



OASIS ebXML RegRep Version 4.0

Part 2: Services and Protocols (ebRS)

Committee Specification Draft 02 /
Public Review Draft 01

12 May 2011

Specification URIs:

This version:

<http://docs.oasis-open.org/regrep/regrep-core/v4.0/csprd01/regrep-core-rs-v4.0-csprd01.odt>
(Authoritative)
<http://docs.oasis-open.org/regrep/regrep-core/v4.0/csprd01/regrep-core-rs-v4.0-csprd01.pdf>
<http://docs.oasis-open.org/regrep/regrep-core/v4.0/csprd01/regrep-core-rs-v4.0-csprd01.html>

Previous 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>

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

Chairs:

Kathryn Breininger, Boeing
Farrukh Najmi, Wellfleet Software

Editors:

Farrukh Najmi, Wellfleet Software
Nikola Stojanovic, Individual

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](#) - 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 \(ebRS\)](#) (this document) - 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 namespaces:

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:

This document was last revised or approved by the [OASIS ebXML Registry TC](#) on the above date. The level of approval is also listed above. Check the "Latest Version" location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the "[Send A Comment](#)" button on the Technical Committee's web page at <http://www.oasis-open.org/committees/regrep/>.

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

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)*.
12 May 2011. OASIS Committee Specification Draft 02 / Public Review Draft 01.
<http://docs.oasis-open.org/regrep/regrep-core/v4.0/csprd01/regrep-core-rs-v4.0-csprd01.odt>.

Notices

Copyright © OASIS Open 2010-2011. All Rights Reserved.

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

| | | |
|---------|--|----|
| 1 | Introduction..... | 11 |
| 1.1 | Terminology..... | 11 |
| 1.2 | Abstract Protocol..... | 11 |
| 1.2.1 | RegistryRequestType..... | 11 |
| 1.2.1.1 | Syntax..... | 11 |
| 1.2.1.2 | Description..... | 11 |
| 1.2.2 | RegistryResponseType..... | 11 |
| 1.2.2.1 | Syntax..... | 12 |
| 1.2.2.2 | Description..... | 12 |
| 1.2.3 | RegistryExceptionType..... | 12 |
| 1.2.3.1 | Syntax..... | 13 |
| 1.2.3.2 | Description..... | 13 |
| 1.3 | Server Plugins..... | 13 |
| 2 | QueryManager Interface..... | 14 |
| 2.1 | Parameterized Queries..... | 14 |
| 2.1.1 | Invoking Adhoc Queries..... | 14 |
| 2.2 | Query Protocol..... | 14 |
| 2.2.1 | QueryRequest..... | 14 |
| 2.2.1.1 | Syntax..... | 15 |
| 2.2.1.2 | Example..... | 15 |
| 2.2.1.3 | Description..... | 15 |
| 2.2.1.4 | Response..... | 16 |
| 2.2.1.5 | Exceptions..... | 16 |
| 2.2.2 | Element Query..... | 16 |
| 2.2.2.1 | Syntax..... | 17 |
| 2.2.2.2 | Description:..... | 17 |
| 2.2.3 | Element ResponseOption..... | 17 |
| 2.2.3.1 | Syntax..... | 17 |
| 2.2.3.2 | Description:..... | 17 |
| 2.2.4 | QueryResponse..... | 18 |
| 2.2.4.1 | Syntax..... | 18 |
| 2.2.4.2 | Example..... | 18 |
| 2.2.4.3 | Description:..... | 19 |
| 2.2.5 | Iterative Queries..... | 19 |
| 2.3 | Parameterized Query Definition..... | 19 |
| 2.4 | Canonical Query: AdhocQuery..... | 19 |
| 2.4.1 | Parameter Summary..... | 20 |
| 2.4.2 | Query Semantics..... | 20 |
| 2.5 | Canonical Query: BasicQuery..... | 20 |
| 2.5.1 | Parameter Summary..... | 20 |
| 2.5.2 | Query Semantics..... | 21 |
| 2.6 | Canonical Query: ClassificationSchemeSelector..... | 21 |
| 2.6.1 | Parameter Summary..... | 21 |
| 2.6.2 | Query Semantics..... | 21 |
| 2.7 | Canonical Query: FindAssociations..... | 22 |
| 2.7.1 | Parameter Summary..... | 22 |
| 2.7.2 | Query Semantics..... | 22 |
| 2.8 | Canonical Query: FindAssociatedObjects..... | 23 |
| 2.8.1 | Parameter Summary..... | 23 |

| | |
|--|----|
| 2.8.2 Query Semantics..... | 24 |
| 2.9 Canonical Query: GarbageCollector..... | 24 |
| 2.9.1 Parameter Summary..... | 24 |
| 2.9.2 Query Semantics..... | 24 |
| 2.10 Canonical Query: GetAuditTrailByLid..... | 24 |
| 2.10.1 Parameter Summary..... | 24 |
| 2.10.2 Query Semantics..... | 25 |
| 2.11 Canonical Query: GetAuditTrailByLid..... | 25 |
| 2.11.1 Parameter Summary..... | 25 |
| 2.11.2 Query Semantics..... | 25 |
| 2.12 Canonical Query: GetAuditTrailByTimeInterval..... | 26 |
| 2.12.1 Parameter Summary..... | 26 |
| 2.12.2 Query Semantics..... | 26 |
| 2.13 Canonical Query: GetChildrenByParentId..... | 26 |
| 2.13.1 Parameter Summary..... | 26 |
| 2.13.2 Query Semantics..... | 27 |
| 2.14 Canonical Query: GetClassificationSchemesByLid..... | 28 |
| 2.14.1 Parameter Summary..... | 28 |
| 2.14.2 Query Semantics..... | 28 |
| 2.15 Canonical Query: GetRegistryPackagesByMemberId..... | 28 |
| 2.15.1 Parameter Summary..... | 28 |
| 2.15.2 Query Semantics..... | 28 |
| 2.16 Canonical Query: GetNotification..... | 29 |
| 2.16.1 Parameter Summary..... | 29 |
| 2.16.2 Query Semantics..... | 29 |
| 2.17 Canonical Query: GetObjectByLid..... | 29 |
| 2.17.1 Parameter Summary..... | 29 |
| 2.17.2 Query Semantics..... | 29 |
| 2.18 Canonical Query: GetObjectsByLid..... | 29 |
| 2.18.1 Parameter Summary..... | 30 |
| 2.18.2 Query Semantics..... | 30 |
| 2.19 Canonical Query: GetReferencedObject..... | 30 |
| 2.19.1 Parameter Summary..... | 30 |
| 2.19.2 Query Semantics..... | 30 |
| 2.20 Canonical Query: KeywordSearch..... | 30 |
| 2.20.1 Canonical Indexes..... | 31 |
| 2.20.2 Parameter Summary..... | 31 |
| 2.20.3 Query Semantics..... | 32 |
| 2.21 Canonical Query: RegistryPackageSelector..... | 32 |
| 2.21.1 Parameter Summary..... | 33 |
| 2.21.2 Query Semantics..... | 33 |
| 2.22 Query Functions..... | 33 |
| 2.22.1 Using Functions in Query Expressions..... | 33 |
| 2.22.2 Using Functions in Query Parameters..... | 34 |
| 2.22.3 Function Processing Model..... | 35 |
| 2.22.4 Function Processor BNF..... | 35 |
| 2.23 Common Patterns In Query Functions..... | 36 |
| 2.23.1 Specifying a null Value for string Param or Return Value..... | 36 |
| 2.24 Canonical Functions..... | 36 |
| 2.24.1 Canonical Function: currentTime..... | 37 |
| 2.24.1.1 Function Semantics..... | 37 |

| | |
|--|----|
| 2.24.2 Canonical Function: currentUserId..... | 37 |
| 2.24.2.1 Function Semantics..... | 37 |
| 2.24.3 Canonical Function: relativeTime..... | 37 |
| 2.24.3.1 Parameter Summary..... | 37 |
| 2.24.3.2 Function Semantics..... | 38 |
| 2.24.4 Canonical Function: getClassificationNodes..... | 38 |
| 2.24.4.1 Parameter Summary..... | 38 |
| 2.24.4.2 Function Semantics..... | 38 |
| 2.25 Query Plugins..... | 39 |
| 2.25.1 Query Plugin Interface..... | 39 |
| 3 LifecycleManager Interface..... | 40 |
| 3.1 SubmitObjects Protocol..... | 40 |
| 3.1.1 SubmitObjectsRequest..... | 40 |
| 3.1.1.1 Syntax..... | 40 |
| 3.1.1.2 Description..... | 41 |
| 3.1.1.3 id and lid Requirements..... | 41 |
| 3.1.1.4 Returns..... | 42 |
| 3.1.1.5 Exceptions..... | 42 |
| 3.1.2 Audit Trail Requirements..... | 43 |
| 3.1.3 Sample SubmitObjectsRequest..... | 43 |
| 3.2 The Update Objects Protocol..... | 43 |
| 3.2.1 UpdateObjectsRequest..... | 44 |
| 3.2.1.1 Syntax..... | 44 |
| 3.2.1.2 Description..... | 44 |
| 3.2.1.3 Returns..... | 45 |
| 3.2.1.4 Exceptions..... | 45 |
| 3.2.2 UpdateAction..... | 45 |
| 3.2.2.1 Syntax..... | 45 |
| 3.2.2.2 Description..... | 46 |
| 3.2.3 Audit Trail Requirements..... | 47 |
| 3.2.4 Sample UpdateObjectsRequest..... | 47 |
| 3.3 RemoveObjects Protocol..... | 47 |
| 3.3.1 RemoveObjectsRequest..... | 48 |
| 3.3.1.1 Syntax..... | 48 |
| 3.3.1.2 Description..... | 48 |
| 3.3.1.3 Returns:..... | 49 |
| 3.3.1.4 Exceptions:..... | 49 |
| 3.3.2 Audit Trail Requirements..... | 49 |
| 3.3.3 Sample RemoveObjectsRequest..... | 49 |
| 4 Version Control..... | 51 |
| 4.1 Version Controlled Resources..... | 51 |
| 4.2 Versioning and Id Attribute..... | 52 |
| 4.3 Versioning and Lid Attribute..... | 52 |
| 4.4 Version Identification for RegistryObjectType..... | 52 |
| 4.5 Version Identification for RepositoryItem..... | 52 |
| 4.5.1 Versioning of RegistryObjectType..... | 52 |
| 4.5.2 Versioning of ExtrinsicObjectType..... | 53 |
| 4.6 Versioning and References..... | 53 |
| 4.7 Versioning of RegistryPackages..... | 54 |
| 4.8 Versioning and RegistryPackage Membership..... | 54 |
| 4.9 Inter-version Association..... | 54 |
| 4.10 Version Removal..... | 54 |

| | | |
|---------|---|----|
| 4.11 | Locking and Concurrent Modifications..... | 55 |
| 4.12 | Version Creation..... | 55 |
| 5 | Validator Interface..... | 56 |
| 5.1 | ValidateObjects Protocol..... | 56 |
| 5.1.1 | ValidateObjectsRequest..... | 56 |
| 5.1.1.1 | Syntax..... | 56 |
| 5.1.1.2 | Example..... | 57 |
| 5.1.1.3 | Description..... | 57 |
| 5.1.1.4 | Response..... | 57 |
| 5.1.1.5 | Exceptions..... | 57 |
| 5.1.2 | ValidateObjectsResponse..... | 57 |
| 5.2 | Validator Plugins..... | 57 |
| 5.2.1 | Validator Plugin Interface..... | 58 |
| 5.2.2 | Canonical XML Validator Plugin..... | 58 |
| 6 | Cataloger Interface..... | 59 |
| 6.1 | CatalogObjects Protocol..... | 59 |
| 6.1.1 | CatalogObjectsRequest..... | 59 |
| 6.1.1.1 | Syntax..... | 59 |
| 6.1.1.2 | Example..... | 60 |
| 6.1.1.3 | Description..... | 60 |
| 6.1.1.4 | Response..... | 60 |
| 6.1.1.5 | Exceptions..... | 60 |
| 6.1.2 | CatalogObjectsResponse..... | 60 |
| 6.1.2.1 | Syntax..... | 61 |
| 6.1.2.2 | Example..... | 61 |
| 6.1.2.3 | Description..... | 61 |
| 6.2 | Cataloger Plugins..... | 62 |
| 6.2.1 | Cataloger Plugin Interface..... | 62 |
| 6.2.2 | Canonical XML Cataloger Plugin..... | 62 |
| 7 | Subscription and Notification..... | 64 |
| 7.1 | Server Events..... | 64 |
| 7.1.1 | Pruning of Events..... | 64 |
| 7.2 | Notifications..... | 64 |
| 7.3 | Creating a Subscription..... | 64 |
| 7.3.1 | Subscription Authorization..... | 64 |
| 7.3.2 | Subscription Quotas..... | 64 |
| 7.3.3 | Subscription Expiration..... | 64 |
| 7.3.4 | Event Selection..... | 65 |
| 7.4 | Event Delivery..... | 65 |
| 7.4.1 | Notification Option..... | 66 |
| 7.4.2 | Delivery to NotificationListener Web Service..... | 66 |
| 7.4.3 | Delivery to Email Address..... | 66 |
| 7.4.4 | Delivery to a NotificationListener Plugin..... | 66 |
| 7.4.4.1 | Processing Email Notification Via XSLT..... | 66 |
| 7.5 | NotificationListener Interface..... | 66 |
| 7.6 | Notification Protocol..... | 67 |
| 7.6.1 | Notification..... | 67 |
| 7.7 | Pulling Notification on Demand..... | 67 |
| 7.8 | Deleting a Subscription..... | 67 |
| 8 | Multi-Server Features..... | 68 |
| 8.1 | Remote Objects Reference..... | 68 |

| | | |
|---------|---|----|
| 8.2 | Local Replication of Remote Objects..... | 68 |
| 8.2.1 | Creating Local Replica and Keeping it Synchronized..... | 69 |
| 8.2.2 | Removing a Local Replica..... | 70 |
| 8.2.3 | Removing Subscription With Remote Server..... | 70 |
| 8.3 | Registry Federations..... | 70 |
| 8.3.1 | Federation Configuration..... | 71 |
| 8.3.1.1 | Creating a Federation..... | 71 |
| 8.3.1.2 | Joining a Federation..... | 71 |
| 8.3.1.3 | Leaving a Federation..... | 72 |
| 8.3.1.4 | Dissolving a Federation..... | 72 |
| 8.3.2 | Local Vs. Federated Queries..... | 72 |
| 8.3.2.1 | Local Queries..... | 72 |
| 8.3.2.2 | Federated Queries..... | 72 |
| 8.3.3 | Local Replication of Federation Configuration..... | 73 |
| 8.3.4 | Time Synchronization Between Federation Members..... | 73 |
| 9 | Governance Features..... | 74 |
| 9.1 | Representing a Governance Collaboration | 74 |
| 9.1.1 | Content of Governance Collaboration BPMN Files..... | 76 |
| 9.2 | Scope of Governance Collaborations..... | 76 |
| 9.2.1 | Packaging Related Objects as a Governance Unit..... | 76 |
| 9.3 | Assigning a Governance Collaboration..... | 77 |
| 9.4 | Determining Applicable Governance Collaboration..... | 77 |
| 9.5 | Determining the Registry Process in a Governance Collaboration..... | 78 |
| 9.6 | Starting the Registry Process for a Governance Collaboration..... | 78 |
| 9.6.1 | Starting Registry Process By WorkflowAction..... | 78 |
| 9.7 | Incoming messageFlows to Registry Process..... | 78 |
| 9.8 | Outgoing messageFlows from Registry Process..... | 78 |
| 9.9 | Canonical Task Patterns..... | 78 |
| 9.9.1 | SendWorkflowAction Task Pattern..... | 79 |
| 9.9.1.1 | Server Processing of WorkflowAction..... | 79 |
| 9.9.2 | ReceiveWorkflowAction Task Pattern..... | 80 |
| 9.9.3 | SendNotification Task Pattern..... | 80 |
| 9.9.4 | ReceiveNotification Task Pattern..... | 81 |
| 9.9.5 | SetStatus Task..... | 81 |
| 9.9.6 | Validate Task..... | 81 |
| 9.9.7 | Catalog Task..... | 82 |
| 9.10 | XPATH Extension Functions..... | 82 |
| 9.11 | Default Governance Collaboration..... | 83 |
| 10 | Security Features..... | 84 |
| 10.1 | Message Integrity..... | 84 |
| 10.1.1 | Transport Layer Security..... | 84 |
| 10.1.2 | SOAP Message Security..... | 84 |
| 10.2 | Message Confidentiality..... | 85 |
| 10.3 | User Registration and Identity Management..... | 85 |
| 10.4 | Authentication..... | 85 |
| 10.5 | Authorization and Access Control..... | 85 |
| 10.6 | Audit Trail..... | 85 |
| 11 | Native Language Support (NLS)..... | 86 |
| 11.1 | Terminology..... | 86 |

| | |
|---|----|
| 11.2 NLS and Registry Protocol Messages..... | 86 |
| 11.3 NLS Support in RegistryObjects | 86 |
| 11.3.1 Language of a LocalizedString | 87 |
| 11.3.2 Character Set of RegistryObject | 87 |
| 11.4 NLS and Repository Items | 88 |
| 11.4.1 Character Set of Repository Items..... | 88 |
| 11.4.2 Language of Repository Items..... | 88 |
| 12 REST Binding..... | 89 |
| 12.1 Canonical URL..... | 89 |
| 12.1.1 Canonical URL for RegistryObjects..... | 89 |
| 12.1.2 Canonical URL for Repository Items..... | 89 |
| 12.2 Query Protocol REST Binding..... | 90 |
| 12.2.1 Parameter queryId..... | 90 |
| 12.2.2 Query Specific Parameters..... | 90 |
| 12.2.3 Canonical Query Parameter: depth..... | 90 |
| 12.2.4 Canonical Query Parameter: format..... | 91 |
| 12.2.5 Canonical Query Parameter: federated..... | 91 |
| 12.2.6 Canonical Query Parameter: federation..... | 91 |
| 12.2.7 Canonical Query Parameter: matchOlderVersions..... | 91 |
| 12.2.8 Canonical Query Parameter: startIndex..... | 92 |
| 12.2.9 Canonical Query Parameter: lang..... | 92 |
| 12.2.10 Canonical Query Parameter: maxResults..... | 92 |
| 12.2.11 Use of Functions in Query Parameters..... | 92 |
| 12.2.12 Query Response..... | 92 |
| 13 SOAP Binding..... | 94 |
| 13.1 WS-Addressing SOAP Headers..... | 94 |
| Appendix A. Protocol Exceptions..... | 95 |

Illustration Index

| | |
|---|----|
| Illustration 1: Query Protocol..... | 14 |
| Illustration 2: SubmitObjects Protocol..... | 40 |
| Illustration 3: UpdateObjects Protocol..... | 44 |
| Illustration 4: RemoveObjects Protocol..... | 48 |
| Illustration 5: A visual example of a version tree..... | 51 |
| Illustration 6: ValidateObjects Protocol..... | 56 |
| Illustration 7: CatalogObjects Protocol..... | 59 |
| Illustration 8: Notification Protocol..... | 67 |
| Illustration 9: Remote Object Reference..... | 68 |
| Illustration 10: Local Replication of Remote Objects..... | 69 |
| Illustration 11: Registry Federations..... | 71 |
| Illustration 12: Default Governance Collaboration..... | 75 |

Index of Tables

| | |
|---|----|
| Table 1: Canonical Functions Defined By This Profile..... | 37 |
| Table 2: Requirements for id and lid During SubmitObjects Protocol..... | 42 |

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 [regrep-overview-v4.0] .

1.1 Terminology

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

1.2 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.

