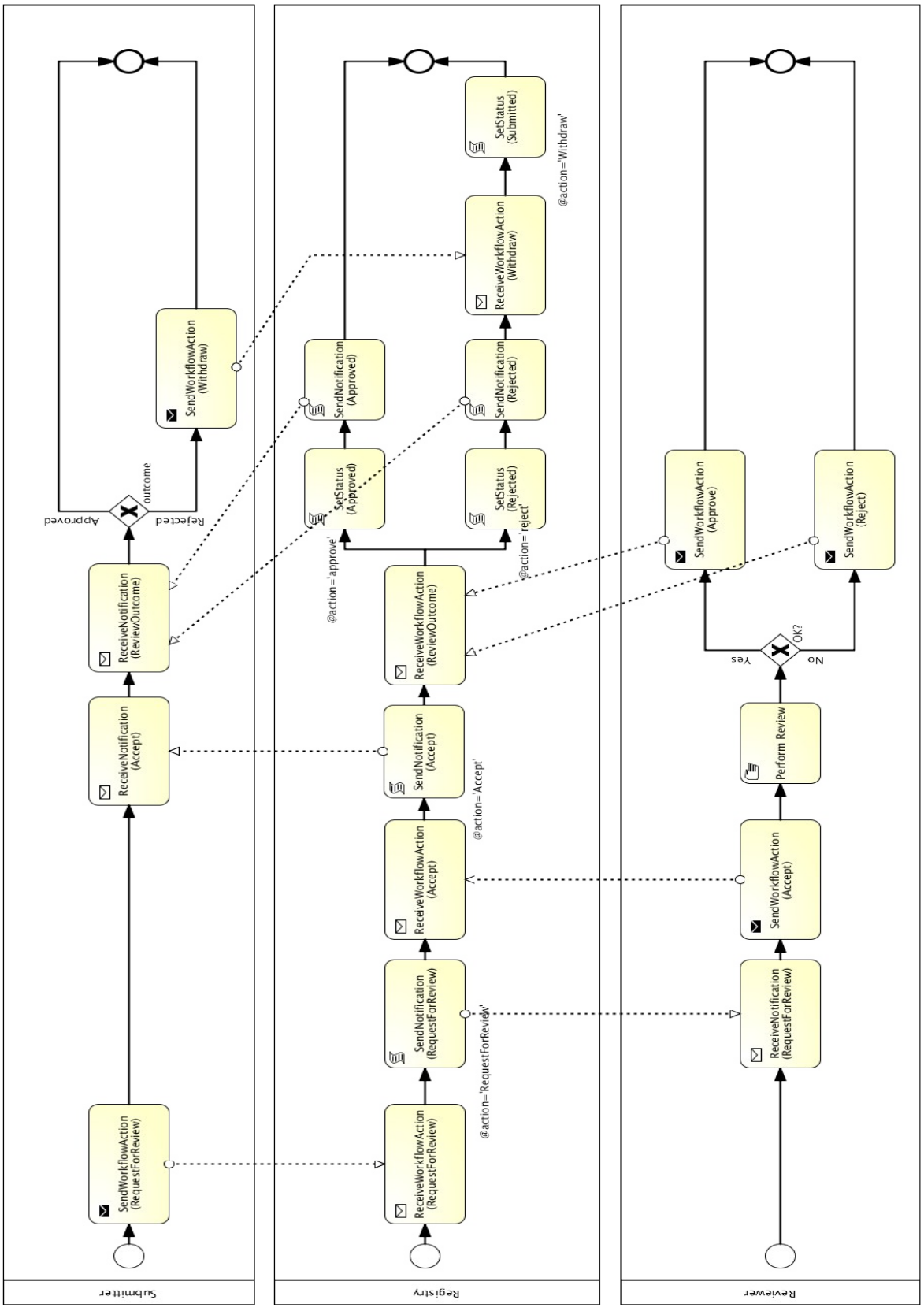
- Uses BPMN 2.0 diagram notation to pictorially represent business collaborations
- Uses BPMN 2.0 XML format to declaratively represent business collaborations in a machine processable syntax

A governance collaboration consists of one or more participants where each participant's activities within the collaboration is described by a separate BPMN process and the interaction between the participants' processes is described by a single BPMN collaboration.

Detailed specification of how to describe governance collaborations in BPMN 2.0 XML format and how a server executes them in a BPMN process engine are provided later in this chapter.

Illustration 12 below provides an example of the Default Governance Collaboration represented by a BPMN 2.0 diagram notation. The Default Governance Collaboration is provided as a standard governance collaboration readily available for use in any server. It is described in detail later in this chapter.

[1] At the time of this writing BPMN 2.0 is not final yet. This specification uses the BPMN 2.0 Beta 2 specification as a reference at this time since BPMN 2.0 is not final yet.



9.1.1 Content of Governance Collaboration BPMN Files

The collective content of the Governance Collaboration BPMN files, whether organized as a set of related modular files or a single monolithic file, MUST meet the following requirements:

- There MUST be exactly one collaboration element
- The collaboration element MUST have at least one participant element
- At least once participant element MUST have id value of “registryPartcipant” and represents the RegRep server as a participant within the governance collaboration
- There MUST be a processRef element for the “registryPartcipant”
- There MUST be a process element for each processRef attribute in each participant element
- The process element for other participants than the “registryPartcipant” participant MAY conform to “Descriptive Conformance Sub-Class ”[1] or “Analytic Conformance Sub-Class ”[2] in [BPMN2] and need not be executed within a BPMN process engine
- The process element for the “registryPartcipant” participant's process MUST conform to “Common Executable Conformance Sub-Class”[3] in [BPMN2] and MUST be executed by the server in a BPMN process engine
- The process elements SHOULD use tasks that conform to canonical task patterns defined later in this specification whenever possible

9.2 Scope of Governance Collaborations

A governance collaboration may govern a single RegistryObject or it may govern a set of related RegistryObjects packaged together within a RegistryPackage as a single unit of governance. In either case, the target object of the governance collaboration is referred to as the governed object.

9.2.1 Packaging Related Objects as a Governance Unit

A client MUST publish a set of related RegistryObjects that are to be governed by the server as a single unit as follows:

- The objects MUST be immediate members of the same RegistryPackage
- The RegistryPackage MUST have a canonical slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryPackage:packageType”
- The value of the packageType slot MUST be a unique identifier for the type of package of which the group of related objects are an instance

A server MUST treat RegistryPackages with a canonical slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryPackage:packageType” as the governed object.

---

[1] This is also referred to as a “Layer 1”, representation layer or presentation layer

[2] This is also referred to as a “Layer 2” or analytical layer

[3] This is also referred to as a “Layer 3” or executable layer

9.3 Assigning a Governance Collaboration

A governance collaboration as represented by a BPMN2 XML file is not directly assigned to a RegistryObject. Instead it is assigned to a RegistryPackage and is implicitly applicable to RegistryObjects that are members of the RegistryPackage.

Governance collaboration MAY be assigned to a specific RegistryPackage using a “GovernedBy” Association as follows:

- The type attribute value of Association MUST reference the canonical “GovernedBy” ClassificationNode within the canonical AssociationType ClassificationScheme whose id is “urn:oasis:names:tc:ebxml-regrep:AssociationType:GovernedBy”
- The targetObject attribute value of Association MUST reference an ExtrinsicObject with objectType “urn:oasis:names:tc:ebxml-regrep:ObjectType:RegistryObject:ExtrinsicObject:XML:BPMN2”
- The repository item for the ExtrinsicObject MUST be an XML document conforming to the BPMN2 model XML Schema. If the modular approach to BPMN description is used then this file MUST be the collaboration BPMN file. The file MUST import or contain the BPMN process for the “Registry” participant
- The sourceObject attribute value of Association MUST reference the RegistryPackage instance to which the governance collaboration is being assigned
- The RegistryPackage MUST NOT have a canonical slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryPackage:packageType”

9.4 Determining Applicable Governance Collaboration

For any given RegistryObject, a server MUST use the following algorithm to determine the applicable governance collaboration (if any):