1.2.1 RegistryRequestType

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

1.2.1.1 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>
```

1.2.1.2 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 [regrep-rim-v4.0].
- 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".

1.2.2 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.

38 1.2.2.1 Syntax

```
39 <complexType name="RegistryResponseType">
40   <complexContent>
41     <extension base="rim:ExtensibleObjectType">
42       <sequence>
43         <element name="Exception" type="tns:RegistryExceptionType"
44           minOccurs="0" maxOccurs="unbounded"/>
45         <element ref="rim:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
46         <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1"/>
47       </sequence>
48       <attribute name="status" type="rim:objectReferenceType" use="required"/>
49       <attribute name="requestId" type="anyURI" use="optional"/>
50     </extension>
51   </complexContent>
52 </complexType>
53 <element name="RegistryResponse" type="tns:RegistryResponseType"/>
```

54 1.2.2.2 Description

- 55 ● Element ObjectRefList – Contains a sequence of zero or more RegistryObject elements. It is used
56 by requests that return
- 57 ● Element RegistryObjectList – Contains a sequence of zero or more ObjectRef elements. It is used
58 by requests that return a list of references to RegistryObject instances
- 59 ● Attribute requestId – This attribute contains the id of the request that returned this
60 QueryResponse.
- 61 ● Attribute status – This attribute contains the status of the response. Its value MUST be a
62 reference to a ClassificationNode within the canonical ResponseStatusType
63 ClassificationScheme. A server MUST support the status types as defined by the canonical
64 ResponseStatusType ClassificationScheme. The canonical ResponseStatusType
65 ClassificationScheme may be extended by adding additional ClassificationNodes to it.

66 The following canonical values are defined for the ResponseStatusType ClassificationScheme:

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

77 1.2.3 RegistryExceptionType

78 The RegistryExceptionType is the abstract base type for all exception or fault messages sent by the
79 server to the client in response to a client request. Error: Reference source not foundError: Reference
80 source not foundError: Reference source not foundA list of all protocol exceptions is available in the
81 [Protocol Exceptions appendix](#).

82 1.2.3.1 Syntax

```
83 <complexType name="RegistryExceptionType">
84   <annotation>
85     <documentation>Base for all registry exceptions. Based upon SOAPFault:
86     http://www.w3schools.com/soap/soap_fault.asp</documentation>
87   </annotation>
88   <complexContent>
89     <extension base="rim:ExtensibleObjectType">
90       <attribute name="code" type="string" use="optional"/>
91       <attribute name="detail" type="string" use="optional"/>
92       <attribute name="message" type="string"/>
93       <attribute name="severity" type="rim:objectReferenceType"
94       default="urn:oasis:names:tc:ebxml-regrep:ErrorSeverityType:Error"/>
95     </extension>
96   </complexContent>
97 </complexType>
```

98 1.2.3.2 Description

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

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

110 1.3 Server Plugins

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

115 Subsequent chapters will specify various Service Provider Interfaces (SPI) that defines the standard
116 interface for various types of server plugins. These interfaces are described in form of [WSDL2, WSDL1]
117 specification.

118 A server may implement these interfaces as external web services invoked by the server using [SOAP-
119 MF, SOAP-ADJ] or as plugin modules that share the same process as the server and are invoked by local
120 function calls.

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

123 This specification does not define how a plugin is implemented or how it is configured within a server. Nor
124 does it define whether or how, plugin configuration functionality is made discoverable to clients.

2 QueryManager Interface

125

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

2.1 Parameterized Queries

127

128 A server may support any number of pre-configured queries known as *Parameterized Queries*, that may
129 be invoked by clients. Parameterized queries are similar in concept to stored procedures in SQL.

130 This specification defines a number of [canonical queries](#) that are standard queries that **MUST** be
131 supported by a server. Profiles, implementations and deployments may define additional parameterized
132 queries beyond the canonical queries defined by this specification.

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

135 A parameterized query **MAY** be stored in the server as a specialized RegistryObject called QueryDefinition
136 object which is defined by [regrep-rim-v4.0]. The definition of a QueryDefinition may contain any number
137 of Parameters supported by the query.

2.1.1 Invoking Adhoc Queries

138

139 A client may invoke a client-specific ad hoc query using a special canonical parameterized query called
140 the [AdhocQuery query](#) defined by this specification. Due to the risks associated with un-controlled ad hoc
141 queries, a deployment **MAY** choose to restrict the invocation of the AdhocQuery query to specific roles.
142 This specification does not define a standard query expression syntax for ad hoc queries. A server **MAY**
143 support any number of query expression syntaxes for ad hoc queries.

2.2 Query Protocol

144

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

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

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

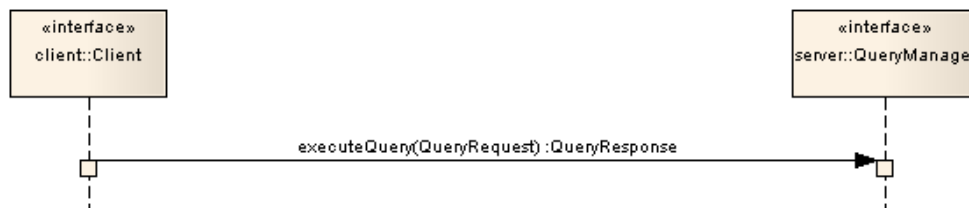


Illustration 1: Query Protocol

2.2.1 QueryRequest

151

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

153 2.2.1.1 Syntax

```
154 <element name="QueryRequest">
155   <complexType>
156     <complexContent>
157       <extension base="rs:RegistryRequestType">
158         <sequence>
159           <element name="ResponseOption" type="tns:ResponseOptionType"
160             minOccurs="1" maxOccurs="1"/>
161           <element name="Query" type="rim:QueryType"
162             minOccurs="1" maxOccurs="1" />
163         </sequence>
164         <attribute name="federated" type="boolean"
165           use="optional" default="false"/>
166         <attribute name="federation" type="anyURI" use="optional"/>
167         <attribute name="format" type="string"
168           use="optional" default="application/ebxml+xml"/>
169         <attribute ref="xml:lang" use="optional"/>
170         <attribute name="startIndex" type="integer" default="0"/>
171         <attribute name="maxResults" type="integer" default="-1"/>
172         <attribute name="depth" type="integer" default="0"/>
173         <attribute name="matchOlderVersions" type="boolean"
174           use="optional" default="false"/>
175       </extension>
176     </complexContent>
177   </complexType>
178 </element>
```

179 2.2.1.2 Example

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

182

```
183 <query:QueryRequest maxResults="-1" startIndex="0" ...>
184   <rs:ResponseOption returnComposedObjects="true"
185   returnType="LeafClassWithRepositoryItem"/>
186   <query:Query queryDefinition="urn:oasis:names:tc:ebxml-
187   regrep:query:GetObjectById">
188     <rim:Slot name="id">
189       <rim:SlotValue xsi:type="StringValue"
190       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
191         <rim:Value>%danyal%</rim:Value>
192       </rim:SlotValue>
193     </rim:Slot>
194   </query:Query>
195 </query:QueryRequest>
```

196 2.2.1.3 Description

- 197 ● [Element ResponseOption](#) - This required element allows the client to control the content of the
198 QueryResponse generated by the server in response to this request.
- 199 ● [Element Query](#) - This element identifies a parameterized query and supplies values for its
200 parameters.
- 201 ● [Attribute depth](#) - This optional attribute specifies the pre-fetch depth of the response desired by
202 the client. A depth of 0 (default) indicates that the server MUST return only those objects that
203 match the query. A depth of N where N is greater than 0 indicates that the server MUST also
204 return objects that are reachable by N levels of references via attributes that reference other

205 objects. A depth of -1 indicates that the server MUST return all objects within the transitive closure
206 of all references from objects that matches the query.

- 207 ● Attribute federated – This optional attribute specifies that the server must process this query as a
208 federated query. By default its value is *false*. This value MUST be false when a server routes a
209 federated query to another server. This is to avoid an infinite loop in federated query processing.
- 210 ● Attribute federation - This optional attribute specifies the id of the target Federation for a federated
211 query in case the server is a member of multiple federations. In the absence of this attribute a
212 server must route the federated query to all registries that are a member of all federations
213 configured within the local server. This value MUST be unspecified when a server routes a
214 federated query to another server. This is to avoid an infinite loop in federated query processing.
- 215 ● Attribute format - This optional attribute specifies the format of the response desired by the client.
216 The default value is “application/x-ebRS+xml” which returns the response in ebRS [QueryResponse](#)
217 format.
- 218 ● Attribute lang - This optional attribute specifies the natural language of the response desired by
219 the client. The default value is to return the response with all available natural languages.
- 220 ● Attribute matchOlderVersions – This optional attribute specifies the behavior when multiple
221 versions of the same object are matched by a query. When the value of this attribute is specified
222 as *false* (the default) then a server MUST only return the latest matched version for any object
223 and MUST not return older versions of such objects even though they may match the query.
224 When the value of this attribute is specified as *true* then a server MUST return all matched
225 versions of all objects.
- 226 ● Attribute maxResults - This optional attribute specifies a limit on the maximum number of results
227 the client wishes the query to return. If unspecified, the server SHOULD return either all the
228 results, or in case the result set size exceeds a server specific limit, the server SHOULD return a
229 sub-set of results that are within the bounds of the server specific limit. This attribute is described
230 further in the [Iterative Queries section](#).
- 231 ● Attribute startIndex - This optional integer value is used to indicate which result must be returned
232 as the first result when iterating over a large result set. The default value is 0, which returns the
233 result set starting with index 0 (first result). This attribute is described further in the [Iterative](#)
234 [Queries section](#).

235 **2.2.1.4 Response**

236 This request returns [QueryResponse](#) as response.

237 **2.2.1.5 Exceptions**

238 In addition to [common exceptions](#), the following exceptions MAY be returned:

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

241 **2.2.2 Element Query**

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

244 2.2.2.1 Syntax

```
245 <complexType name="QueryType">
246   <complexContent>
247     <extension base="tns:ExtensibleObjectType">
248       <attribute name="queryDefinition"
249         type="tns:objectReferenceType" use="required"/>
250     </extension>
251   </complexContent>
252 </complexType>
```

253

254 2.2.2.2 Description:

- 255 ● *Element Slot* - Each Slot element specifies a parameter value for a parameter supported by the
256 query. The slot name MUST match a parameterName attribute within a rim:Parameter definition
257 within the rim:QueryDefinition definition. The slot value provides a value for the parameter. Order
258 of parameters is not significant.
- 259 ● *Attribute query* - The value of this attribute must be a reference to a parameterized query that is
260 supported by the server.

261 2.2.3 Element ResponseOption

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

264 2.2.3.1 Syntax

```
265 <complexType name="ResponseOptionType">
266   <attribute name="returnType" default="LeafClassWithRepositoryItem">
267     <simpleType>
268       <restriction base="NCName">
269         <enumeration value="ObjectRef"/>
270         <enumeration value="RegistryObject"/>
271         <enumeration value="LeafClass"/>
272         <enumeration value="LeafClassWithRepositoryItem"/>
273       </restriction>
274     </simpleType>
275   </attribute>
276   <attribute name="returnComposedObjects"
277     type="boolean" use="optional" default="false"/>
278 </complexType>
279 <element name="ResponseOption" type="tns:ResponseOptionType"/>
```

280 2.2.3.2 Description:

- 281 ● *Attribute returnComposedObjects* - This optional attribute specifies whether the RegistryObjects
282 returned should include composed objects as defined by Figure 1 in [regrep-rim-v4.0]. The default
283 is to return all composed objects.
- 284 ● *Attribute returnType* - This optional attribute specifies the type of RegistryObject to return within
285 the response. Values for returnType are as follows:
 - 286 ○ *ObjectRef* - This option specifies that the QueryResponse MUST contain a
287 <rim:ObjectRefList> element. The purpose of this option is to return references to objects
288 rather than the actual objects.

- 289 ○ *RegistryObject* - This option specifies that the QueryResponse MUST contain a
290 <rim:RegistryObjectList> element containing <rim:RegistryObject> elements with
291 xsi:type="rim:RegistryObjectType".
 - 292 ○ *LeafClass* - This option specifies that the QueryResponse MUST contain a collection of
293 <rim:RegistryObjectList> element containing <rim:RegistryObject> elements that have an
294 xsi:type attribute that corresponds to leaf classes as defined in [regrep-xsd-v4.0]. No
295 RepositoryItems SHOULD be included for any rim:ExtrinsicObjectType instance in the
296 <rim:RegistryObjectList> element.
 - 297 ○ *LeafClassWithRepositoryItem* - This option is the same as the LeafClass option with the
298 additional requirement that the response include the RepositoryItems, if any, for every
299 rim:ExtrinsicObjectType instance in the <rim:RegistryObjectList> element.
- 300 If "returnType" specified does not match a result returned by the query, then the server MUST use the
301 closest matching semantically valid returnType that matches the result. For example, consider a case
302 where a Query that matches rim:OrganizationType instances is asked to return
303 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume the LeafClass option
304 instead.

305 2.2.4 QueryResponse

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

308 2.2.4.1 Syntax

```
309 <element name="QueryResponse">
310   <complexType>
311     <complexContent>
312       <extension base="rs:RegistryResponseType">
313         <attribute name="startIndex" type="integer" default="0"/>
314         <attribute name="totalResultCount" type="integer" use="optional"/>
315       </extension>
316     </complexContent>
317   </complexType>
318 </element>
```

319 2.2.4.2 Example

320 The following shows a sample response for the [example QueryRequest](#) presented earlier.

```
321 <query:QueryResponse totalResultCount="1" startIndex="0"
322 status="urn:oasis:names:tc:ebxml-regrep:ResponseStatusType:Success">
323   <rim:RegistryObjectList>
324     <RegistryObject xsi:type="PersonType"
325       status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
326       objectType="urn:oasis:names:tc:ebxml-
327 regrep:ObjectType:RegistryObject:Person"
328       lid="urn:acme:Person:Danyal" id="urn:acme:Person:Danyal">
329       <Name>
330         <LocalizedString value="Danyal Najmi" xml:lang="en-US"/>
331       </Name>
332       <VersionInfo versionName="1"/>
333       <PersonName lastName="Najmi" middleName="Idris" firstName="Danyal"/>
334     </RegistryObject>
335   </rim:RegistryObjectList>
336 </query:QueryResponse>
```

337 2.2.4.3 Description:

- 338 ● Element RegistryObjectList (inherited) - This is the element that contains the RegistryObject
339 instances that matched the specified query. A server MUST provide this element in a
340 QueryResponse even if it contains no RegistryObject instances.
- 341 ● Attribute startIndex - This optional integer value is used to indicate the index for the first result in
342 the result set returned by the query, within the complete result set matching the query. By default,
343 this value is 0. This attribute is described further in the [Iterative Queries](#) section.
- 344 ● Attribute totalResultCount - This optional parameter specifies the size of the complete result set
345 matching the query within the server. When this value is unspecified, the client should assume it
346 is the size of the result set contained within the result. When this value is -1, the client should
347 assume that the number of total results is unknown. In this case the client should keep iterating
348 through the remaining result set for the query until no more results are returned. This attribute is
349 described further in the [Iterative Queries](#) section.

350 2.2.5 Iterative Queries

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

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

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

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

368 2.3 Parameterized Query Definition

369 A parameterized query is defined by submitting a rim:QueryDefinitionType instance to the server using the
370 [submitObjects](#) protocol. A detailed specification of the rim:QueryDefinitionType is defined in ebRIM. The
371 definition of a parameterized query includes detailed specification of each supported parameter including
372 its name, description, data type, cardinality and domain.

373 2.4 Canonical Query: AdhocQuery

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

378 **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 |

379 **2.4.2 Query Semantics**

- 380 ● The queryExpression may specify any number of named parameters
- 381 ● The server MUST use rim:Slot child elements of the rim:Query as named parameters to the query
382 queryExpression
- 383 ● The server MUST return a QueryException fault message if the queryLanguage used by the
384 queryExpression is not supported by the server
- 385 ● The server SHOULD return an AuthorizationException fault message if the client is not authorized
386 to invoke this query
- 387 ● The server MUST return the objects matching the query if the query is processed without any
388 exceptions

389 **2.5 Canonical Query: BasicQuery**

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

392 **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 predicates for each parameter | boolean | false | 0..1 |

| | | | | |
|------------|---|-----------------|--|------|
| name | Matches rim:RegistryObject/rim:Name/rim:LocalizedString/@value | string | | 0..1 |
| objectType | Matches RegistryObjects whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value | 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 |

393 2.5.2 Query Semantics

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

401 2.6 Canonical Query: ClassificationSchemeSelector

402 The [canonical query ClassificationSchemeSelector](#) allows clients to create a Subscription to a remote

403 server to replicate a remote ClassificationScheme. This query may be used as Selector query in the

404 subscription as defined in the [object replication feature](#).

405 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 |

406 2.6.2 Query Semantics

- 407 ● The server MUST return the specified ClassificationScheme and all ClassificationNodes that are
- 408 descendants of that ClassificationScheme.

- 409 ● The ClassificationNodes MUST NOT be returned as nested elements inside their parent
410 Taxonomy element. Instead they MUST be returned as sibling elements with the
411 RegistryObjectList element of the QueryResponse.

412 2.7 Canonical Query: FindAssociations

413 The canonical query [FindAssociations](#) query allows clients to find Associations that match the specified
414 criteria.

415 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 specified value | taxonomyElement | | 0..1 |

416 2.7.2 Query Semantics

- 417 ● All parameters are optional

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

423 2.8 Canonical Query: FindAssociatedObjects

424 The [canonical query FindAssociatedObjects](#) allows clients to find RegistryObjects that are associated with
 425 the specified RegistryObject and match the specified criteria.

426 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. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character. | string | | 0..1 |
| targetObjectType | Matches source RegistryObjects of Associations whose targetObject attribute | taxonomyElement | | 0..1 |

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

427 2.8.2 Query Semantics

- 428 ● All parameters are optional
- 429 ● The server MUST return the objects matching the query if the query is processed without any
430 exceptions
- 431 ● Either sourceObjectId or targetObjectId MUST be specified. If neither are specified then
432 QueryException fault MUST be returned
- 433 ● Both sourceObjectId and targetObjectId MUST NOT be specified. If both are specified then
434 QueryException fault MUST be returned
- 435 ● Predicates for each supplied parameter are combined using an implicit LOGICAL AND if
436 matchOnAnyParameter is unspecified or false. If it is specified as true then predicates for each
437 supplied parameters are combined using a LOGICAL OR

438 2.9 Canonical Query: GarbageCollector

439 The [canonical query GarbageCollector](#) allows clients to find RegistryObjects that are deemed as garbage
440 by the server.

441 2.9.1 Parameter Summary

442 This query specifies no parameters.

443 2.9.2 Query Semantics

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

449 2.10 Canonical Query: GetAuditTrailById

450 The [canonical query GetAuditTrailById](#) allows clients to get the change history or audit trail for a
451 RegistryObject whose id attribute value is the same as the value of the id parameter.

452 2.10.1 Parameter Summary

| Parameter | Description | Data Type | Default Value | Cardinality |
|-----------|--|-----------|---------------|-------------|
| endTime | Specifies the end of the time interval (inclusive) for | dateTime | | 0..1 |

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

453 2.10.2 Query Semantics

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

461 2.11 Canonical Query: GetAuditTrailByLid

462 The [canonical query GetAuditTrailByLid](#) allows clients to get the change history or audit trail for all
463 RegistryObjects whose lid attribute value is the same as the value of the lid parameter.

464 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 rim:/RegistryObject[@xsi:type="rim:AuditableEventType"]/@timestamp value | dateTime | | 0..1 |

465 2.11.2 Query Semantics

- 466 ● The server MUST return a set of AuditableEvents that affected objects with lid matching the
467 specified lid parameter value. The set is sorted by the timestamp attribute value in descending
468 order (latest first)
- 469 ● If startTime is specified the server MUST only include AuditableEvents whose timestamp is >=
470 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: 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

2.13.1 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 | integer | 1 | 0..1 |

| | | | | |
|-----------------------|---|---------|-------|------|
| | •depth <= 0 implies get all descendants | | | |
| exclusiveChildrenOnly | Specifies how to handle children that may have multiple parents: <ul style="list-style-type: none"> • 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 | boolean | false | 0..1 |
| objectType | Specifies the type of object hierarchy for the query | string | | 0..1 |
| parentId | Specifies the id of the parent object | string | | 0..1 |

493 2.13.2 Query Semantics

- 494 ● If objectType and parentId are both unspecified the server MUST return all RegistryObjects that
495 are not members of a RegistryPackage (root level objects)
- 496 ● If parentId parameter is unspecified and objectType parameter is specified the server MUST
497 return all root level objects for the object hierarchy identified by the objectType as follows:
- 498 ○ If objectType parameter value contains the string “ClassificationScheme” the server MUST
499 return all ClassificationSchemes
- 500 ○ If objectType parameter value contains the string “Organization” the server MUST return all
501 Organizations that are not a member of another Organization (root level Organizations)
- 502 ○ If objectType parameter value contains the string “RegistryPackage” the server MUST return
503 all RegistryPackages that are not a member of another RegistryPackage (root level
504 RegistryPackages)
- 505 ● If parentId parameter is specified then the behavior is as follows:
- 506 ○ If objectType parameter value is unspecified or if its value contains the string
507 “RegistryPackage” the server MUST return all RegistryObjects that are member of a
508 RegistryPackage whose id is the same as the value of the parentId attribute
- 509 ○ If objectType parameter is specified and its value contains the string “ClassificationScheme”
510 the server MUST return all ClassificationNodes that are children of a TaxonomyElementType
511 instance whose id is the same as the value of the parentId attribute
- 512 ○ If objectType parameter is specified and its value contains the string “Organization” the server
513 MUST return all Organizations that are members of an Organization whose id is the same as
514 the value of the parentId attribute
- 515 ● If depth parameter is specified then the server MUST also return all descendants upto the
516 specified depth as described by the definition of the depth parameter above
- 517 ● If exclusiveChildrenOnly is specified with a true value then the server MUST not return any
518 descendants that have multiple parents

519 2.14 Canonical Query: GetClassificationSchemesById

520 The [canonical query GetClassificationSchemesById](#) allows clients to fetch specified
521 ClassificationSchemes.

522 2.14.1 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 |

523 2.14.2 Query Semantics

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

528 2.15 Canonical Query: GetRegistryPackagesByMemberId

529 The [canonical query GetRegistryPackagesByMemberId](#) allows clients to get the RegistryPackages that a
530 specified RegistryObject is a member of.

531 2.15.1 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 |

532 2.15.2 Query Semantics

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

535 2.16 Canonical Query: GetNotification

536 The [canonical query GetNotification](#) allows clients to “pull” any pending Notification for a Subscription at a
537 time of their choosing. This is defined in detail under section titled “[Pulling Notification on Demand](#)”.

538 2.16.1 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 |

539 2.16.2 Query Semantics

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

543 2.17 Canonical Query: GetObjectById

544 The [canonical query GetObjectById](#) allows clients to find RegistryObjects based upon the value of their id
545 attribute.

546 2.17.1 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 |

547 2.17.2 Query Semantics

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

550 2.18 Canonical Query: GetObjectsByLid

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

554 2.18.1 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 |

555 2.18.2 Query Semantics

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

558 2.19 Canonical Query: GetReferencedObject

559 The [canonical query GetReferencedObject](#) allows clients to get a RegistryObject that is the target of an
560 rim:objectReferenceType attribute value.

561 2.19.1 Parameter Summary

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

562 2.19.2 Query Semantics

- 563 ● The server MUST return the RegistryObjectType instance that is being referenced by the
564 specified value for the objectReference parameter.
 - 565 ○ If the objectReference contains the id of a local object that is not a DynamicObjectRef
566 instance then the server MUST return that object.
 - 567 ○ If the objectReference contains the id of a local DynamicObjectRef instance then the server
568 MUST invoke the Query within the DynamicObjectRef instance and resolve the reference to
569 the singleton result of the Query and return the matching object.
 - 570 ○ If the objectReference contains the [canonical URL](#) for a remote object then the server MUST
571 invoke the GetReferencedObject query against the remote server using the id of the remote
572 object as the value of the objectReference parameter and return the matching object. The id
573 of the remote object is accessible from its canonical URL as the value of the id parameter
574 within the URL.

575 2.20 Canonical Query: KeywordSearch

576 The [canonical query KeyWordSearch](#) allows clients to find RegistryObjects and RepositoryItems that
577 contain text that matches keywords identified by specified search patterns.

578 **2.20.1 Canonical Indexes**

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

583

| 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 |

584

585

586 **2.20.2 Parameter Summary**

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

587 2.20.3 Query Semantics

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

590 Example: ebxml regrep
591 Semantics: Matches objects containing either “ebxml” or “regrep”
592

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

595 Example: “ebxml regrep”
596 Semantics: Matches objects containing “ebxml regrep”
597

- 599 ● Terms may be specified using wildcard characters where “*” matches one or more characters and
600 “?” matches a single character.

601 Example: eb?ml reg*
602

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

- 606 ●
607 Example: ebxml AND regrep
608 Semantics: Matches objects containing “ebxml” and “regrep”
609

- 610 Example: ebxml NOT regrep
611 Semantics: Matches objects containing “ebxml” and not containing “regrep”
612

- 613 Example: ebxml OR regrep
614 Semantics: Matches objects containing “ebxml” or “regrep”
615

- 616 Example: ebxml regrep
617 Semantics: Matches objects containing “ebxml” or “regrep”

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

- 622 Example: ebxml AND (registry OR regrep)
623 Semantics: Matches objects containing both “ebxml” and either “registry” or “regrep”
624

- 625 ● The server MUST return all RegistryObjects that contain indexed data matching the semantics of
626 the keywords parameter.

- 627 ● The server MUST return all ExtrinsicObjects that have a repository item that contains indexed
628 data matching the semantics of the keywords parameter.

629 2.21 Canonical Query: RegistryPackageSelector

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

634 2.21.1 Parameter Summary

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

635 2.21.2 Query Semantics

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

643 2.22 Query Functions

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

649 This specification defines a number of [canonical functions](#) that are standard functions that MUST be
650 supported by a server. Profiles, implementations and deployments may define additional query functions
651 beyond the canonical functions defined by this specification.

652 2.22.1 Using Functions in Query Expressions

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

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

- 662 ● Allows a parser to recognize the non-function parts of the query expression that MUST be
663 preserved as *is*
- 664 ● Allows implementations to be optimized to skip function parsing and evaluation if the special
665 delimiter characters are not present in query expression

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

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