1. Check if objects is an immediate member of a RegistryPackage that has a canonical slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryPackage:packageType”.
    a) If it is so, then the object is not governed directly and instead its parent RegistryObjects is the governed object
    b) Otherwise, proceed to next step
2. Check if there is a governance collaboration assigned to a RegistryPackage ancestor using the canonical “HasGovernance” Association as follows:
    a) Do a breadth-first traversal of the tree consisting of all RegistryPackage ancestors of the object and for each RegistryPackage see if it has a governance collaboration assigned to it
    b) Stop when you find the first such governance collaboration
    c) If a governance collaboration is found then use it as applicable governance collaboration
3. If no RegistryPackage-specific governance collaboration is found then the object is not governed by any governance collaboration

9.5 Determining the Registry Process in a Governance Collaboration

For any given governance collaboration, a server MUST use the following algorithm to determine the special Registry process:

1. Find the participant element within the collaboration whose id is the canonical "registryParticipant"
2. Find the processRef attribute of the "registryParticipant" and use the referenced process as the Registry process

9.6 Starting the Registry Process for a Governance Collaboration

The BPMN process for the "registryParticipant" within a governance collaboration is the only process in the collaboration that is required to be executed by the server within a BPMN process engine. This section specifies when and how a server starts this process.

9.6.1 Starting Registry Process By WorkflowAction

A server MAY start the Registry process for a governance collaboration in response to the publishing of a WorkflowAction object. This is specified in detail in 10.8.1.1 Server Processing of WorkflowAction.

9.7 Incoming messageFlows to Registry Process

Within a governance collaboration, a server MUST support incoming messageFlows to the Registry process from other processes in the collaboration that meet the following requirements:

- The sourceRef attribute of the messageFlow references a task that conforms to the SendWorkflowAction task template described later in this chapter
- The targetRef attribute of the messageFlow references a task that conforms to the ReceiveWorkflowAction task template described later in this chapter
- The messageRef attribute of the messageFlow is defined and references a message whose itemDefinition has attribute structureRef="rim:WorkflowActionType"

A server MAY support other types of incoming messages.

9.8 Outgoing messageFlows from Registry Process

A Registry process communicates with non-Registry processes by sending them notification messages. These messages may be an email message to an email endpoint for a person or a rim:NotificationType message to a service endpoint. Details are provided in the specification for the SendNotification task pattern.

A server MAY support other types of outgoing messages.

9.9 Canonical Task Patterns

This section specifies a set of canonical task patterns that may be used within participant processes in a governance collaboration. Some of these task patterns can only be used within the Registry process while some may only be used in the non-Registry processes of a governance collaboration.

2135 The following table provides a brief summary each of the canonical tasks defined by this specification.
2136 Subsequent sections specify these tasks in more detail.

2137

Task Pattern	Task Type	Used In	Description
SendWorkflow Action	sendTask	Non-Registry Process	Sends a WorkflowAction message to the Registry process
ReceiveWorkflow Action	receiveTask	Registry Process	Waits until a WorkflowAction message is received from a non-Registry process
SendNotification	scriptTask	Registry Process	Sends a Notification message to a non-Registry process
ReceiveNotification	receiveTask	Non-Registry Process	Receives a Notification message from the Registry process
SetStatus	scriptTask	Registry Process	Sets the status of the specified RegistryObject
Validate	serviceTask	Any Process	Validates a RegistryObject
Catalog	serviceTask	Any Process	Catalogs a RegistryObject

2138

2139 9.9.1 SendWorkflowAction Task Pattern

2140 This canonical task pattern is used by a sendTask to represent the performing of a process-specific
2141 action upon the governed object. This task pattern is the primary means for a non-Registry process to
2142 send a message to the Registry process to trigger the Registry process forward.

2143 **Task Inputs**: The task has the following inputs as defined by dataInput elements in its ioSpecification:

2144 - A dataInput that has an itemSubjectRef attribute that references an itemDefinition element
2145 whose structureRef attribute value is “rim:WorkflowActionType”

2146 **Task Outputs**:The task has no outputs.

2147 **Task Actors**: This task SHOULD be performed by a role other than Registry role to indicate that some
2148 external action (e.g. “approval”) has been performed on the targetObject specified by the
2149 WorkflowAction.