```

673 a.sourceObject = :sourceObject AND
674 a.type IN (@# getClassificationNodes("urn:oasis:names:tc:ebxml-
675 regrep:AssociationType:AffiliatedWith", 0, 2, "false", "", "${id}") #e) AND
676 targetObject.id = a.targetObject@#
677

```

678 2.22.2 Using Functions in Query Parameters

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

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

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

```

690 --Following is the query expression within the server
691 --This time it has no query functions as they are in the query parameters
692 SELECT targetObject.* FROM
693 RegistryObjectType targetObject, AssociationType a WHERE
694
695 a.sourceObject = :sourceObject AND
696 a.type IN ( :types ) AND
697 targetObject.id = a.targetObject

```

698

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

702

```

703 <query:QueryRequest maxResults="-1" startIndex="0" ...>
704   <rs:ResponseOption returnComposedObjects="true"
705   returnType="LeafClassWithRepositoryItem"/>
706   <query:Query queryDefinition="urn:acme:ExampleQuery">
707     <rim:Slot name="sourceObject">
708       <rim:SlotValue xsi:type="StringValue"
709       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
710         <rim:Value>urn:test:Person:Danyal</rim:Value>
711       </rim:SlotValue>
712     <rim:Slot name="types">
713       <rim:SlotValue xsi:type="StringValue"
714       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
715         <rim:Value>getClassificationNodes("urn:oasis:names:tc:ebxml-
716         regrep:AssociationType:AffiliatedWith", 0, 2, "false", "", "${
717         id}")#@@#</rim:Value>
718       </rim:SlotValue>
719     </rim:Slot>
720   </query:Query>
721 </query:QueryRequest>

```

722 2.22.3 Function Processing Model

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

- 725 ● When processing a query expression elements (rim:QueryDefinition/rim:QueryExpression) the
726 server SHOULD NOT perform function processing if the special delimiter sequences of “#@” and
727 “@#” are not found in the query expression
- 728 ● When processing query invocation parameter elements
729 (query:QueryRequest/query:Query/rim:Slot/rim:SlotValue) the server SHOULD NOT perform
730 function processing if the special delimiter sequences of “#@” and “@#” are not found in the query
731 expression
- 732 ● When processing a query expression element if the special delimiter sequences of “#@” and
733 “@#” are found then the server MUST process query expression elements to replace all function
734 invocations with the value returned when the function is invoked with specified parameters
- 735 ● When processing query invocation parameter elements if the special delimiter sequences of “#@”
736 and “@#” are found then the server MUST process each query parameter element to replace all
737 function invocations with the value returned when the function is invoked with specified
738 parameters
- 739 ● When invoking a function that has another function invocation as its parameter the inner most
740 functions MUST be invoked first so that the outer function can be invoked with the value returned
741 by the inner function invocation
- 742 ● When processing a query expression or query parameter the special delimiter characters “#@”
743 and “@#” MUST be removed and the value contained within them MUST be preserved without
744 any change

745 2.22.4 Function Processor BNF

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

749

```
750 <DEFAULT> SKIP : {  
751 " "  
752 | "\t"  
753 | "\r"  
754 | "\n"  
755 }  
756  
757  
758  
759 <DEFAULT> TOKEN : {  
760 <FLOAT: <INTEGER> "." <INTEGER> | "." <INTEGER> | <INTEGER> ".">  
761 | <INTEGER: (<DIGIT>)+>  
762 | <DIGIT: ["0"- "9"]>  
763 | <BOOLEAN: "true" | "false">  
764 }  
765  
766  
767  
768 <DEFAULT> TOKEN : {  
769 <S_IDENTIFIER: (<LETTER>)+ (<DIGIT> | <LETTER> | <SPECIAL_CHARS>)*>  
770 | <#LETTER: ["a"- "z", "A"- "Z"]>  
771 | <#SPECIAL_CHARS: "_ ">
```

```

772 | <S_CHAR_LITERAL: "\"'\" (~[\"'\"])* \"'\" (\"'\"] (~[\"'\"])* \"'\")*>
773 | <S_QUOTED_IDENTIFIER: "\"\" (~[\"\\n\", \"\\r\", \"\\\"])* \"\">
774 | <OPENPAREN: \"(>
775 | <CLOSEPAREN: \")\">
776 | <COMMA: \",\">
777 | <COLON: \":\">
778 | <DELIMITED_TEXT: \"#@\" (~[\"@\"])* \"@#\">
779 | }
780
781 start ::= ( textOrFunctionCall )+ <EOF>
782 text ::= ( ( <DELIMITED_TEXT> ) )
783 textOrFunctionCall ::= ( text | FunctionCall )
784 FunctionCall ::= FunctionReference <OPENPAREN> ( FunctionArgumentList )*
785 <CLOSEPAREN>
786 FunctionReference ::= <S_IDENTIFIER> <COLON> <S_IDENTIFIER>
787 FunctionArgumentList ::= FunctionArgument ( <COMMA> FunctionArgument )*
788 FunctionArgument ::= ( FunctionCall | <S_CHAR_LITERAL> |
789 <S_QUOTED_IDENTIFIER> | <FLOAT> | <INTEGER> | <BOOLEAN> )

```

790 2.23 Common Patterns In Query Functions

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

794 2.23.1 Specifying a null Value for string Param or Return Value

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

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

799 2.24 Canonical Functions

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

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

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

809 Table 1 summarizes the canonical functions defined by this specification.

810

| 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. |

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

812 **2.24.1 Canonical Function: currentTime**

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

814 **2.24.1.1 Function Semantics**

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

818 **2.24.2 Canonical Function: currentUserId**

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

822 **2.24.2.1 Function Semantics**

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

825 **2.24.3 Canonical Function: relativeTime**

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

828 **2.24.3.1 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 |

829 2.24.3.2 Function Semantics

- 830 ● The server MUST return a string if the query is processed without any exceptions
- 831 ● The format of the duration parameter MUST conform to the format as specified by the duration
832 type defined by XML Schema duration type otherwise the server MUST return
833 InvalidRequestException
- 834 ● The value of the string MUST be a time in ISO 8601 format that is offset by the specified period in
835 the future relative to the current time. An example of value returned is “2010-02-
836 25T15:22:14.534Z”

837 2.24.4 Canonical Function: getClassificationNodes

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

842 2.24.4.1 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 |

843 2.24.4.2 Function Semantics

- 844 ● The server MUST return a string if the query is processed without any exceptions
- 845 ● The string MUST be “rs:null” if no ClassificationNode is found that matches the function
846 parameters
- 847 ● The string MUST consist of a set of substrings separated by the appropriate delimiter character
848 when any ClassificationNode's are found that match the function parameters:
 - 849 ○ There MUST be a substring for each ClassificationNode matched by the function
 - 850 ○ Each substring MUST conform to the specified template such that all occurrences of \${id} are
851 replaced by the id of a ClassificationNode matched by the function
- 852 ● The id of the reference ClassificationNode MUST be included if and only if the includeSelf
853 parameter value is true

- 854 ● A ancestorLevels value of N where N > 0 matches all ClassificationNodes upto the Nth level
855 ancestors of the reference ClassificationNode. A value of 1 matches the immediate parents of the
856 reference ClassificationNode while a value of 2 matches the parents and grandparents of the
857 reference ClassificationNode. A value of -1 matches all ancestors of the reference
858 ClassificationNode
- 859 ● A descendantsLevels value of N where N > 0 matches all ClassificationNodes upto the Nth level
860 descendants of the reference ClassificationNode. A value of 1 matches the immediate children of
861 the reference ClassificationNode while a value of 2 matches the children and grandchildren of the
862 reference ClassificationNode. A value of -1 matches all descendants of the reference
863 ClassificationNode
- 864 ● A template value of “rs:null” is implicitly equivalent to a template value of “\${id}”
865

866 2.25 Query Plugins

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

871 2.25.1 Query Plugin Interface

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

3 LifecycleManager Interface

876

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

881 A server MUST implement the LifecycleManager interface as an endpoint.

3.1 SubmitObjects Protocol

882

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

885 A client initiates the SubmitObjects protocol by sending a SubmitObjectsRequest message to the
886 LifecycleManager endpoint.

887 The LifecycleManager sends a [RegistryResponse](#) back to the client as response.

888



Illustration 2: SubmitObjects Protocol

3.1.1 SubmitObjectsRequest

890

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

3.1.1.1 Syntax

892

```
893 <simpleType name="mode">
894   <restriction base="NCName">
895     <enumeration value="CreateOrReplace"/>
896     <enumeration value="CreateOrVersion"/>
897     <enumeration value="CreateOnly"/>
898   </restriction>
899 </simpleType>
900
901 <element name="SubmitObjectsRequest">
902   <complexType>
903     <complexContent>
904       <extension base="rs:RegistryRequestType">
905         <sequence>
906           <element ref="rim:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
907         </sequence>
908         <attribute name="checkReferences" type="boolean" use="optional"
909           default="false"/>
910       </extension>
911     </complexContent>
912   </complexType>
913 </element>
```



```
910     <attribute name="mode" type="tns:mode" use="optional"
911         default="CreateOrReplace"/>
912     </extension>
913 </complexContent>
914 </complexType>
915 </element>
```

916 3.1.1.2 Description

- 917 ● Element RegistryObjectList - Specifies a set of RegistryObject instances that are being submitted
918 to the server. The RegistryObjects in the list may be new objects being submitted to the server or
919 they may be current objects already existing in the server.
- 920 ● Attribute checkReferences – Specifies the reference checking behavior expected of the server
 - 921 ○ true - Specifies that a server MUST check submitted objects and make sure that all
922 references via reference attributes and slots to other RegistryObjects are resolvable. If a
923 reference does not resolve then the server MUST return UnresolvedReferenceException
 - 924 ○ false (default) – Specifies that a server MUST NOT check submitted objects to make sure
925 that all references via reference attributes and slots to other RegistryObjects are resolvable. If
926 a reference does not resolve then the server MUST NOT return
927 UnresolvedReferenceException
- 928 ● Attribute mode – Specifies the semantics for how the server should handle RegistryObjects being
929 submitted when they already exist in the server:
 - 930 ○ CreateOrReplace (default) - If an object does not exist, server MUST create it as a new
931 object. If an object already exists, server MUST replace the existing object with the submitted
932 object
 - 933 ○ CreateOrVersion - If an object does not exist, server MUST create it as a new object. If an
934 object already exists, server MUST not alter the existing object and instead it MUST create a
935 new version of the existing object using the state of the submitted object
 - 936 ○ CreateOnly - If an object does not exist, server MUST create it as a new object. If an object
937 already exists, the server MUST return an ObjectExistsException fault message

938 3.1.1.3 id and lid Requirements

939 Table 2 defines the requirements for id and lid attribute values for RegistryObjectType instances that are
940 submitted via the SubmitObjects protocol.

941

| Mode / Requirements | ID Requirements | LID Requirements |
|---------------------|--|---|
| CreateOrReplace | <ul style="list-style-type: none"> ● MUST be specified by client or else server MUST return InvalidRequestException ● If id does not exist, server MUST create new object using that id (create) ● If id exists, server MUST replace existing object matching that id (update) | <ul style="list-style-type: none"> ● MUST be specified by client or else server MUST return InvalidRequestException |
| CreateOrVersion | <ul style="list-style-type: none"> ● MUST be specified by client or else server MUST return InvalidRequestException ● If id does not exist and lid does not exist, server MUST create new object using that id (create) ● If id does not exist 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) | <ul style="list-style-type: none"> ● MUST be specified by client or else server MUST return InvalidRequestException |
| CreateOnly | <ul style="list-style-type: none"> ● MAY be specified by client ● If unspecified Server MUST generate UUID URN ● If id does not exist, server MUST create new object using that id (create) ● If id exists, server MUST return ObjectExistsException | <ul style="list-style-type: none"> ● MUST be specified by client or else server MUST return InvalidRequestException ● MUST NOT exist or else server MUST return ObjectExistsException |

Table 2: Requirements for id and lid During SubmitObjects Protocol

942

943

944 3.1.1.4 Returns

945 This request returns a [RegistryResponse](#).

946 3.1.1.5 Exceptions

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

950 3.1.2 Audit Trail Requirements

- 951 ● The server **MUST** create a single AuditableEvent object as follows:
 - 952 ○ If RegistryObjects were created by the request, it contain a single Action sub-element with
953 event Type *Created* for all the RegistryObjects created during processing of the request
 - 954 ○ If RegistryObjects were updated by the request, it contain a single Action sub-element with
955 event Type *Updated* for all the RegistryObjects updated during processing of the request
- 956 ● The server **SHOULD** create AuditableEvents *after* successfully processing the request in a
- 957 separate transaction from the request

958 3.1.3 Sample SubmitObjectsRequest

959 The following simplified example shows a SubmitObjectsRequest that submits a single Organization

960 object to the server.

961

```
962 <lcm:SubmitObjectsRequest>
963   <rim:RegistryObjectList>
964     <rim:RegistryObject xsi:type="rim:OrganizationType" lid="{LOGICAL_ID}"
965       id="{ID}" ...>
966     ...
967   </rim:RegistryObject>
968 </rim:RegistryObjectList>
969 </SubmitObjectsRequest>
```

972 3.2 The Update Objects Protocol

973 The UpdateObjectsRequest protocol allows a client to make partial updates to one or more

974 RegistryObjects that already exist in the server. This protocol enables *partial* update of RegistryObjects

975 rather than a *complete replacement*. A client **SHOULD** use the SubmitObjects protocol for complete

976 replacement of RegistryObjects.

977 A server **MUST** return InvalidRequestException fault message if the client attempts to update the id, lid or

978 objectType attribute of a RegistryObject.

979

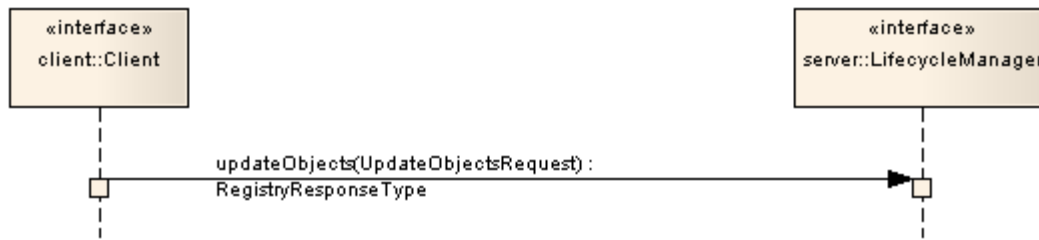


Illustration 3: UpdateObjects Protocol

981 3.2.1 UpdateObjectsRequest

982 The UpdateObjectsRequest message is sent by a client to partially update existing RegistryObjects in the
 983 server. An UpdateObjectsRequest identifies a set of RegistryObjects as target objects to be updated by
 984 the request. It also specifies the update action that modifies each target object. Update actions may insert
 985 a node within a target object, delete an existing node from a target object or update an existing node
 986 within the target object. A node in the context of the UpdateObjects protocol is defined to be an XML DOM
 987 node (typically an element or an attribute).

988 3.2.1.1 Syntax