2150 **Description**: To perform this task the actor submits a WorkflowAction to the server using the standard
2151 SubmitObjects protocol. The name of the task SHOULD reflect the action being performed by the task
2152 (e.g. name='SendWorkflowAction(RequestForReview)'. The WorkflowAction MUST specify:

2153 - An action attribute identifying the action performed

2154 - A targetObject attribute identifying the object that is the target of the action. Typically, this is the
2155 governed object

2156 9.9.1.1 Server Processing of WorkflowAction

2157 Upon publishing of a WorkflowAction a server MUST process it as shown in the following pseudo-code
2158 and explained further below:

```
WorkflowActionType workflowAction = ...;
Collaboration collaboration =
  getApplicableGovernanceCollaboration(workflowAction.getTargetObject());

if (collaboration != null) {
  Process registryProcess = collaboration.getRegistryProcess();
  if (registryProcess != null) {
    if (!registryProcess.isActive()) {
      registryProcess.start();
    }
    registryProcess.deliverMessage(workflowAction);
  }
}
```

1. Determine and get the applicable Governance Collaboration (as defined in 10.3 Determining Applicable Governance Collaboration)
2. Determine and get the applicable Registry process for the collaboration (as defined in 10.4 Determining the Registry Process in a Governance Collaboration)
3. If the Registry process has not yet been started then start it within the BPMN process engine
4. Deliver the WorkflowAction message to the Registry process where presumably a receiveTask based on the ReceiveWorkflowAction task pattern is waiting for it

9.9.2 ReceiveWorkflowAction Task Pattern

This canonical task pattern is used by a receiveTask that waits for a process-specific action to be performed upon the governed object. This task pattern is the primary means for the Registry process to receive a message from a non-Registry process to trigger the Registry process forward.

Task Inputs: The task has the following inputs as defined by dataInput elements in its ioSpecification:

- A dataInput that has an itemSubjectRef attribute that references an itemDefinition element whose structureRef attribute value is “rim:WorkflowActionType”

Task Outputs:The task has no outputs.

Task Actors: This task MUST be performed by the Registry role to wait until some external action (e.g. “approval”) has been performed on the targetObject specified by the WorkflowAction.

Description: This task waits until the server delivers a WorkflowAction message to the Registry process. The name of the task SHOULD reflect the action being performed (e.g. name='ReceiveWorkflowAction(RequestForReview)'. The task is typically followed by sequenceFlow elements that have a conditionExpression that predicate on the value of the action attribute of the WorkflowAction.

9.9.3 SendNotification Task Pattern

This canonical task pattern is used by a scriptTask to send a Notification message regarding the governed object to the roles and email addresses specified for the task. This task pattern is the primary means for the Registry process to send a message to a non-Registry process to trigger the non-Registry process forward.

Task Inputs: None

Task Outputs: None

Task Actors: This task MUST be performed by the Registry role to keep governance roles for the governed object informed of important changes (e.g. status attribute changes) during the course of the life cycle of the governed object.

Description: To perform this task the actor uses the sendNotification canonical XPATH extension function defined later in this chapter. The name of the task SHOULD reflect the nature of the notification being sent by the task (e.g. name='SendNotification(Accept)'.

9.9.4 ReceiveNotification Task Pattern

This canonical task pattern is used by a receiveTask that waits for a Notification message to be delivered. This task pattern is the primary means for a non-Registry process to receive a message from the Registry process to trigger the non-Registry process forward.

Task Inputs: The task has the following inputs as defined by dataInput elements in its ioSpecification:

- A dataInput that has an itemSubjectRef attribute that references an itemDefinition element whose structureRef attribute value is “rim:NotificationType”

Task Outputs:The task has no outputs.

Task Actors: This task MUST be performed by a non-Registry role

Description: This task waits until the server delivers a Notification message. The name of the task SHOULD reflect the nature of the notification being received by the task (e.g. name='ReceiveNotification(Accept)'.

9.9.5 SetStatus Task

This canonical task pattern is used by a scripTask that updates the status of the specified object to a specified status value.

Task Inputs: None

Task Outputs: None

Task Actors: This task MUST be performed by the Registry role to reflect changes in life cycle status during the course of the life cycle of the governed object.

Description: To perform this task the actor uses the setStatus canonical XPATH extension function defined later in this chapter. The name of the task SHOULD reflect the status being set by the task (e.g. name='SendStatus(Approved)'.