```

989 <element name="UpdateObjectsRequest">
990   <complexType>
991     <complexContent>
992       <extension base="rs:RegistryRequestType">
993         <sequence>
994           <!-- Query and ObjectRefList select objects to update -->
995           <element name="Query" type="rim:QueryType" minOccurs="0" maxOccurs="1" />
996           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
997
998           <!-- Specifies how to update selected objects -->
999           <element name="UpdateAction" type="tns:UpdateActionType"
1000             minOccurs="1" maxOccurs="unbounded"/>
1001         </sequence>
1002         <attribute name="checkReferences" type="boolean" use="optional"
1003           default="false"/>
1004         <attribute name="mode" type="tns:mode" use="optional"
1005           default="CreateOrReplace"/>
1006       </extension>
1007     </complexContent>
1008   </complexType>
1009 </element>
  
```

1010 3.2.1.2 Description

- 1011 ● Element Query - Specifies a query to be invoked. A server MUST use all objects that match the
 1012 specified query in addition to any other objects identified by the ObjectRefList element as targets
 1013 of the update action.
- 1014 ● Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances
 1015 in the server. A server MUST use all objects that are referenced by this element in addition to any
 1016 other objects identified by the Query element as targets of the update action.
- 1017 ● Element UpdateAction – Specifies the details of how to update the target objects

- 1018 ● Attribute checkReferences – Specifies the reference checking behavior expected of the server
 - 1019 ○ true - Specifies that a server MUST check updated objects and make sure that all references
 - 1020 via reference attributes and slots to other RegistryObjects are resolvable. If a reference does
 - 1021 not resolve then the server MUST return UnresolvedReferenceException
 - 1022 ○ false (default) – Specifies that a server MUST NOT check updated objects to make sure that
 - 1023 all references via reference attributes and slots to other RegistryObjects are resolvable. If a
 - 1024 reference does not resolve then the server MUST NOT return UnresolvedReferenceException
- 1025 ● Attribute mode – Specifies the semantics for how the server should handle RegistryObjects being
 - 1026 updated in the server:
 - 1027 ○ CreateOrReplace (default) - If an object does not exist, server MUST return
 - 1028 ObjectNotFoundException. If an object already exists, server MUST update the existing object
 - 1029 without creating a new version
 - 1030 ○ CreateOrVersion - If an object does not exist, server MUST return ObjectNotFoundException.
 - 1031 If an object already exists, server MUST create a new version of the existing object before
 - 1032 applying the requested update action
 - 1033 ○ CreateOnly – This mode does not apply to UpdateObjectsRequest. If specified, server MUST
 - 1034 return an InvalidRequestException

1035 3.2.1.3 Returns

1036 This request returns a [RegistryResponse](#).

1037 3.2.1.4 Exceptions

- 1038 ● A server MUST return an UnsupportedCapabilityException fault message if the request contains a
 - 1039 type that is an extension of types defined by ebRIM and if the server cannot support such
 - 1040 extension.

1041 3.2.2 UpdateAction

1042 An UpdateRequest contains one or more UpdateActions. Each UpdateObjectsRequest defines a specific
 1043 update action to be performed on each target object.

1044 3.2.2.1 Syntax

```

1045 <complexType name="UpdateActionType">
1046   <annotation>
1047     <documentation xml:lang="en">
1048       </documentation>
1049     </annotation>
1050   <sequence>
1051     <!-- Value for attribute or element -->
1052     <element name="ValueHolder" type="rim:ValueType"
1053       minOccurs="0" maxOccurs="1"/>
1054     <!--
1055     Value of selector is an XPATH expression that uniquely identifies
1056     an attribute or an element within target documents.
1057     -->
1058     <element name="Selector" type="rim:QueryExpressionType"
1059       minOccurs="1" maxOccurs="1"/>
1060   </sequence>
1061 </complexType>
1062 <!--
  
```

```

1063 Specifies whether to insert, update or delete a node from
1064 target document.
1065 -->
1066 <attribute name="mode" use="required">
1067   <simpleType>
1068     <restriction base="NCName">
1069       <enumeration value="Insert"/>
1070       <enumeration value="Update"/>
1071       <enumeration value="Delete"/>
1072     </restriction>
1073   </simpleType>
1074 </attribute>
1075 </complexType>

```

1076 3.2.2.2 Description

- 1077 ● Element Selector – Is a QueryExpressionType that contains the expression that identifies a node
- 1078 of the resource representation to be updated.

1079 The value of this element MUST conform to the queryLanguage specified in the queryLanguage

1080 attribute of the Selector. A resource MUST generate an QueryException fault if the expression is

1081 invalid. If the expression syntax is not valid with respect to the queryLanguage then a resource

1082 SHOULD specify a fault detail of "InvalidExpressionSyntaxException". If the expression value is

1083 not valid for the resource type then the resource SHOULD specify a fault detail of

1084 "InvalidExpressionValueException".

1085 A server MUST minimally support XPATH 1.0 as the queryLanguage for Selector element. The

1086 scope of the XML document that is processed by the XPATH expression is the

1087 RegistryObjectType instance. A server MUST implicitly support the standard namespace prefixes

1088 used by RegRep schemas (rim:, query:, rs:, lcm:, spi:) as a notational convenience. These

1089 standard namespace prefixes should map to the latest version of the specification supported by

1090 the server.

1091 An XPATH selector expression MUST be specified using the RegistryObject being updated as the

1092 context node.

1093 An XPATH selector expression may select an attribute or an element relative to the

1094 RegistryObject context node. If it selects an attribute then the ValueHolder element should use a

1095 ValueType subtype for a primitive type (instead of AnyValueType) that corresponds to the

1096 primitive type for the attribute (e.g. StringValueType). The ValueHolder/Value element's content

1097 shall contain the attribute value.

- 1102 ● Element ValueHolder - This element contains the value to be written to the target object. If the
- 1103 mode attribute is "Insert" or "Update" then this element MUST be present. If the mode is "Delete"
- 1104 then this element MUST NOT be present.

- 1105 ● Attribute mode – This attribute specifies the semantics for how the server should update target
- 1106 objects:

- 1107 ○ Insert - Indicates that the value provided by ValueHolder MUST be added to the target object.
- 1108 If the selector targets a repeated element (maxOccurs > 1), the node MUST be added at the
- 1109 end. If the selector targets a non-repeated element (maxOccurs = 1) that already exists, the
- 1110 resource MUST generate an InvalidRequestException with a fault detail of
- 1111 NodeAlreadyExistsException. If the selector targets an existing item of a repeated element,
- 1112 the value provided by ValueHolder MUST be added before the existing item.
- 1113 ○ Update – Indicates that the node identified by selector MUST be replaced by value by the
- 1114 ValueHolder in its place. If the selector resolves to nothing then there should be no change to
- 1115 the target object.

- 1116 ○ Delete - indicates that the node identified by selector MUST be deleted from the target object
1117 if it is present.

1118 3.2.3 Audit Trail Requirements

- 1119 ● The server MUST create a single AuditableEvent object as follows:
- 1120 ○ If RegistryObjects were updated by the request, it contain a single Action sub-element with
1121 eventType Updated for all the RegistryObjects updated during processing of the request
- 1122 ● The server SHOULD create AuditableEvents *after* successfully processing the request in a
1123 separate transaction from the request

1124 3.2.4 Sample UpdateObjectsRequest

1125 The following example shows an UpdateObjectsRequest which updates the Name element within a
1126 PersonType instance with the Name element specified by the Value element within UpdateAction. The
1127 Selector element uses an XPATH expression to select the Name element node within the Person objects
1128 identified as target of update in the ObjectRefList. The context node of the XPATH expression is the
1129 RegistryObject element for the PersonType instance. The target objects could also have been chosen by
1130 a Query element.

```
1131 <UpdateObjectsRequest ...>  
1132   <rim:ObjectRefList>  
1133     <rim:ObjectRef id="urn:acme:person:Danyal"/>  
1134   </rim:ObjectRefList>  
1135   <UpdateAction mode="Update">  
1136     <Value xsi:type="rim:AnyValueType">  
1137       <rim:Name>  
1138         <rim:LocalizedString xml:lang="en-US" value="Danny"/>  
1139       </rim:Name>  
1140     </Value>  
1141     <Selector xsi:type="rim:StringQueryExpressionType"  
1142       queryLanguage="urn:oasis:names:tc:exml-regrep:QueryLanguage:XPath">  
1143  
1144       <rim:Value>./rim:Name</rim:Value>  
1145     </Selector>  
1146   </UpdateAction>  
1147 </UpdateObjectsRequest>
```

1148 3.3 RemoveObjects Protocol

1149 The Remove Objects protocol allows a client to remove or delete one or more RegistryObject instances
1150 from the server.

1151 A client initiates the RemoveObjects protocol by sending a RemoveObjectsRequest message to the
1152 LifecycleManager endpoint.

1153 The LifecycleManager sends a [RegistryResponse](#) back to the client as response.

1154

1155

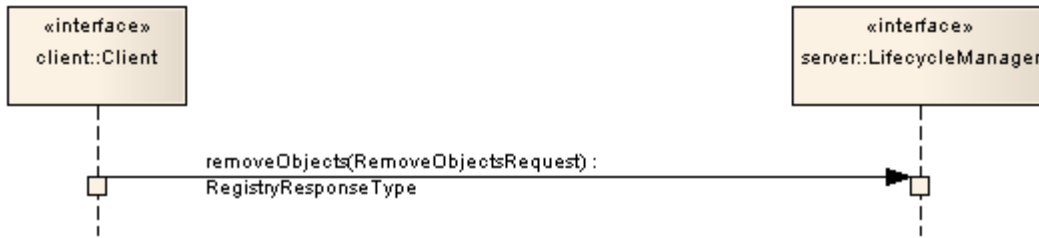


Illustration 4: RemoveObjects Protocol

1156 3.3.1 RemoveObjectsRequest

1157 The RemoveObjectsRequest message is sent by a client to remove one or more existing RegistryObjects
 1158 from the server.

1159 3.3.1.1 Syntax

```

1160 <element name="RemoveObjectsRequest">
1161   <complexType>
1162     <complexContent>
1163       <extension base="rs:RegistryRequestType">
1164         <sequence>
1165           <element name="Query" type="rim:QueryType"
1166             minOccurs="0" maxOccurs="1" />
1167           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
1168         </sequence>
1169         <attribute name="checkReferences" type="boolean" use="optional"
1170           default="false"/>
1171         <attribute name="deleteChildren" type="boolean" use="optional"
1172           default="false"/>
1173         <attribute name="deletionScope" type="rim:objectReferenceType"
1174           use="optional" default="urn:oasis:names:tc:ebxml-
1175   regrep:DeletionScopeType:DeleteAll"/>
1176       </extension>
1177     </complexContent>
1178   </complexType>
1179 </element>

```

1180

1181 3.3.1.2 Description

- 1182 ● Attribute checkReferences – Specifies the reference checking behavior expected of the server
 - 1183 ○ true - Specifies that a server MUST check objects being removed and make sure that there
 - 1184 are no references to them from other objects via reference attributes and slots. If a reference
 - 1185 exists then the server MUST return ReferencesExistsException
 - 1186 ○ false (default) – Specifies that a server MUST NOT check objects being removed to make
 - 1187 sure that there are no references to them from other objects via reference attributes and slots.
 - 1188 If a reference exists then the server MUST NOT return ReferencesExistsException
- 1189 ● Attribute deleteChildren – This attribute specifies whether or not to delete children of the objects
 1190 being deleted according to the following behavior:
 - 1191 ○ false – Specifies the server MUST NOT delete the children of objects that are specified to be
 - 1192 deleted

- 1193 ○ true – Specifies the server MUST delete children of objects being deleted if and only if those
1194 children are not children of any other parent objects
- 1195 ● Attribute deletionScope - This attribute specifies the scope of impact of the
1196 RemoveObjectsRequest. The value of the deletionScope attribute MUST be a reference to a
1197 ClassificationNode within the canonical DeletionScopeType ClassificationScheme as described in
1198 ebRIM. A server MUST support the deletionScope types as defined by the canonical
1199 DeletionScopeType ClassificationScheme. The canonical DeletionScopeType
1200 ClassificationScheme may be extended by adding additional ClassificationNodes to it.
1201
- 1202 The following canonical ClassificationNodes are defined for the DeletionScopeType
1203 ClassificationScheme:
- 1204 ○ DeleteRepositoryItemOnly - Specifies that the server MUST delete the RepositoryItem for the
1205 specified ExtrinsicObjects but MUST NOT delete the specified ExtrinsicObjects
- 1206 ○ DeleteAll (default) - Specifies that the request MUST delete both the RegistryObject and the
1207 RepositoryItem (if any) for the specified objects
- 1208 ● Element Query - Specifies a query to be invoked. A server MUST remove all objects that match
1209 the specified query in addition to any other objects identified by the ObjectRefList element.
- 1210 ● Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances
1211 in the server. A server MUST remove all objects that are referenced by this element in addition to
1212 any other objects identified by the Query element.

1213 3.3.1.3 Returns:

1214 This request returns a [RegistryResponse](#).

1215 3.3.1.4 Exceptions:

1216 In addition to the exceptions common to all requests, the following exceptions MAY be returned:

- 1217 ● UnresolvedReferenceException - Indicates that the requestor referenced an object within the
1218 request that was not resolved during the processing of the request.
- 1219 ● ReferencesExistException - Indicates that the requestor attempted to remove a RegistryObject
1220 while references to it still exist. Note that it is valid to remove a RegistryObject and all
1221 RegistryObjects that refer to it within the same request. In such cases the
1222 ReferencesExistException MUST not be thrown.

1223 3.3.2 Audit Trail Requirements

- 1224 ● The server MUST create a single AuditableEvent object as follows:
- 1225 ○ If RegistryObjects were removed by the request, it contain a single Action sub-element with
1226 eventType Deleted for all the RegistryObjects removed during processing of the request
- 1227 ● The server SHOULD create AuditableEvents *after* successfully processing the request in a
1228 separate transaction from the request

1229 3.3.3 Sample RemoveObjectsRequest

1230 The following is a sample RemoveObjectsRequest to remove an Object by its id.

1231 `<lcm:RemoveObjectsRequest ...>`