9.9.6 Validate Task

This canonical task represents the validation of the governed object.

Task Inputs: The task has no explicit inputs.

Task Outputs:The task has no outputs.

Task Actors: This task SHOULD be performed by the Registry role in response to the creation or updating of the governed object.

Description: To perform this task the actor validates the governed object using the standard ValidateObjects protocol. The name of the task SHOULD be 'Validate' or an equivalent native language translation.

9.9.7 Catalog Task

This canonical task represents the cataloging of the governed object.

Task Inputs: The task has no explicit inputs.

Task Outputs: The task has no outputs.

Task Actors: This task SHOULD be performed by the Registry role in response to the creation or updating of the governed object.

Description: To perform this task the actor catalogs the governed object using the standard CatalogObjects protocol. The name of the task SHOULD be 'Catalog' or an equivalent native language translation.

9.10 XPATH Extension Functions

The following table specifies XPATH extension functions that MUST be supported by the BPMN process engine used by the server. The function signatures are described using the same conventions as used in section 1.4 of [XPATHFUNC].

These functions MAY be used within XPATH expressions in a BPMN file wherever a **tExpression** type is supported by the BPMN schema.

- The namespace URI for these functions MUST be "urn:oasis:names:tc:ebxml-regrep:xsd:rs:4.0"
- The namespace prefix SHOULD be “rs”

XPATH Extension Function	Description
rs:generateId() as xs:string	Returns a newly generated unique id for a RegistryObject. This SHOULD be a URN in the urn:uuid namespace
rs:getRegistryObject(id as xs:string) as element()	Returns the RegistryObject element for the RegistryObject that matches the specified id after retrieving it from the server. This is typically used to get the governed object.
rs:setStatus(targetObject as xs:string, status as xs:string) as none	Sets the status of the object matching targetObject with the specified status. Used by the SetStatus task pattern. This function returns no value.
rs:sendNotification(toRoles as xs:string*, toEmails as xs:string*, subject as xs:string?, message as xs:string) as none	Send a notification message using an optional subject to specified roles and email addresses. If toRoles is specified then the server MUST be able to resolve each role to a target person or service instances and determine a delivery endpoint for the target. The message SHOULD be specified as a CDATA if it contains any special characters used by XML. This function returns no value. Used by the SendNotification task pattern.

In addition to the functions described in table above, all canonical query functions supported by the server MUST also be supported by the server as XPATH functions.

9.11 Default Governance Collaboration

This section defines a canonical governance collaboration called the “Default Governance Collaboration”. The Default Governance Collaboration is defined by this specification to provide a standard governance process that can be supported by all implementations and may be assigned to specific RegistryPackages.

The Default Governance Collaboration is represented by a canonical ExtrinsicObjectType instance with id “urn:oasis:names:tc:ebxml-regrep:collaboration:DefaultGovernanceCollaboration”.

A BPMN diagram for the Default Governance Collaboration has been provided in Illustration 12 earlier.

The Default Governance Collaboration is summarized as follows:

- The submitter requests review and approval of the governed object using SendWorkflowAction canonical task pattern with action “RequestForReview”
- The server receives the “RequestForReview” WorkflowAction and notifies the reviewer roles of the request for review using Notify canonical task pattern
- A reviewer accepts the request for review using SendWorkflowAction canonical task with WorkflowAction “Accept”
- The server notifies submitter roles that the governed object is under review using the using Notify canonical task
- The reviewer approves or rejects the governed objects using SendWorkflowAction canonical task and actions “Approve” or “Reject”
- The server notifies the submitter of the outcome of the review using the using Notify canonical task

10 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

10.1 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).

10.1.1 Transport Layer Security

A server SHOULD 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.

10.1.2 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]

10.2 Message Confidentiality

A server SHOULD support encryption of protocol messages as defined by 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.

10.3 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.

10.4 Authentication

A server MUST support authentication of the client requests based on the security tokens provided by the client and supported by the server. 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.

10.5 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.

10.6 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 for each request that affected the state of server resources. A server MUST create this audit trail using AuditableEventType 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].

11 Native Language Support (NLS)

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

11.1 Terminology

The following terms are used in NLS.

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

11.2 NLS and Registry 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.

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].