```
1232     <rim:ObjectRefList>
1233         <rim:ObjectRef id="urn:acme:Person:Danyal"/>
1234     </rim:ObjectRefList>
1235 </lcm:RemoveObjectsRequest>
```

1236 4 Version Control

1237 This section describes the version control features of the ebXML RegRep.

1238 Versioning of a RegistryObjectType instance is the process of updating the object in such a way that the
1239 original instance remains unchanged while a new instance is created as a new version of the original
1240 instance. Any specific version of an object may itself be versioned. Thus in general the versions of an
1241 object form a tree structure referred to as the Version Tree for that object.

1242 A *Version Tree* for an object is defined to be a tree structure where:

- 1243 ● There is a single root node for the tree
- 1244 ● The root is the original version
- 1245 ● Each non-root node in the tree is a version of the object
- 1246 ● Each version is created from a parent version and is represented in the version tree as a child
1247 node of the node representing the parent version node for that version



Illustration 5: A visual example of a version tree

1249 Illustration 5 visualizes the version tree concept. In this non-normative example the object TestRegister
1250 has 8 versions. Each node's version is identified by the parenthesized string suffix like "(1.2.2)". Version 1
1251 is the original version. Version 1 was versioned twice to create versions 1.1 and 1.2. Version 1.1 was
1252 versioned twice to create versions 1.1.1 and 1.1.2. Version 1.2 was versioned twice to create versions
1253 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
1254 version naming convention for ease of understanding only. This specification does not prescribe a specific
1255 version naming convention for server to use when assigning version names.

1256 The terms "logical object" or "logical RegistryObject" are used to refer to all version of a RegistryObject in
1257 a version independent manner. The terms "object version" or "RegistryObject version" are used to refer to
1258 a specific version of the logical object. The terms "RegistryObject instance" and "RegistryObjectType
1259 instance" imply a specific object version.

1260 Illustration 5 visualizes a single logical object TestRegister with 8 object versions.

1261 4.1 Version Controlled Resources

1262 Version controlled resources are resources that support versioning capability.

1263 All repository items in an ebXML RegRep are implicitly version-controlled resources as defined by section
1264 2.2.1 of [DeltaV]. No explicit action is required to make them a version-controlled resource.

1265 Instances of RegistryObjectType types are also implicitly version-controlled resources. The only
1266 exceptions are those sub-types of RegistryObjectType that are composed¹ types and their instances do
1267 not have independent lifecycles that are separate from the lifecycle of their parent objects. Some example
1268 of such composed types are:

- 1269 ● ClassificationType
- 1270 ● ExternalIdentifierType
- 1271 ● ExternalLinkType
- 1272 ● ServiceEndpointType

1273 A server MAY further limit specific non-composed types from being version-controlled resources based
1274 upon server specific policies.

1275 4.2 Versioning and Id Attribute

1276 Each object version of a logical RegistryObject is a unique object and as such has its own unique value for
1277 its id attribute as defined by [regrep-rim-v4.0].

1278 4.3 Versioning and Lid Attribute

1279 A RegistryObject instance MUST have a *Logical ID (LID)* defined by its “lid” attribute to identify the logical
1280 RegistryObject of which it is a version. All versions of a logical RegistryObject have the same “lid” attribute
1281 value. Note that this is in contrast with the “id” attribute that MUST be unique for each version of the same
1282 logical RegistryObject. A client may refer to the logical RegistryObject in a version independent manner
1283 using its LID.

1284 4.4 Version Identification for RegistryObjectType

1285 A RegistryObjectType instance MUST have a VersionInfo element whose type is the VersionInfoType type
1286 defined by ebRIM. The VersionInfo element identifies the version information for that RegistryObjectType
1287 instance. The versionName attribute of the VersionInfo element identifies the version name for a specific
1288 version of a logical object. A server MUST not allow two versions of the same logical object to have the
1289 same versionName attribute value within its VersionInfo element.

1290 4.5 Version Identification for RepositoryItem

1291 When a RegistryObject is an ExtrinsicObject with an associated repository item, the version identification
1292 for the repository item is distinct from the version identification for the ExtrinsicObject.

1293 An ExtrinsicObject that has an associated repository item MUST have a contentVersionInfo element
1294 whose type is VersionInfoType defined by ebRIM. The contentVersionInfo attributes identifies the version
1295 information for that repository item instance.

1296 4.5.1 Versioning of RegistryObjectType

1297 This section describes the versioning of all RegistryObjectType types with the exception of
1298 ExtrinsicObjectType which is defined [in a separate section](#).

¹ Composed object types are identified in class diagrams in [regrep-rim-v4.0] as classes with composition or “solid diamond” relationship with a RegistryObject type.

1299 The following rules apply to versioning of all RegistryObjectType instances that are not instances of
1300 ExtrinsicObjectType type. It assumes that versioning is enabled for such RegistryObjectType types:

- 1301 ● A server MUST create a new version of a version-controlled, non-composed RegistryObjectType
1302 instance in the following cases:
 - 1303 ○ An existing object is replaced using the submitObjects protocol with mode of CreateOrVersion
 - 1304 ○ An existing object is updated using the updateObjects protocol with mode of CreateOrVersion
- 1305 ● A server MUST NOT create a new version of a composed RegistryObjectType instance when it is
1306 updated.
- 1307 ● When creating a new version for a non-composed RegistryObjectType instance, a server MUST
1308 create new logical objects for any composed logical objects within the new version of the
1309 composed object. Any such new logical object for composed objects MUST have a new server
1310 generated universally unique id and lid attribute.

1311 4.5.2 Versioning of ExtrinsicObjectType

1312 The ExtrinsicObjectType type requires special consideration for versioning because it may have an
1313 associated RepositoryItem which is versioned independently from the ExtrinsicObjectType instance.

1314 The following rules apply to versioning of ExtrinsicObjectType instances assuming that a server has
1315 versioning enabled for the ExtrinsicObjectType type:

- 1316 ● A server MUST create a new version of an existing ExtrinsicObjectType instance and assign it a
1317 new unique versionName within its VersionInfo element when either the ExtrinsicObjectType
1318 instance or its RepositoryItem are updated using the submitObjects or updateObjects protocol
1319 and the mode is CreateOrVersion
 - 1320 ○ A server MUST create a new version of an ExtrinsicObjectType instance and assign it a new
1321 unique versionName within its VersionInfo element when the previous version had a
1322 RepositoryItem and the new version does not have one (RepositoryItem was deleted).
 - 1323 ○ A server MUST create a new version of an ExtrinsicObjectType instance and assign it a new
1324 unique versionName within its VersionInfo element when the previous version did not have
1325 RepositoryItem and the new version has one (RepositoryItem was added). In such cases the
1326 server MUST also create a new version of the RepositoryItem and assign it a new unique
1327 versionName within the ContentVersionInfo element.
 - 1328 ○ A server MUST create a new version of the RepositoryItem for an existing
1329 ExtrinsicObjectType instance and assign it a new unique versionName within the
1330 ContentVersionInfo element when the RepositoryItem is updated using the submitObjects or
1331 updateObjects protocol and the mode is CreateOrVersion

1332 4.6 Versioning and References

1333 An object reference from a RegistryObjectType instance references a specific version of the referenced
1334 RegistryObjectType instance. When a server creates a new version of a referenced RegistryObjectType
1335 instance it MUST NOT move references from other objects from the previous version to the new version
1336 of the referenced object. Clients that wish to always reference the latest versions of an object MAY use
1337 the “dynamic reference” defined in eBRIM feature to always reference the latest version.

1338 A special case is when a SubmitObjectsRequest contains an object that is being versioned by the server
1339 and the request contains other objects that reference the object being versioned. In such case, the server
1340 MUST update all references within the submitted objects to the object being versioned such that those
1341 objects now reference the new version of the object being created by the request.

1342 4.7 Versioning of RegistryPackages

1343 When a server creates a new version of a RegistryPackageType instance, it MUST implicitly make all
1344 members of the old version also be members of the new version. This requires that the server MUST
1345 make a copy of all HasMember Associations in which the old version of the RegistryPackage is the
1346 sourceObject as follows:

- 1347 ● The copied Associations MUST be new versions of their original Association (MUST have the
1348 same lid)
- 1349 ● The sourceObject of the copied Associations MUST reference the new version of the
1350 RegistryPackage rather than the older version

1351

1353 4.8 Versioning and RegistryPackage Membership

1354 A RegistryPackage MUST NOT contain more than version of the same logical object as its member.

- 1355 ● A server MUST return an InvalidRequestException fault message if a client attempts to publish
1356 more than one version of the same logical object as member of the same RegistryPackage
1357 instance

1358

1359 4.9 Inter-version Association

1360 Each RegistryObject node in the version tree of a logical object except for the root version MUST be
1361 linked to the RegistryObject node in the version tree that was its immediate predecessor (previous
1362 version).

- 1363 ● A server MUST automatically link each new version in the version tree for a RegistryObject to its
1364 predecessor using an Association between the two versions
- 1365 ● The type attribute value of the Association MUST reference the canonical AssociationType
1366 "Supersedes"
- 1367 ● The sourceObject attribute value of the Association MUST reference the new version
- 1368 ● The targetObject attribute value of the Association MUST reference the old version

1369 Note that this section is functionally equivalent to the predecessor-set successor-set elements of the
1370 Version Properties as defined by [DeltaV].

1371 4.10 Version Removal

1372 Specific versions of a logical object MAY be deleted using the RemoveObjects protocol by specifying the
1373 version by its unique id.

- 1374 ● A server MAY allow authorized clients to remove specified versions of a RegistryObject
- 1375 ● A server MAY prune older versions of RegistryObjects based upon server specific administrative
1376 policies in order to manage storage resources
- 1377 ● When a non-leaf version within a version tree is deleted, a server MUST implicitly delete the entire
1378 version sub-tree under that non-leaf version such that no versions created directly or indirectly
1379 from the specified remain in the registry

1380 **4.11 Locking and Concurrent Modifications**

1381 This specification does not define explicit checkin and checkout capabilities as defined by [DeltaV]. A
1382 server MAY support such features in an implementation specific manner.

1383 This specification does not prescribe a locking model. An implementation may choose to support a locking
1384 model in an implementation specific manner. A future specification may address these capabilities.

1385 **4.12 Version Creation**

1386 The server manages creation of new version of a version-controlled resource automatically. A server that
1387 supports versioning MUST implicitly create a new version for the resource if an existing version of the
1388 resource is updated via a SubmitObjectsRequest or UpdateObjectsRequest when the mode attribute
1389 value is CreateOrVersion. A server MUST update the existing version of a resource without creating a
1390 new version when the mode attribute is set to CreateOrReplace.

1391

5 Validator Interface

1392 The Validator interface allows the validation of objects published to the server. The interface may be used
1393 by clients to validate objects already published to the server or may be used by the server to validate
1394 objects during the processing of the submitObjects or updateObjects protocol

1395 A server MUST implement the Validator interface as an endpoint. The Validator interface validates objects
1396 using [Validator Plugins](#) specific to the type of object being validated.

5.1 ValidateObjects Protocol

1398 The ValidateObjects protocol is initiated by sending an ValidateObjectsRequest message to the Validator
1399 endpoint.

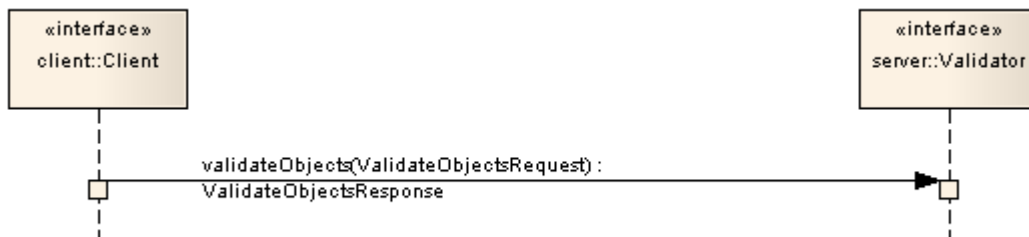


Illustration 6: ValidateObjects Protocol

1400 The Validator endpoint sends an ValidateObjectsResponse back as response. The
1401 ValidateObjectsResponse contains information on whether the objects were valid and if invalid objects
1402 were found it includes any validation errors that were encountered.

5.1.1 ValidateObjectsRequest

1404 The ValidateObjectsRequest message initiates the validateObjects protocol and specifies the objects that
1405 need to be validated.

5.1.1.1 Syntax

```

1407 <element name="ValidateObjectsRequest">
1408   <complexType>
1409     <complexContent>
1410       <extension base="rs:RegistryRequestType">
1411         <sequence>
1412           <element name="Query" type="rim:QueryType"
1413             minOccurs="0" maxOccurs="1" />
1414           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
1415           <element name="OriginalObjects" type="rim:RegistryObjectListType"
1416             minOccurs="1" maxOccurs="1"/>
1417           <element name="InvocationControlFile"
1418             type="rim:ExtrinsicObjectType"
1419             minOccurs="0" maxOccurs="unbounded"/>
1420         </sequence>
1421       </extension>
1422     </complexContent>
1423   </complexType>
1424 </element>

```


1425 5.1.1.2 Example

1426 The following example shows a client request to validate a specified WSDL file. It assumes that the server
1427 will be configured with a Validator plugin for WSDL files. It also assumes that the server will specify
1428 OriginalObjects and InvocationControlFile elements when it relays the request to the appropriate Validator
1429 plugin.

```
1430 <spi:ValidateObjectsRequest ...>  
1431   <rim:ObjectRefList>  
1432     <rim:ObjectRef id="urn:acme:wSDL:purchaseOrder.wsdl"/>  
1433   </rim:ObjectRefList>  
1434 </ValidateObjectsRequest>
```

1435 5.1.1.3 Description

- 1436 ● Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the validation
1437 process in a type specific manner. See [Canonical XML Validator plugin](#) for an example. This
1438 element MAY be specified by server when sending the request to the Validator plugin if the
1439 Validator plugin requires an invocation control file. It SHOULD NOT be specified by the client.
- 1440 ● Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances
1441 in the server. A server MUST validate all objects that are referenced by this element. This element
1442 is typically used when a client initiates the validateObjects protocol.
- 1443 ● Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST
1444 validate all objects that are contained in this element. This element is typically used when a server
1445 initiates the validateObjects protocol during the processing of a submitObjects or updateObjects
1446 protocol request or when it is delegating a client initiated validateObjects protocol request to a
1447 Validator plugin.
- 1448 ● Element Query - Specifies a query to be invoked. A server MUST validate all objects that match
1449 the specified query. This element is typically used when a client initiates the validateObjects
1450 protocol.

1451 5.1.1.4 Response

1452 This request returns [ValidateObjectsResponse](#) as response.

1453 5.1.1.5 Exceptions

1454 In addition to the [common exceptions](#), the following exceptions MAY be returned:

- 1455 ● ValidationException: signifies that an exception was encountered during the validateObjects operation

1456 5.1.2 ValidateObjectsResponse

1457 Currently ValidateObjectsResponse is a simple extension to [RegistryResponseType](#) and does not define
1458 additional attributes or elements.

1459 5.2 Validator Plugins

1460 Validator plugins allow a server to use specialized extension modules to validate specific types of objects
1461 during the processing of a SubmitObjectsRequest, UpdateObjectsRequest or a ValidateObjectsRequest.

1462 A specific instance of a Validator plugin is designed and configured to validate a specific type of object.
1463 For example, [the canonical XML Validator plugin](#) is designed and configured to validate XML Objects
1464 using Schematron documents as InvocationControlFile.

1465 **5.2.1 Validator Plugin Interface**

1466 A Validator plugin implements the [Validator interface](#). The server's Validator endpoint SHOULD delegate a
1467 validateObjects operation to any number of Validator plugins using the following algorithm:

- 1468 ● The server selects the RegistryObjects that are the target of the validateObjects operations using
1469 the <spi:Query> and <rim:ObjectRefList> elements. Any objects specified by the OriginalObjects
1470 element MUST be ignored by the server.
- 1471 ● The server partitions the set of target objects into multiple sets based upon the objectType
1472 attribute value for the target objects
- 1473 ● The server determines whether there is a Validator plugin configured for each objectType for
1474 which there is a set of target objects
- 1475 ● For each set of target objects that share a common objectType and for which there is a
1476 configured Validator plugin, the server MUST invoke the Validator plugin. The Validator plugin
1477 invocation MUST specify the target objects for that set using the OriginalObjects element. The
1478 server MUST NOT specify <spi:Query> and <rim:ObjectRefList> elements when invoking
1479 validateObjects operation on a Validator plugin
- 1480 ● Each Validator plugin MUST process the ValidateObjectsRequest and return a
1481 ValidateObjectsResponse or fault message to the server's Validator endpoint.
- 1482 ● The server's Validator endpoint MUST then combine the results of the individual
1483 ValidateObjectsRequest to Validator plugins into a single unified ValidateObjectsResponse and
1484 return it to the client.

1485 **5.2.2 Canonical XML Validator Plugin**

1486 The canonical XML Validator plugin is a validator plugin that validates XML content using a Schematron
1487 file as InvocationControlFile. The Schematron file specifies validation rules using [Schematron] language
1488 to validate XML content. The server may configure the canonical XML Validator plugin such that it is
1489 invoked with an appropriate schematron file as InvocationControlFile based upon the objectType of the
1490 object being validated.

1491

6 Cataloger Interface

1492 The Cataloger interface allows a client to catalog or index objects already in the server. The interface may
1493 be used by clients to catalog objects already published to the server or may be used by the server to
1494 catalog objects during the processing of the submitObjects or updateObjects protocol .

1495 A server MUST implement the Cataloger interface as an endpoint. The Cataloger interface catalogs
1496 objects using [Cataloger Plugins](#) specific to the type of object being cataloged.

6.1 CatalogObjects Protocol

1498 A client catalogs RegistryObjects residing in the server using the *CatalogObjects* protocol supported by
1499 the catalogObjects operation of the Cataloger interface.

1500 The CatalogObjects protocol is initiated by sending an CatalogObjectsRequest message to the Cataloger
1501 endpoint.

1502

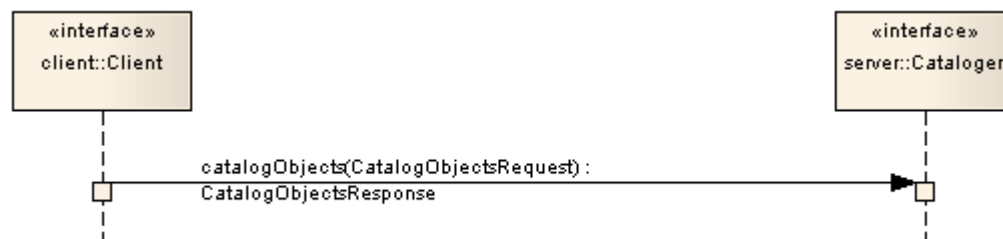


Illustration 7: CatalogObjects Protocol

1504 The Cataloger endpoint sends a CatalogObjectsResponse back to the client as response.

6.1.1 CatalogObjectsRequest

1506 The CatalogObjectsRequest message initiates the catalogObjects protocol and specifies the objects that
1507 need to be cataloged.

6.1.1.1 Syntax

```
1509 <element name="CatalogObjectsRequest">
1510   <complexType>
1511     <complexContent>
1512       <extension base="rs:RegistryRequestType">
1513         <sequence>
1514           <element name="Query" type="rim:QueryType"
1515             minOccurs="0" maxOccurs="1" />
1516           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
1517           <element name="OriginalObjects" type="rim:RegistryObjectListType"
1518             minOccurs="0" maxOccurs="1"/>
1519           <element name="InvocationControlFile"
1520             type="rim:ExtrinsicObjectType"
1521             minOccurs="0" maxOccurs="unbounded"/>
1522         </sequence>
1523       </extension>
1524     </complexContent>
```

```
1525 </complexType>
1526 </element>
```

1527 **6.1.1.2 Example**

1528 The following example shows a client request to catalog a specified WSDL file. It assumes that the server
1529 will be configured with a Cataloger plugin for WSDL files. It also assumes that the server will specify
1530 OriginalObjects and InvocationControlFile elements when it relays the request to the appropriate
1531 Cataloger plugin.

```
1532 <spi:CatalogObjectsRequest ...>
1533   <rim:ObjectRefList>
1534     <rim:ObjectRef id="urn:acme:wSDL:purchaseOrder.wSDL"/>
1535   </rim:ObjectRefList>
1536 </CatalogObjectsRequest>
```

1537 **6.1.1.3 Description**

- 1538 ● Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the cataloging
1539 process in a type specific manner. See [Canonical XML Cataloger plugin](#) for an example. This
1540 element MAY be specified by server when sending the request to the Cataloger plugin if the
1541 Cataloger plugin requires an an invocation control file. It SHOULD NOT be specified by the client.
- 1542 ● Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances
1543 in the server. A server MUST catalog all objects that are referenced by this element. This element
1544 is typically used when a client initiates the catalogObjects protocol.
- 1545 ● Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST
1546 catalog all objects that are contained in this element. This element is typically used when a server
1547 initiates the catalogObjects protocol during the processing of a submitObjects or updateObjects
1548 protocol request or when it is delegating a client initiated catalogObjects protocol request to a
1549 Cataloger plugin.
- 1550 ● Element Query - Specifies a query to be invoked. A server MUST catalog all objects that match
1551 the specified query. This element is typically used when a client initiates the catalogObjects
1552 protocol.

1553

1554 **6.1.1.4 Response**

1555 This request returns [CatalogObjectsResponse](#) as response.

1556 **6.1.1.5 Exceptions**

1557 In addition to [common exceptions](#), the following exceptions MAY be returned:

- 1558 ● CatalogingException: signifies that an exception was encountered during the catalogObjects operation

1559 **6.1.2 CatalogObjectsResponse**

1560 The CatalogObjectsResponse message is sent by the Cataloger endpoint in response to an
1561 CatalogObjectsRequest.

1562 6.1.2.1 Syntax

```
1563 <element name="CatalogObjectsResponse">
1564   <complexType>
1565     <complexContent>
1566       <extension base="rs:RegistryResponseType">
1567         </extension>
1568       </complexContent>
1569     </complexType>
1570   </element>
```

1571 6.1.2.2 Example

1572 The following example shows a CatalogObjectsResponse sent by a server to the client in response to a
1573 CatalogedObjectRequest. It shows that the Cataloger augmented the Original object with a new Slot that
1574 catalogs the target namespace used by the WSDL file.

1575

```
1576 <CatalogObjectsResponse status="urn:oasis:names:tc:ebxml-
1577   regrep:ResponseStatusType:Success">
1578   <rim:RegistryObjectList>
1579     <rim:RegistryObject xsi:type="rim:ExtrinsicObjectType"
1580       mimeType="text/xml"
1581       status="urn:oasis:names:tc:ebxml-regrep:StatusType:Submitted"
1582       objectType="urn:oasis:names:tc:ebxml-
1583   regrep:ObjectType:RegistryObject:ExtrinsicObject:XML:WSDL"
1584       lid="urn:acme:wSDL:purchaseOrder.wSDL"
1585       id="urn:acme:wSDL:purchaseOrder.wSDL">
1586       <rim:Slot
1587         name="urn:oasis:names:tc:ebxml-
1588   regrep:profile:wSDL:slot:targetNamespace">
1589         <rim:SlotValue xsi:type="rim:StringValueType">
1590           <rim:Value>urn:acme:Service:PurchaseOrder</rim:Value>
1591         </rim:SlotValue>
1592       </rim:Slot>
1593       <rim:RepositoryItem>...binary encoded content...</rim:RepositoryItem>
1594     </rim:RegistryObject>
1595   </rim:RegistryObjectList>
1596 </CatalogObjectsResponse>
```

1597 6.1.2.3 Description

1598 In addition to elements and attributes defined by [RegistryResponseType](#) the following are defined:

- 1599 ● Element RegistryObjectList (Inherited) – Contains the RegistryObjects that are produced as
1600 output of the catalogObjects operation. Typically this list contains the objects that were input to the
1601 catalogObjects operation, as well as new objects that were the output of the catalogObjects
1602 operation. The input objects MAY be modified by the cataloger as a result of the catalogObjects
1603 operation.
- 1604 ○ A cataloger MUST create AssociationType instance between the source object for the
1605 catalogObjects operation (specified by OriginalObjects element in CatalogRequest) and each
1606 of the cataloged RegistryObjectType instances generated by the cataloger. Each such
1607 AssociationType instance
 - 1608 ■ MUST have its type attribute reference the canonical AssociationType
1609 "urn:oasis:names:tc:ebxml-regrep:AssociationType:HasCatalogedMetadata"
 - 1610 ■ MUST have its sourceObject attribute reference the source object for the catalogObjects
1611 operation

- 1612 ■ MUST have its targetObject attribute reference a cataloged RegistryObjectType instance
1613 generated by the cataloger
- 1614 ○ A cataloger SHOULD assign the same accessControlPolicy to cataloged objects as their
1615 source object. A cataloger MAY use a different strategy for assigning access control policy to
1616 cataloged objects.
- 1617 ○ A server MUST delete all cataloged metadata generated by a cataloger when the source
1618 object is deleted.
- 1619 ○ A server MUST update all cataloged metadata generated by a cataloger when the source
1620 object is updated without creating a new version.

1621 **6.2 Cataloger Plugins**

1622 Cataloger plugins allow a server to use specialized extension modules to catalog specific types of objects
1623 during the processing of a SubmitObjectsRequest, UpdateObjectsRequest or a CatalogObjectsRequest.

1624 A specific instance of a Cataloger plugin is designed and configured to catalog a specific type of object.
1625 For example, [the canonical XML Cataloger plugin](#) is designed and configured to catalog XML Objects
1626 using XSLT documents as InvocationControlFile.

1627 **6.2.1 Cataloger Plugin Interface**

1628 A Cataloger plugin implements the [Cataloger interface](#). The server's Cataloger endpoint SHOULD
1629 delegate a catalogObjects operation to any number of Cataloger plugins using the following algorithm:

- 1630 ● The server selects the RegistryObjects that are the target of the catalogObjects operations using
1631 the <spi:Query> and <rim:ObjectRefList> elements. Any objects specified by the OriginalObjects
1632 element MUST be ignored by the server.
- 1633 ● The server partitions the set of target objects into multiple sets based upon the objectType
1634 attribute value for the target objects
- 1635 ● The server determines whether there is a Cataloger plugin configured for each objectType for
1636 which there is a set of target objects
- 1637 ● For each set of target objects that share a common objectType and for which there is a
1638 configured Cataloger plugin, the server MUST invoke the Cataloger plugin. The Cataloger plugin
1639 invocation MUST specify the target objects for that set using the OriginalObjects element. The
1640 server MUST NOT specify <spi:Query> and <rim:ObjectRefList> elements when invoking
1641 catalogObjects operation on a Cataloger plugin
- 1642 ● Each Cataloger plugin MUST process the CatalogObjectsRequest and return a
1643 CatalogObjectsResponse or fault message to the server's Cataloger endpoint.
- 1644 ● The server's Cataloger endpoint MUST then combine the results of the individual
1645 CatalogObjectsRequest to Cataloger plugins and commit these objects as part of the transaction
1646 associated with the request. It MUST then combine the individual CatalogObjectsResponse
1647 messages into a single unified CatalogObjectsResponse and return it to the client.

1648 **6.2.2 Canonical XML Cataloger Plugin**

1649 The canonical XML Cataloger plugin is a Cataloger plugin that catalogs XML content using an XSLT file
1650 as InvocationControlFile. The XSLT file specifies transformations rules using [XSLT] language to catalog
1651 XML content. The server may configure the canonical XML Cataloger plugin such that it is invoked with an
1652 appropriate XSLT file as InvocationControlFile based upon the objectType of the object being cataloged.

- 1653 An XSLT file used as InvocationControlFile with the Canonical XML Cataloger MUST meet the following
1654 constraints:
- 1655 ● Support an ExtrinsicObject as primary input
 - 1656 ● Support an XML RepositoryItem for the ExtrinsicObject object as a secondary input
 - 1657 ● The secondary input is specified using an <xsl:param> with name “repositoryItem” and with value
1658 that is the id of the ExtrinsicObject for which it is a RepositoryItem
- 1659 A server MUST implement the Canonical XML Cataloger with the following constraints:
- 1660 ● Uses an XSLT processor with the XSLT file specified as InvocationControlFile
 - 1661 ● Specifies the ExtrinsicObject being cataloged as the primary input to the XSLT processor
 - 1662 ● Specifies the RepositoryItem for the ExtrinsicObject object being cataloged by setting the
1663 parameter named “repositoryItem” with a value that is the id of the ExtrinsicObject for which it is a
1664 RepositoryItem
 - 1665 ● Resolves references to the RepositoryItem via the \$repositoryItem parameter value within the
1666 XSLT file specified as InvocationControlFile
- 1667

1668 7 Subscription and Notification

1669 A client MAY subscribe to events that transpire in the server by creating a Subscription. A server
1670 supporting Subscription and Notification feature MUST deliver a Notification to the subscriber when an
1671 event transpires that matches the event selection criteria specified by the client.

1672 7.1 Server Events

1673 Activities within the server result in events. [regrep-rim-v4.0] defines the AuditableEvent element,
1674 instances of which represent server events. A server creates AuditableEvent instances during the
1675 processing of client requests.

1676 7.1.1 Pruning of Events

1677 A server MAY periodically prune AuditableEvents in order to manage its resources. It is up to the server
1678 when such pruning occurs. A server SHOULD perform such pruning by removing the older
1679 AuditableEvents first.

1680 7.2 Notifications

1681 A Notification message is used by the server to notify clients of events they have subscribed to. A
1682 Notification contains the RegistryObjects, or references to the RegistryObjects, that are affected by the
1683 event for which the Notification is being sent, based upon the notificationOption within the DeliveryInfo for
1684 the subscription.

1685 Details for the Notification element are defined in [regrep-rim-v4.0].

1686 7.3 Creating a Subscription

1687 A client MAY create a subscription within a server if it wishes the server to send it a Notification when a
1688 specific type of event transpires. A client creates a subscription by submitting a rim:SubscriptionType
1689 instance to the server using the standard [SubmitObjects protocol](#).

1690 Details for the rim:SubscriptionType are defined in [regrep-rim-v4.0].

1691 7.3.1 Subscription Authorization

1692 A deployment MAY use custom Access Control Policies to decide which users are authorized to create a
1693 subscription and to what events. A server MUST return an AuthorizationException in the event that an
1694 unauthorized user submits a Subscription to a server.

1695 7.3.2 Subscription Quotas

1696 A server MAY use server specific policies to decide an upper limit on the number of Subscriptions a user
1697 is allowed to create. A server SHOULD return a QuotaExceededException in the event that an authorized
1698 user submits more Subscriptions than allowed by their server-specific quota.

1699 7.3.3 Subscription Expiration

1700 Each subscription MAY define a startTime and endTime attribute which determines the period within
1701 which a Subscription is valid. If startTime is unspecified then a server MUST set it to the time of

1702 submission of the subscription. If endTime is unspecified then the server MUST choose a default value
1703 based on its policies.

1704 Outside the bounds of the valid period, a Subscription MAY exist in an expired state within the server. A
1705 server MAY remove an expired Subscription at any time.

1706 A server MUST NOT deliver notifications for an event to an expired Subscriptions. An expired
1707 Subscription MAY be renewed by updating the startTime and / or endTime for the Subscription using the
1708 [UpdateObjects protocol](#).

1709 **7.3.4 Event Selection**

1710 A client MUST specify a Selector element within the Subscription to specify its criteria for selecting events
1711 of interest. The Selector element is of type rim:QueryType and specifies an parameterized query to be
1712 invoked with specified query parameters.

1713 A server MUST process AuditableEvents and determine which Subscriptions match the event using the
1714 algorithm illustrated by the following pseudo-code fragment:

1715