11.3 NLS Support in RegistryObjects

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

```
<complexType name="LocalizedStringType">
  <attribute ref="xml:lang" use="optional" default="en-US"/>
  <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 sensitive <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:RegistryObject xsi:type="rim:ExtrinsicObjectType"...>
  <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:RegistryObjectType>
```

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

11.3.1 Language of a LocalizedString

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

11.3.2 Character Set of RegistryObject

The character set used by a 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.

11.4 NLS and Repository Items

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

11.4.1 Character Set of Repository Items

When a submitter submits a repository item, they MAY specify the character set used by the repository 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"
```

A server MUST preserve the charset of a repository item as it is originally specified when it is submitted to the server.

11.4.2 Language of Repository Items

This specification currently does not provide for a mechanism to specify the language of a RepositoryItem.

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.

12 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.

12.1 Canonical URL

The canonical URL is an HTTP GET URL that MAY be used to reference or access RegistryObjectType instance in a RESTful manner. 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 the chapter titled Access Control Information Model.

12.1.1 Canonical URL for RegistryObjects

The canonical URL for RegistryObjectType has the following pattern:

```
//The {id} parameter specifies the id of a RegistryObject
GET /rest/registryObjects/{id}
```

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 id: urn:acme:pictures:danyal.jpg
GET
http://acme.com/myregistry/rest/registryObjects/urn:acme:pictures:danyal.jpg


//Get RegistryObject id: http://www.acme.com/pictures/danyal.jpg
GET
http://acme.com/myregistry/rest/registryObjects/http://www.acme.com/pictures
/danyal.jpg
```

12.1.2 Canonical URL for Repository Items

The canonical URL for repository items has the following pattern:

```
//The {id} parameter specifies the id of a RegistryObject for repository
item
GET /rest/repositoryItems/{id}
```

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 repository item associated with
```

```
//ExtrinsicObject with id: urn:acme:pictures:danyal.jpg
GET
http://acme.com/myregistry/rest/repositoryItems/urn:acme:pictures:danyal.jpg


//Get repository item associated with
//ExtrinsicObject with id: http://www.acme.com/pictures/danyal.jpg
GET
http://acme.com/myregistry/rest/repositoryItems/http://www.acme.com/pictures
/danyal.jpg
```

12.2 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>/rest/search?queryId={the query id}(&{<param-name>=<param-
value>})*
```

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/rest/search?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectById&id=urn:acme:pictures:danyal.jpg
```

12.2.1 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.

12.2.2 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 query-specific parameter defined by the FindObjectsByIdAndType query.

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

12.2.3 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&depth=10
```

12.2.4 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-
us&format=application/x-ebrs+xml
```

12.2.5 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true
```

12.2.6 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true&f
ederation=urn:acme:federation:acme-partners
```

12.2.7 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:BasicQuery&name=TestRegister1&matchOlderVersionsOnQuery=true
```

12.2.8 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10&st
artIndex=30
```

12.2.9 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-us
```

12.2.10 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
/rest/search/?queryId=urn:oasis:names:tc:ebxml-
regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10
```

12.2.11 Use of Functions in Query Parameters

Query functions may be used in query parameters as defined in Query Function. The only caveat is that the special characters such as the special sequences "#@" and "@#", special characters "(", ")" etc. MUST be specified in their URL encoded representation as defined by RFC 3986 and RFC 3629.

For example a query parameter "#@'@#rs:currentTime#@'@#" would evaluate to the current time as a quoted timestamp string in ISO 8601 format such as "#@'@#2010-08-05T17:14:18.866#@'@#". Such a query parameter in REST interface would have to be URL encoded to be as shown in the following example:

```
http://localhost:8080/omar-server/rest/search?
queryId=urn:ogc:specification:regrep:profile:ISO19139:query:DatasetDiscovery
Query&title=%23%40%%2740%23ebrs:currentTime%28%29%23%40%%2740%23
```

12.2.12 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/x-ebrs+xml” then the response document must have query:QueryResponse element as its root element.

13 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 available 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.

13.1 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:onNotific
ation</wsa:Action>
  </env:Header>
  <env:Body>
     <rim:Notification .../>
  </env:Body>
</env:Envelope>
```

Appendix A. 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.