```
1716 //Get objects that match selector query  
1717 List<RegistryObjectType> objectsOfInterest =  
1718     getObjectMatchingSelectorQuery(selectorQuery);  
1719  
1720 if (objectsOfInterest.size() > 0) {  
1721  
1722     //Now get AuditableEvents that affected objectsOfInterest  
1723     //MUST not include AuditableEvents that have already been delivered  
1724     //to this subscriber  
1725     List<RegistryObjectType> eventsOfInterest =  
1726         getEventsOfInterest(objectsOfInterest);  
1727  
1728     if (eventsOfInterest.size() > 0) {  
1729         //Now create Notification on objectsOfInterest.  
1730         //Notification will include eventsOfInterest that only include objects  
1731         //that are affected by the event and are also in objectsOfInterest  
1732         NotificationType notification = createNotification(  
1733             objectsOfInterest, eventsOfInterest);  
1734  
1735         //Now send notification using info in DeliveryInfo  
1736         sendNotification(notification);  
1737     }  
1738 }
```

1739

- 1740 ● Objects of interest MUST be those objects that match the selector query for the subscription
- 1741 ● Events of interest MUST have affected at least one object of interest
- 1742 ● Events of interest MUST contain all objects of interest (or references to them) that were affected
1743 by the event
- 1744 ● Events of interest MUST NOT contain an object or reference to an object that is not an object of
1745 interest

1746 **7.4 Event Delivery**

1747 A client MAY specify zero or more DeliveryInfo elements within the Subscription to specify how the server
1748 should deliver events matching the subscription to the client. The DeliveryInfo element MUST include a

1749 NotifyTo element which specifies an EndPoint Reference (EPR) as defined by [WSA-CORE]. The
1750 NotifyTo element contains a <wsa:Address> element which contains a URI to the endpoint.

1751 Details for the DeliveryInfo element are defined in [regrep-rim-v4.0].

1752 **7.4.1 Notification Option**

1753 A client MAY specify a notificationOption attribute in DeliveryInfo element of a Subscription. The
1754 notificationOption attribute specifies how the client wishes to be notified of events. This attribute controls
1755 whether the Event within a Notification contains complete RegistryObjectType instances or only
1756 ObjectRefType instances. It is defined in detail in ebRIM.

1757 **7.4.2 Delivery to NotificationListener Web Service**

1758 If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-
1759 regrep:endPointType:soap”, then the server MUST use the specified address as the web service endpoint
1760 URL to deliver the Notification to. The target web service in this case MUST implement the
1761 NotificationListener interface.

1762 **7.4.3 Delivery to Email Address**

1763 If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-
1764 regrep:endPointType:rest”, then the server MUST use the specified address as the email address to
1765 deliver the Notification via email. This specification does not define how a server is configured to send
1766 Notifications via email.

1767 **7.4.4 Delivery to a NotificationListener Plugin**

1768 If the <wsa:Address> element has a rim:endpointType attribute value of “urn:oasis:names:tc:ebxml-
1769 regrep:endPointType:plugin”, then the server MUST use the specified address as a Notification plugin
1770 identifier and deliver the Notification via local call to the plugin. This specification does not define how a
1771 server is configured for Notification plugins.

1772 **7.4.4.1 Processing Email Notification Via XSLT**

1773 A client MAY specify an XSLT style sheet within a DeliveryInfo element to process a Notification prior to it
1774 being delivered to an email address. The XSLT style sheet MAY be specified using a Slot in DeliveryInfo
1775 element where the Slot's name is “urn:oasis:names:tc:ebxml-
1776 regrep:rim:DeliveryInfo:emailNotificationFormatter” and the Slots value is the id of an ExtrinsicObject
1777 whose repository item is the XSLT. The ExtrinsicObject and repository item MUST be submitted prior to or
1778 at the same time as the Subscription.

1779 **7.5 NotificationListener Interface**

1780 The NotificationListener interface allows a client to receive Notifications from the server for their
1781 Subscriptions. A client MUST implement the NotificationListener interface as an endpoint if they wish to
1782 receive Notifications via SOAP or REST. A server MUST implement a NotificationListener interface as an
1783 endpoint if it supports the object [replication feature](#) as this endpoint will be used by remote servers to
1784 deliver Notification of changes to replicated objects.

1785 7.6 Notification Protocol

1786 A server sends a Notification to an endpoint using the *Notification* protocol supported by the onNotification
1787 operation of the NotificationListener interface.

1788 A server initiates the Notification protocol by sending a Notification message to the NotificationListener
1789 endpoint registered within the Subscription for which the Notification is being delivered.



Illustration 8: Notification Protocol

1790 The onNotification operation does not send a response back to the server.

1791 7.6.1 Notification

1792 The Notification message is sent by the server to a NotificationListener interface implemented by the client
1793 and delivers an event notification for a subscription. It is a one-way request pattern and produces no
1794 response. The syntax and semantics of the Notification message is described in detail in ebRIM.

1795 7.7 Pulling Notification on Demand

1796 A client MAY “pull” Notifications for a Subscription by invoking the [GetNotification canonical query](#). A client
1797 MAY specify a startTime since which it wishes to include events within the pulled Notification. If client does
1798 not specify a startTime then all events since the last “push” delivery to that client's NotifyTo endpoint
1799 MUST be included in the Notification. If Subscription does not define any “push” delivery for that client's
1800 NotifyTo endpoint then a client MUST use startTime parameter to avoid getting the same events within the
1801 Notification returned by the GetNotification query.

1802 Pulling a Notification leaves the Notification intact on the server for any potential pushing of the Notification
1803 to endpoints defined in DeliveryInfo elements of the Subscription.

1804 7.8 Deleting a Subscription

1805 A client MAY terminate a Subscription with a server if it no longer wishes to be notified of events related to
1806 that Subscription. A client terminates a Subscription by deleting the corresponding Subscription object
1807 using the standard [RemoveObjects protocol](#).

1808 8 Multi-Server Features

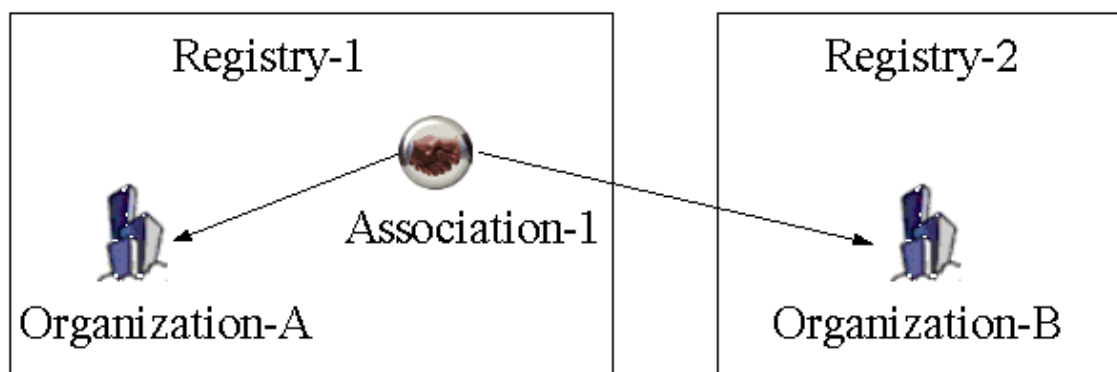
1809 This chapter describes features of ebXML RegRep that involve more than one ebXML RegRep server
1810 instances. These features include:

- 1811 ● Remote Object Reference – Allows references between objects residing in different servers
- 1812 ● Object Replication – Allows replication of objects residing in a remote server to a local server
- 1813 ● Federated Queries – Allows queries that execute against, and return results from multiple servers

1814 8.1 Remote Objects Reference

1815 A RegistryObject in one ebXML RegRep server MAY contain a reference to a RegistryObject in *any* other
1816 ebXML RegRep server that is compatible with ebXML RegRep specifications of a compatible version
1817 number as the source server. Remote object reference feature does not require the local and remote
1818 servers to be part of the same federation. Remote object references are described in detail in [regrep-rim-
1819 v4.0].

1820



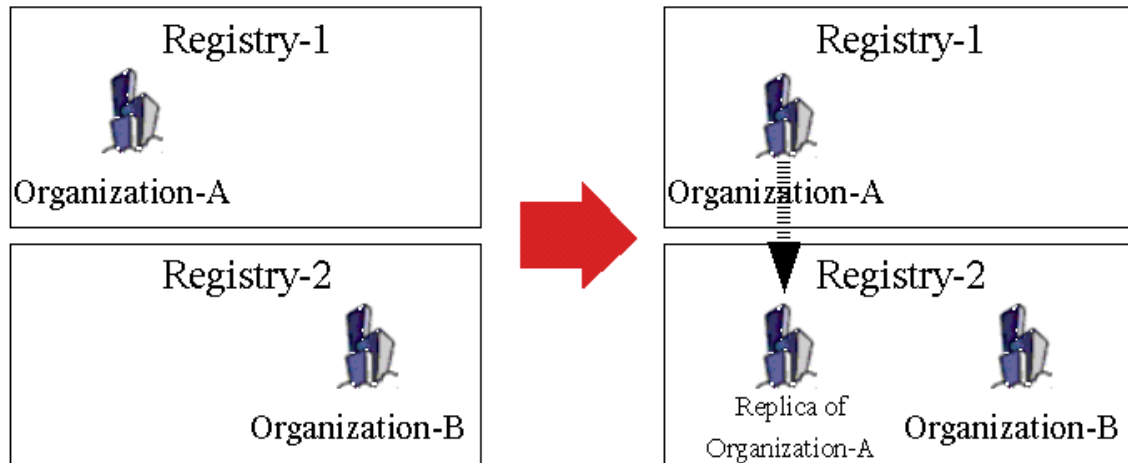
1821

Illustration 9: Remote Object Reference

1822 8.2 Local Replication of Remote Objects

1823 RegistryObjects within a server MAY be replicated in another server. A replicated copy of a remote object
1824 is referred to as its replica. The remote object MAY be an original object or it MAY be a replica. A replica
1825 from an original is referred to as a first-generation replica. A replica of a replica is referred to as a second-
1826 generation replica (and so on).

1827 A server that replicates a remote object locally is referred to as the local server for the replication. The
1828 server that contains the remote object being replicated is referred to as the remote server for the
1829 replication.



Before Replication

After Replication

Illustration 10: Local Replication of Remote Objects

1830

1831 The following rules govern replication of remote objects:

- 1832 ● A server MUST match local replicas of remote objects in the same manner as local objects within
1833 the Query protocol.
- 1834 ● A client MUST NOT perform update operations via SubmitObjects and UpdateObjects operations
1835 on a local replica of a remote object.
- 1836 ● A server MUST return an InvalidRequestException fault message if a client attempts to update a
1837 replica via SubmitObjects and UpdateObjects operations.
- 1838 ● A server MUST delete a replica if a client uses RemoveObjects operation to remove the replica.
- 1839 ● Objects MAY be replicated from any server to any other server without any requirement that the
1840 registries belong to the same federation.

1841 8.2.1 Creating Local Replica and Keeping it Synchronized

1842 Replication feature relies upon the Subscription and Notification feature to keep replicas synchronized with
1843 changes to the remote object. A local replica of a remote objects is created as follows:

- 1844 ● A client submits a Subscription to the remote server on behalf of the local server.
 - 1845 ○ The subscription is published like any other RegistryObjectType instance using the Submit
1846 Objects protocol with the LifecycleManager endpoint of the remote server.
 - 1847 ○ This typically requires that the client is registered with the remote server and can authenticate
1848 with it.
- 1849 ● The Subscription defines a Selector query that matches one or more objects that need to be
1850 replicated from remote server to local server.
 - 1851 ○ Selector query may match any number of objects using any selection criteria supported by the
1852 query.
- 1853 ● The Subscription specifies the address of a NotificationListener endpoint implemented by the local
1854 server where the remote server may send Notifications regarding the objects that need to be
1855 replicated.

- 1856 ● The local server uses the selector query for the subscription to PULL the initial copy of the remote
1857 object(s)
- 1858 ○ A server MUST NOT create a local replica for an object if a local object exists with the same
1859 id. In such case the server MUST return an ObjectExistsException fault message.
- 1860 ● Whenever the remote server send Notifications to the local server for the same Subscription, the
1861 local server synchronizes the local replica with the remote object.
- 1862 ○ A server MUST delete a local replica when its source object is deleted at the remote server.
- 1863 ○ A server MUST NOT delete a local object that is not a replica of a remote object if a
1864 notification arrives regarding the deletion of a remote object with the same id as the local
1865 object. In such case the server MUST return an InvalidRequestException fault message.
- 1866 A server MUST use standard QueryManager interface to read the state of a remote object. No prior
1867 registration or contract is needed for a server to read the state of a remote object if that object is readable
1868 by anyone, as is the case with the default access control policy.
- 1869 Once the state of the remote object has been read, a server MAY use server specific means to create a
1870 local replica of the remote object.
- 1871 A server MUST set a Slot with name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:home” on a
1872 local replica. The value of the Slot MUST be a StringValueType that specifies the base URL of the home
1873 server for the remote object that is the source of the local replica. A server MUST NOT set a Slot with
1874 name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:home” on a local object within its home
1875 server. The presence of this slot distinguished a local replica of a remote object from a local object.

1876 **8.2.2 Removing a Local Replica**

1877 An authorized client can remove a local replica in the same manner as removal of local objects using the
1878 standard [RemoveObjects protocol](#).

1879 **8.2.3 Removing Subscription With Remote Server**

1880 An authorized client can remove the Subscription at the remote server that was created on behalf of the
1881 local server using the standard [RemoveObjects protocol](#) with the remote server.

1882 **8.3 Registry Federations**

1883 A server federation is a set of ebXML RegRep servers that have voluntarily agreed to form a loosely
1884 coupled union. Such a federation may be based on common business interests or membership in a
1885 community-of-interest. Registry federations enabled clients to query the content of their member servers
1886 using federated queries as if they are a single logical server.

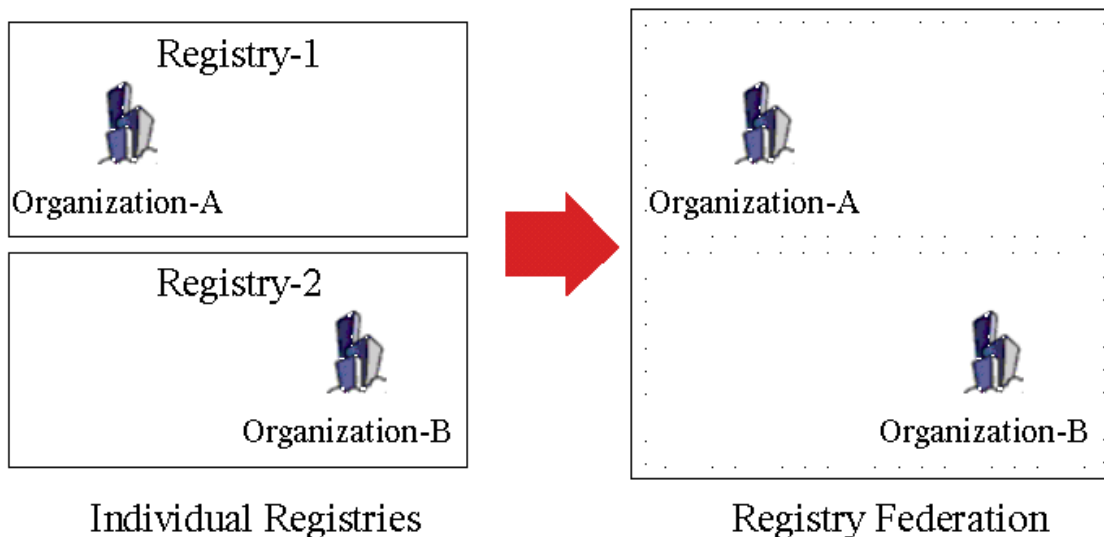


Illustration 11: Registry Federations

1887

1888 8.3.1 Federation Configuration

1889 A deployment MAY configure a set of related ebXML RegRep servers as a Federation using the Registry
 1890 and Federation classes defined in detail by [regrep-rim-v4.0]. Instances of these classes and the
 1891 associations between these instances describe a federation and its members.

1892 The Federation information model is described in [regrep-rim-v4.0].

1893 8.3.1.1 Creating a Federation

1894 The following rules govern how a federation is created:

- 1895 ● A Federation is created by submitting a Federation instance to a server using the [SubmitObjects](#)
 1896 [protocol](#)
- 1897 ● The server where the Federation is created is referred to as the federation home
- 1898 ● A federation home MAY contain multiple Federation instances

1899 8.3.1.2 Joining a Federation

1900 The following rules govern how a server joins a federation:

- 1901 ● Each server SHOULD have exactly one local RegistryType instance. Each server MAY have
 1902 multiple remote RegistryType instances
- 1903 ● A server MAY join an existing federation by submitting an instance of an Association that
 1904 associates the Federation instance as sourceObject, to the Registry instance representing the
 1905 server as targetObject, using a type of *HasFederationMember*. The home server for the
 1906 Association and the Federation objects MUST be the same
- 1907 ● A Federation (child federation) MAY join an existing federation (parent federation) by submitting
 1908 an instance of an Association that associates the Federation instance representing the parent
 1909 federation as sourceObject, to the Federation instance representing the child federation as
 1910 targetObject, using a type of *HasFederationMember*. The home server for the Association and the
 1911 parent Federation objects MUST be the same

1912 **8.3.1.3 Leaving a Federation**

1913 The following rules govern how a server leaves a federation:

- 1914 ● A server or a federation MAY leave a federation at any time by removing the
1915 *HasFederationMember* Association instance for its RegistryType or FederationType instance that
1916 links it with the parent FederationType instance. This is done using the standard [RemoveObjects](#)
1917 [protocol](#).

1918 **8.3.1.4 Dissolving a Federation**

1919 The following rules govern how a federation is dissolved:

- 1920 ● A federation is dissolved using the standard [RemoveObjects protocol](#) against the Federation's
1921 home server and removing its FederationType instance
- 1922 ● The removal of a FederationType instance is governed by Access Control Policies like any other
1923 RegistryObject

1924 **8.3.2 Local Vs. Federated Queries**

1925 A client MAY query a federation as a single unified logical server. A QueryRequest sent by a client to a
1926 federation member MAY be local or federated depending upon the value of the federated attribute of the
1927 QueryRequest.

1928 **8.3.2.1 Local Queries**

1929 When the federated attribute of QueryRequest has the value of *false* (default) then the query is a local
1930 query.

1931 A local QueryRequest is only processed by the server that receives the request.

1932 **8.3.2.2 Federated Queries**

1933 When the *federated* attribute of QueryRequest has the value of *true* then the query is a federated query.

1934 A server MUST route a federated query received by it to all servers that are represented by RegistryType
1935 instances in the membership tree of the federation(s) that is the target of the federated query on a best
1936 attempt basis.

1937 If an exception is encountered while dispatching a query to a federation member the server MUST return
1938 a QueryResponse as follows:

- 1939 ● The status of the QueryResponse MUST reference the canonical "PartialSuccess"
1940 ClassificationNode within the canonical ResponseStatusType ClassificationScheme
- 1941 ● The QueryResponse MUST have a set of Exception sub-elements of type
1942 rs:RegistryExceptionType, one for each exception encountered while dispatching a query to a
1943 remote server

1944 When a server routes a federated query to a federation member server then it MUST set the federated
1945 attribute value of the QueryRequest to *false* and the *federation* attribute value to null to avoid infinite loops.

1946 A federated query operates on data that is distributed across all the members of the target federation.

1947 When a client submits a federated query to a server and no federations exist in the server, then the server
1948 MUST treat it as a local query.

1949 The following rules apply to the treatment of iterative queries when the query is federated:

- 1950 ● A server MUST return a result set whose size is less than or equal to the maxResults parameter
1951 depending upon whether enough results are available within the scope of servers in the
1952 federation, starting at startIndex.
- 1953 ● A server MUST return the same result in a deterministic manner for the same federated
1954 QueryRequest if no changes have been made in between the request to the federation member
1955 servers and their collective state.
- 1956 ● A server MAY choose any implementation specific algorithm to select results from its federation
1957 members for each iteration of an iterative query as long as the algorithm is deterministic and
1958 repeatably produces the same results for the same set of federation members and their collective
1959 state. For example a server MAY use a sequential algorithm that gets as many results from each
1960 of its server sequentially until it satisfies the maxResults parameter or until there are no more
1961 results. Alternatively, a server MAY use a parallel algorithm that balances the amount of data
1962 retrieved from each of its federation members.

1963 **8.3.3 Local Replication of Federation Configuration**

1964 A federation member is required to locally cache the federation configuration metadata in the Federation
1965 home server for each federation that it is a member of. A server SHOULD use the replication feature for
1966 locally caching the Federation configuration.

1967 The federation member MUST keep the cached federation configuration synchronized with the original
1968 object in the Federation home.

1969 **8.3.4 Time Synchronization Between Federation Members**

1970 Federation members are not required to synchronize their system clocks with each other. However, each
1971 Federation member SHOULD keep its clock synchronized with an atomic clock server within the latency
1972 described by the replicationSyncLatency attribute of the Federation.

1973

9 Governance Features

1974 This chapter specifies how a server supports governance of RegistryObjects.

1975 Governance is defined as the enforcement of business processes and policies defined by a Community of
1976 Practice, that guide, direct, and control how its members collaborate to achieve its business goals.

1977 Within this specification, governance is defined as the enforcement of collaborative business processes
1978 and policies defined by a Community of Practice to manage the end-to-end life cycle of RegistryObjects
1979 within the server. Such collaborative business processes will be referred to as “governance
1980 collaborations”.

1981 The remainder of this chapter specifies:

- 1982 ● Scope of governance collaborations
- 1983 ● How governance collaborations are represented,
- 1984 ● How representations of governance collaborations are assigned to RegistryObjects, and
- 1985 ● How a server uses the representation of governance collaborations assigned to a RegistryObjects
1986 to govern them

1987

1988 9.1 Representing a Governance Collaboration

1989 This specification makes use of BPMN 2.0¹ [BPMN2] to represent business collaborations that govern
1990 RegistryObjects as follows:

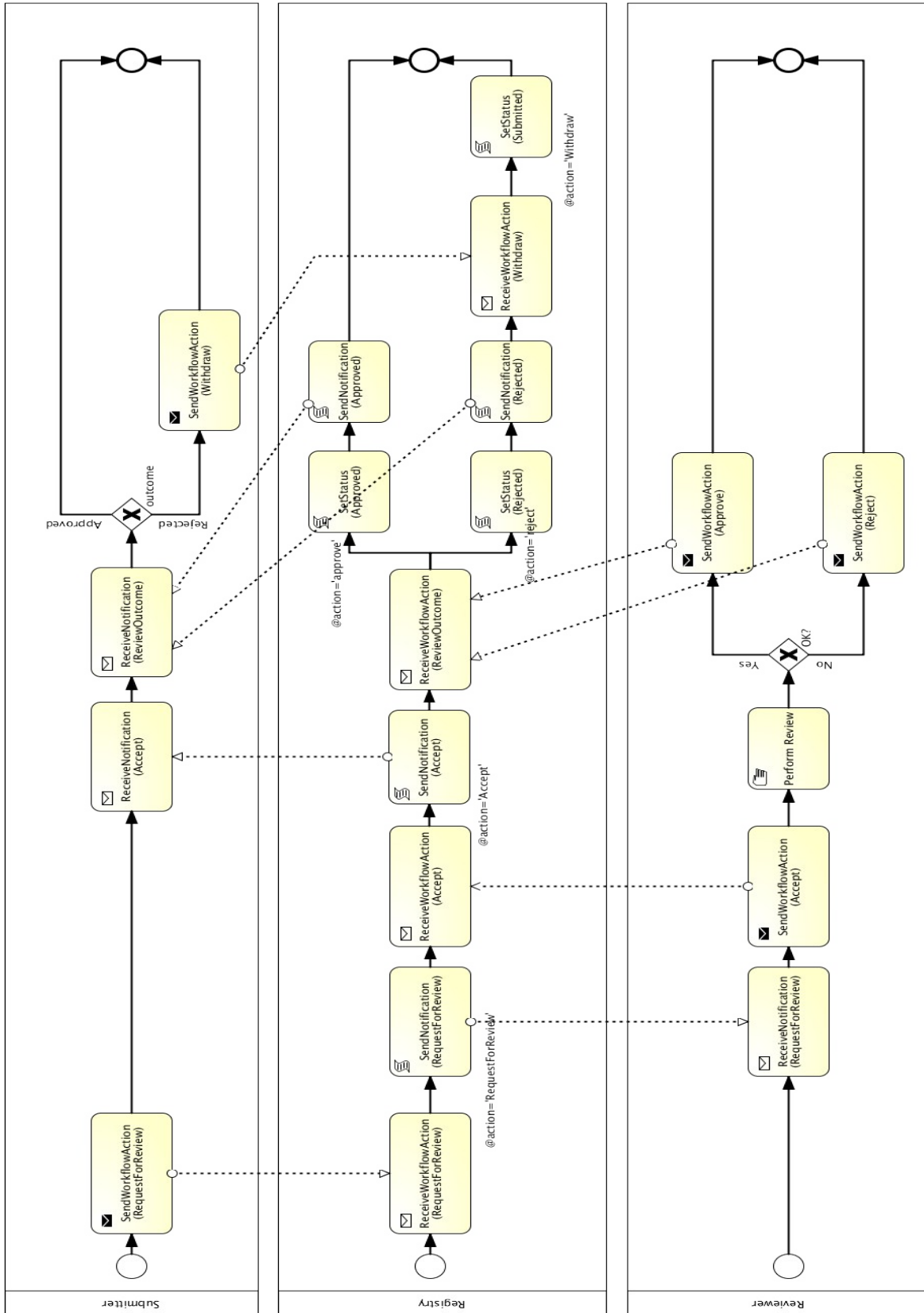
- 1991 ● Uses BPMN 2.0 diagram notation to pictorially represent business collaborations
- 1992 ● Uses BPMN 2.0 XML format to declaratively represent business collaborations in a machine
1993 processable syntax

1994 A governance collaboration consists of one or more participants where each participant's activities within
1995 the collaboration is described by a separate BPMN process and the interaction between the participants'
1996 processes is described by a single BPMN collaboration.

1997 Detailed specification of how to describe governance collaborations in BPMN 2.0 XML format and how a
1998 server executes them in a BPMN process engine are provided later in this chapter.

1999 Illustration 12 below provides an example of the Default Governance Collaboration represented by a
2000 BPMN 2.0 diagram notation. The [Default Governance Collaboration](#) is provided as a standard governance
2001 collaboration readily available for use in any server. It is described in detail later in this chapter.

¹At 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.



2003 **9.1.1 Content of Governance Collaboration BPMN Files**

2004 The collective content of the Governance Collaboration BPMN files, whether organized as a set of related
2005 modular files or a single monolithic file, MUST meet the following requirements:

- 2006 ● There MUST be exactly one collaboration element
- 2007 ● The collaboration element MUST have at least one participant element
- 2008 ● At least once participant element MUST have id value of “registryParticipant” and represents the
2009 RegRep server as a participant within the governance collaboration
- 2010 ● There MUST be a processRef element for the “registryParticipant”
- 2011 ● There MUST be a process element for each processRef attribute in each participant element
- 2012 ● The process element for other participants than the “registryParticipant” participant MAY conform
2013 to “Descriptive Conformance Sub-Class”¹ or “Analytic Conformance Sub-Class”² in [BPMN2] and
2014 need not be executed within a BPMN process engine
- 2015 ● The process element for the “registryParticipant” participant’s process MUST conform to
2016 “Common Executable Conformance Sub-Class”³ in [BPMN2] and MUST be executed by the
2017 server in a BPMN process engine
- 2018 ● The process elements SHOULD use tasks that conform to [canonical task patterns](#) defined later in
2019 this specification whenever possible

2020

2021 **9.2 Scope of Governance Collaborations**

2022 A governance collaboration may govern a single RegistryObject or it may govern a set of related
2023 RegistryObjects packaged together within a RegistryPackage as a single unit of governance. In either
2024 case, the target object of the governance collaboration is referred to as the governed object.

2025 **9.2.1 Packaging Related Objects as a Governance Unit**

2026 A client MUST publish a set of related RegistryObjects that are to be governed by the server as a single
2027 unit as follows:

- 2028 ● The objects MUST be immediate members of the same RegistryPackage
- 2029 ● The RegistryPackage MUST have a canonical slot with name “urn:oasis:names:tc:ebxml-
2030 regrep:rim:RegistryPackage:packageType”
- 2031 ● The value of the packageType slot MUST be a unique identifier for the type of package of which
2032 the group of related objects are an instance

2033 A server MUST treat RegistryPackages with a canonical slot with name “urn:oasis:names:tc:ebxml-
2034 regrep:rim:RegistryPackage:packageType” as the governed object.

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

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

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

2035 **9.3 Assigning a Governance Collaboration**

2036 A governance collaboration as represented by a BPMN2 XML file is not directly assigned to a
2037 RegistryObject. Instead it is assigned to a RegistryPackage and is implicitly applicable to RegistryObjects
2038 that are members of the RegistryPackage.

2039 Governance collaboration MAY be assigned to a specific RegistryPackage using a “GovernedBy”
2040 Association as follows:

- 2041 ● The type attribute value of Association MUST reference the canonical “GovernedBy”
2042 ClassificationNode within the canonical AssociationType ClassificationScheme whose id is
2043 “urn:oasis:names:tc:ebxml-regrep:AssociationType:GovernedBy”
- 2044 ● The targetObject attribute value of Association MUST reference an ExtrinsicObject with
2045 objectType “urn:oasis:names:tc:ebxml-
2046 regrep:ObjectType:RegistryObject:ExtrinsicObject:XML:BPMN2”
- 2047 ● The repository item for the ExtrinsicObject MUST be an XML document conforming to the BPMN2
2048 model XML Schema. If the modular approach to BPMN description is used then this file MUST be
2049 the collaboration BPMN file. The file MUST import or contain the BPMN process for the “Registry”
2050 participant
- 2051 ● The sourceObject attribute value of Association MUST reference the RegistryPackage instance to
2052 which the governance collaboration is being assigned
- 2053 ● The RegistryPackage MUST NOT have a canonical slot with name “urn:oasis:names:tc:ebxml-
2054 regrep:rim:RegistryPackage:packageType”

2055 **9.4 Determining Applicable Governance Collaboration**

2056 For any given RegistryObject, a server MUST use the following algorithm to determine the applicable
2057 governance collaboration (if any):

- 2058 1. Check if objects is an immediate member of a RegistryPackage that has a canonical slot with
2059 name “urn:oasis:names:tc:ebxml-regrep:rim:RegistryPackage:packageType”.
 - 2060 a) If it is so, then the object is not governed directly and instead its parent RegistryObjects is the
2061 governed object
 - 2062 b) Otherwise, proceed to next step
- 2063 2. Check if there is a governance collaboration assigned to a RegistryPackage ancestor using the
2064 canonical “HasGovernance” Association as follows:
 - 2065 a) Do a breadth-first traversal of the tree consisting of all RegistryPackage ancestors of the
2066 object and for each RegistryPackage see if it has a governance collaboration assigned to it
 - 2067 b) Stop when you find the first such governance collaboration
 - 2068 c) If a governance collaboration is found then use it as applicable governance collaboration
- 2069 3. If no RegistryPackage-specific governance collaboration is found then the object is not governed
2070 by any governance collaboration

2071

2072 **9.5 Determining the Registry Process in a Governance Collaboration**

2073 For any given governance collaboration, a server MUST use the following algorithm to determine the
2074 special Registry process:

- 2075 1. Find the participant element within the collaboration whose id is the canonical “registryParticipant”
- 2076 2. Find the processRef attribute of the “registryParticipant” and use the referenced process as the
2077 Registry process

2078 **9.6 Starting the Registry Process for a Governance Collaboration**

2079 The BPMN process for the “registryParticipant” within a governance collaboration is the only process in
2080 the collaboration that is required to be executed by the server within a BPMN process engine. This section
2081 specifies when and how a server starts this process.

2082 **9.6.1 Starting Registry Process By WorkflowAction**

2083 A server MAY start the Registry process for a governance collaboration in response to the publishing of a
2084 WorkflowAction object. This is specified in detail in [10.8.1.1 Server Processing of WorkflowAction](#).

2085 **9.7 Incoming messageFlows to Registry Process**

2086 Within a governance collaboration, a server MUST support incoming messageFlows to the Registry
2087 process from other processes in the collaboration that meet the following requirements:

- 2088 ● The sourceRef attribute of the messageFlow references a task that conforms to the
2089 [SendWorkflowAction task template](#) described later in this chapter
- 2090 ● The targetRef attribute of the messageFlow references a task that conforms to the
2091 [ReceiveWorkflowAction task template](#) described later in this chapter
- 2092 ● The messageRef attribute of the messageFlow is defined and references a message whose
2093 itemDefinition has attribute structureRef="rim:WorkflowActionType"

2094

2095 A server MAY support other types of incoming messages.

2096 **9.8 Outgoing messageFlows from Registry Process**

2097 A Registry process communicates with non-Registry processes by sending them notification messages.
2098 These messages may be an email message to an email endpoint for a person or a rim:NotificationType
2099 message to a service endpoint. Details are provided in the specification for the [SendNotification task](#)
2100 [pattern](#).

2101 A server MAY support other types of outgoing messages.

2102 **9.9 Canonical Task Patterns**

2103 This section specifies a set of canonical task patterns that may be used within participant processes in a
2104 governance collaboration. Some of these task patterns can only be used within the Registry process while
2105 some may only be used in the non-Registry processes of a governance collaboration.

2106 The following table provides a brief summary each of the canonical tasks defined by this specification.
 2107 Subsequent sections specify these tasks in more detail.

2108

| 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 |

2109

2110 9.9.1 SendWorkflowAction Task Pattern

2111 This canonical task pattern is used by a sendTask to represent the performing of a process-specific action
 2112 upon the governed object. This task pattern is the primary means for a non-Registry process to send a
 2113 message to the Registry process to trigger the Registry process forward.

2114 **Task Inputs:** The task has the following inputs as defined by dataInput elements in its ioSpecification:

- 2115 ● A dataInput that has an itemSubjectRef attribute that references an itemDefinition element whose
 2116 structureRef attribute value is "rim:WorkflowActionType"

2117 **Task Outputs:** The task has no outputs.

2118 **Task Actors:** This task SHOULD be performed by a role other than Registry role to indicate that some
 2119 external action (e.g. "approval") has been performed on the targetObject specified by the WorkflowAction.

2120 **Description:** To perform this task the actor submits a WorkflowAction to the server using the standard
 2121 SubmitObjects protocol. The name of the task SHOULD reflect the action being performed by the task
 2122 (e.g. name='SendWorkflowAction(RequestForReview)'). The WorkflowAction MUST specify:

- 2123 ● An action attribute identifying the action performed
- 2124 ● A targetObject attribute identifying the object that is the target of the action. Typically, this is the
 2125 governed object

2126 9.9.1.1 Server Processing of WorkflowAction

2127 Upon publishing of a WorkflowAction a server MUST process it as shown in the following pseudo-code
 2128 and explained further below:

2129


```

2130 WorkflowActionType workflowAction = ...;
2131 Collaboration collaboration =
2132     getApplicableGovernanceCollaboration(workflowAction.getTargetObject());
2133
2134 if (collaboration != null) {
2135     Process registryProcess = collaboration.getRegistryProcess();
2136     if (registryProcess != null) {
2137         if (!registryProcess.isActive()) {
2138             registryProcess.start();
2139         }
2140         registryProcess.deliverMessage(workflowAction);
2141     }
2142 }

```

2143

- 2144 1. Determine and get the applicable Governance Collaboration (as defined in [10.3 Determining](#)
2145 [Applicable Governance Collaboration](#))
- 2146 2. Determine and get the applicable Registry process for the collaboration (as defined in [10.4](#)
2147 [Determining the Registry Process in a Governance Collaboration](#))
- 2148 3. If the Registry process has not yet been started then start it within the BPMN process engine
- 2149 4. Deliver the WorkflowAction message to the Registry process where presumably a receiveTask
2150 based on the ReceiveWorkflowAction task pattern is waiting for it

2151

2152 9.9.2 ReceiveWorkflowAction Task Pattern

2153 This canonical task pattern is used by a receiveTask that waits for a process-specific action to be
2154 performed upon the governed object. This task pattern is the primary means for the Registry process to
2155 receive a message from a non-Registry process to trigger the Registry process forward.

2156 **Task Inputs:** The task has the following inputs as defined by dataInput elements in its ioSpecification:

- 2157 ● A dataInput that has an itemSubjectRef attribute that references an itemDefinition element whose
2158 structureRef attribute value is "rim:WorkflowActionType"

2159 **Task Outputs:** The task has no outputs.

2160 **Task Actors:** This task MUST be performed by the Registry role to wait until some external action (e.g.
2161 "approval") has been performed on the targetObject specified by the WorkflowAction.

2162 **Description:** This task waits until the server delivers a WorkflowAction message to the Registry process.
2163 The name of the task SHOULD reflect the action being performed (e.g.
2164 name='ReceiveWorkflowAction(RequestForReview)'. The task is typically followed by sequenceFlow
2165 elements that have a conditionExpression that predicate on the value of the action attribute of the
2166 WorkflowAction.

2167 9.9.3 SendNotification Task Pattern

2168 This canonical task pattern is used by a scriptTask to send a Notification message regarding the governed
2169 object to the roles and email addresses specified for the task. This task pattern is the primary means for
2170 the Registry process to send a message to a non-Registry process to trigger the non-Registry process
2171 forward.

2172 **Task Inputs:** None

2173 **Task Outputs:** None

2174 **Task Actors:** This task MUST be performed by the Registry role to keep governance roles for the
2175 governed object informed of important changes (e.g. status attribute changes) during the course of the life
2176 cycle of the governed object.

2177 **Description:** To perform this task the actor uses the sendNotification canonical [XPath extension](#)
2178 [function](#) defined later in this chapter. The name of the task SHOULD reflect the nature of the notification
2179 being sent by the task (e.g. name='SendNotification(Accept)').

2180 **9.9.4 ReceiveNotification Task Pattern**

2181 This canonical task pattern is used by a receiveTask that waits for a Notification message to be delivered.
2182 This task pattern is the primary means for a non-Registry process to receive a message from the Registry
2183 process to trigger the non-Registry process forward.

2184 **Task Inputs:** The task has the following inputs as defined by dataInput elements in its ioSpecification:

- 2185 ● A dataInput that has an itemSubjectRef attribute that references an itemDefinition element whose
2186 structureRef attribute value is "rim:NotificationType"

2187 **Task Outputs:**The task has no outputs.

2188 **Task Actors:** This task MUST be performed by a non-Registry role

2189 **Description:** This task waits until the server delivers a Notification message. The name of the task
2190 SHOULD reflect the nature of the notification being received by the task (e.g.
2191 name='ReceiveNotification(Accept)').

2192 **9.9.5 SetStatus Task**

2193 This canonical task pattern is used by a scripTask that updates the status of the specified object to a
2194 specified status value.

2195 **Task Inputs:** None

2196 **Task Outputs:** None

2197 **Task Actors:** This task MUST be performed by the Registry role to reflect changes in life cycle status
2198 during the course of the life cycle of the governed object.

2199 **Description:** To perform this task the actor uses the setStatus canonical [XPath extension function](#)
2200 defined later in this chapter. The name of the task SHOULD reflect the status being set by the task (e.g.
2201 name='SendStatus(Approved)').

2202 **9.9.6 Validate Task**

2203 This canonical task represents the validation of the governed object.

2204 **Task Inputs:** The task has no explicit inputs.

2205 **Task Outputs:**The task has no outputs.

2206 **Task Actors:** This task SHOULD be performed by the Registry role in response to the creation or
2207 updating of the governed object.

2208 **Description:** To perform this task the actor validates the governed object using the standard
2209 ValidateObjects protocol. The name of the task SHOULD be 'Validate' or an equivalent native language
2210 translation.

2211 9.9.7 Catalog Task

2212 This canonical task represents the cataloging of the governed object.

2213 **Task Inputs:** The task has no explicit inputs.

2214 **Task Outputs:** The task has no outputs.

2215 **Task Actors:** This task SHOULD be performed by the Registry role in response to the creation or
2216 updating of the governed object.

2217 **Description:** To perform this task the actor catalogs the governed object using the standard
2218 CatalogObjects protocol. The name of the task SHOULD be 'Catalog' or an equivalent native language
2219 translation.

2220 9.10 XPATH Extension Functions

2221 The following table specifies XPATH extension functions that MUST be supported by the BPMN process
2222 engine used by the server. The function signatures are described using the same conventions as used in
2223 section 1.4 of [\[XPATHFUNC\]](#).

2224 These functions MAY be used within XPATH expressions in a BPMN file wherever a **tExpression** type is
2225 supported by the BPMN schema.

- 2226 ● The namespace URI for these functions MUST be "urn:oasis:names:tc:ebxml-regrep:xsd:rs:4.0"
- 2227 ● The namespace prefix SHOULD be "rs"

2228

| 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. |

2229

2230 In addition to the functions described in table above, all [canonical query functions](#) supported by the server
2231 MUST also be supported by the server as XPATH functions.

2232 9.11 Default Governance Collaboration

2233 This section defines a canonical governance collaboration called the “Default Governance Collaboration”.
2234 The Default Governance Collaboration is defined by this specification to provide a standard governance
2235 process that can be supported by all implementations and may be assigned to specific RegistryPackages.

2236 The Default Governance Collaboration is represented by a canonical ExtrinsicObjectType instance with id
2237 “urn:oasis:names:tc:ebxml-regrep:collaboration:DefaultGovernanceCollaboration”.

2238 A BPMN diagram for the Default Governance Collaboration has been provided in Illustration 12 earlier.

2239 The Default Governance Collaboration is summarized as follows:

- 2240 ● The submitter requests review and approval of the governed object using SendWorkflowAction
2241 canonical task pattern with action “RequestForReview”
- 2242 ● The server receives the “RequestForReview” WorkflowAction and notifies the reviewer roles of
2243 the request for review using Notify canonical task pattern
- 2244 ● A reviewer accepts the request for review using SendWorkflowAction canonical task with
2245 WorkflowAction “Accept”
- 2246 ● The server notifies submitter roles that the governed object is under review using the using Notify
2247 canonical task
- 2248 ● The reviewer approves or rejects the governed objects using SendWorkflowAction canonical task
2249 and actions “Approve” or “Reject”
- 2250 ● The server notifies the submitter of the outcome of the review using the using Notify canonical
2251 task

2252

10 Security Features

2253
2254

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:

2255
2256
2257

- **[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>

2258
2259
2260

- **[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>

2261
2262
2263

- **[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>

2264
2265
2266

- **[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>

2267
2268
2269

- **[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>

2270

2271

10.1 Message Integrity

2272
2273

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](#)).

2274

10.1.1 Transport Layer Security

2275
2276

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.

2277

10.1.2 SOAP Message Security

2278
2279

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

2280

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

2281
2282

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:

2283
2284
2285
2286

- 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]

2287 **10.2 Message Confidentiality**

2288 A server SHOULD support encryption of protocol messages as defined by section 9 of [WSS-CORE] as a
2289 mechanism to support confidentiality of *all* ebXML RegRep protocols defined by this specification when
2290 those protocols are bound to SOAP.

2291 **10.3 User Registration and Identity Management**

2292 A server MUST provide a user registration mechanism to register and manage authorized users of the
2293 server. A server MUST also provide an identity management mechanism to register and manage the
2294 security tokens associated with registered users. This specification does not define how a server provides
2295 user registration and identity management mechanisms.

2296 **10.4 Authentication**

2297 A server MUST support authentication of the client requests based on the security tokens provided by the
2298 client and supported by the server. This specification does not specify the mechanism used by a server to
2299 authenticate client requests. Server implementations MAY use any means to provide authentication
2300 capability.

2301 **10.5 Authorization and Access Control**

2302 A server MUST control access by client to resources it manages based upon:

- 2303 ● The access control policy associated with each resource.
- 2304 ● The action the client is performing
- 2305 ● The identity associated with the client as well as any roles assigned to that identity

2306 A server MUST provide an access control and authorization mechanism based upon chapter titled
2307 “Access Control Information Model” in [regrep-rim-v4.0]. This model defines a default access control
2308 policy that MUST be supported by the server. In addition it also defines a binding to [XACML] that allows
2309 fine-grained access control policies to be defined.

2310 **10.6 Audit Trail**

2311 A server MUST keep a journal or audit trail of all operations that result in changing the state of its
2312 resources. This provides a basic form of non-repudiation where a client cannot repudiate that it performed
2313 actions that are logged in the Audit Trail.

2314 A server MUST create an audit trail for each request that affected the state of server resources. A server
2315 MUST create this audit trail using AuditableEventType instances as define by the chapter title “Event
2316 Information Model” of [regrep-rim-v4.0].

2317 Details of how a server maintains an Audit Trail of client requests is described in the chapter title “Event
2318 Information Model” of [regrep-rim-v4.0].

2319 11 Native Language Support (NLS)

2320 This chapter describes the Native Languages Support (NLS) features of ebXML RegRep.

2321 11.1 Terminology

2322 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) | <ul style="list-style-type: none">• 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]. |

2323

2324 11.2 NLS and Registry Protocol Messages

2325 For the accurate processing of data in both client and server, it is essential for the recipient of a protocol
2326 message to know the character set being used by it.

2327 A client SHOULD specify charset parameter in MIME header when they specify text/xml as Content-Type.

2328 The following is an example of specifying the character set in the MIME header.

2329

```
2330 Content-Type: text/xml; charset=ISO-2022-JP
```

2331

2332

2333 If a server receives a protocol message with the charset parameter omitted then it MUST use the default
2334 charset value of "us-ascii" as defined in [RFC 3023].

2335 Also, when an application/xml entity is used, the charset parameter is optional, and client and server
2336 MUST follow the requirements in Section 4.3.3 of [REC-XML] which directly address this contingency.

2337 If another Content-Type is used, then usage of charset MUST follow [RFC 3023].

2338 11.3 NLS Support in RegistryObjects

2339 The information model XML Schema [regrep-xsd-v4.0] defines the rim:InternationalStringType for defining
2340 elements that contains a locale sensitive string value.

2341

```
2342 <complexType name="InternationalStringType">
```

```
2343 <sequence>
2344 <element name="LocalizedString" type="tns:LocalizedStringType"
2345 minOccurs="0" maxOccurs="unbounded" />
2346 </sequence>
2347 </complexType>
```

2348

2349 An InternationalStringType may contain zero or more rim:LocalizedString elements within it where each
2350 LocalizedString contain a string value is a specified local language.

2351

```
2352 <complexType name="LocalizedStringType">
2353 <attribute ref="xml:lang" use="optional" default="en-US"/>
2354 <attribute name="value" type="tns:FreeFormText" use="required"/>
2355 </complexType>
```

2356

2357 Examples of such elements are the “Name” and “Description” elements of the RegistryObject class
2358 defined by [regrep-rim-v4.0].

2359 An element InternationalString is capable of supporting multiple locales within its collection of
2360 LocalizedStrings.

2361 The schema allows a single RegistryObject instance to include values for any NLS sensitive element in
2362 multiple locales.

2363 The following example illustrates how a single RegistryObject can contain NLS sensitive <rim:Name> and
2364 “<rim:Description>” elements with their value specified in multiple locales. Note that the <rim:Name> and
2365 <rim:Description> use the rim:InternationalStringType as their type.

```
2366 <rim:RegistryObject xsi:type="rim:ExtrinsicObjectType"...>
2367 <rim:Name>
2368 <rim:LocalizedString xml:lang="en-US" value="customACP1.xml"/>
2369 <rim:LocalizedString xml:lang="fi-FI" value="customACP1.xml"/>
2370 <rim:LocalizedString xml:lang="pt-BR" value="customACP1.xml"/>
2371 </rim:Name>
2372 <rim:Description>
2373 <rim:LocalizedString xml:lang="en-US" value="A sample custom ACP"/>
2374 <rim:LocalizedString xml:lang="fi-FI" value="Esimerkki custom ACP"/>
2375 <rim:LocalizedString xml:lang="pt-BR" value="Exemplo de ACP customizado"/>
2376 </rim:Description>
2377 </rim:RegistryObjectType>
```

2378

2379 Since locale information is specified at the sub-element level there is no language associated with a
2380 specific RegistryObject instance.

2381 **11.3.1 Language of a LocalizedString**

2382 The language MAY be specified in xml:lang attribute (Section 2.12 [REC-XML]).

2383 **11.3.2 Character Set of RegistryObject**

2384 The character set used by a RegistryObjects is defined by the charset attribute within the *Content-Type*
2385 mime header for the XML document containing the RegistryObject as shown below:

```
2386 Content-Type: text/xml; charset="UTF-8"
```

2388

2389

2390 Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of LocalizedStrings for
2391 maximum interoperability. A server MUST preserve the charset of a repository item as it is originally
2392 specified when it is submitted to the server.

2393 **11.4 NLS and Repository Items**

2394 While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is always
2395 associated with a single repository item. The repository item MAY be in a single locale or MAY be in
2396 multiple locales. This specification does not specify any NLS requirements for repository items.

2397 **11.4.1 Character Set of Repository Items**

2398 When a submitter submits a repository item, they MAY specify the character set used by the repository
2399 item using the MIME *Content-Type* mime header for the mime multipart containing the repository item as
2400 shown below:

2401

```
2402 Content-Type: text/xml; charset="UTF-8"
```

2403

2404 A server MUST preserve the charset of a repository item as it is originally specified when it is submitted to
2405 the server.

2406 **11.4.2 Language of Repository Items**

2407 This specification currently does not provide for a mechanism to specify the language of a RepositoryItem.

2408 This document currently specifies only the method of sending the information of character set and
2409 language, and how it is stored in a server. However, the language information MAY be used as one of the
2410 query criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation
2411 procedure, like client asking a preferred language for messages from server, could be functionality for a
2412 future revision of this document.

2413

12 REST Binding

2414 This chapter specifies a minimal REST binding for the QueryManager interface. This binding will be
2415 referred to as Core REST binding. Additional, more detailed REST bindings such as binding for ATOM,
2416 ATOM Pub, Open Search etc. will be defined by separate specifications. These additional specification will
2417 also provide a RESTful interface to the LifecycleManager interface.

2418

12.1 Canonical URL

2419 The canonical URL is an HTTP GET URL that MAY be used to reference or access RegistryObjectType
2420 instance in a RESTful manner. The canonical URL provides a simple universally supported means to
2421 access the object via HTTP GET. A server MUST provide access to its RegistryObjectType instances and
2422 repository items via canonical URLs as defined in sections below. Access to such resources MUST be
2423 controlled by the applicable access control policies associated with these resources as defined by ebRIM
2424 under the chapter titled Access Control Information Model.

2425

12.1.1 Canonical URL for RegistryObjects

2426 The canonical URL for RegistryObjectType has the following pattern:

```
2427 //The {id} parameter specifies the id of a RegistryObject  
2428 GET /rest/registryObjects/{id}
```

2429

2430 The following are examples of valid canonical URLs for RegistryObjectType instances. Note that for
2431 readability we do not encode special characters in the id attribute value.

2432

```
2433 //Get RegistryObject with id: urn:acme:pictures:danyal.jpg  
2434 GET  
2435 http://acme.com/myregistry/rest/registryObjects/urn:acme:pictures:danyal.jpg  
2436  
2437 //Get RegistryObject id: http://www.acme.com/pictures/danyal.jpg  
2438 GET  
2439 http://acme.com/myregistry/rest/registryObjects/http://www.acme.com/pictures/d  
2440 anyal.jpg
```

2441

2442

12.1.2 Canonical URL for Repository Items

2443 The canonical URL for repository items has the following pattern:

```
2444 //The {id} parameter specifies the id of a RegistryObject for repository item  
2445 GET /rest/repositoryItems/{id}
```

2446

2447 The following are examples of valid canonical URLs for RegistryObjectType instances. Note that for
2448 readability we do not encode special characters in the id attribute value.

2449

```
2450 //Get repository item associated with
```

```
2451 //ExtrinsicObject with id: urn:acme:pictures:danyal.jpg
2452 GET
2453 http://acme.com/myregistry/rest/repositoryItems/urn:acme:pictures:danyal.jpg
2454
2455 //Get repository item associated with
2456 //ExtrinsicObject with id: http://www.acme.com/pictures/danyal.jpg
2457 GET
2458 http://acme.com/myregistry/rest/repositoryItems/http://www.acme.com/pictures/d
2459 anyal.jpg
```

2460

2461 12.2 Query Protocol REST Binding

2462 A server MUST implement a REST Binding for the [Query Protocol](#) of the [Query Manager interface](#) as
2463 specified in this section. This binding allows a client to invoke any parameterized query supported by the
2464 server in a RESTful manner.

2465 The URL pattern or template for the parameterized query invocation is as follows:

2466

```
2467 #Template URL for parameterized query invocation
2468 <server base url>/rest/search?queryId={the query id}(&{<param-name>=<param-
2469 value>}) *
```

2471

2472 The following example shows the use of the FindObjectsByIdAndType canonical query using the REST
2473 binding.

```
2474 #Get RegistryObject with id: urn:acme:pictures:danyal.jpg
2475 GET http://acme.com/myregistry/rest/search?queryId=urn:oasis:names:tc:ebxml-
2476 regrep:query:FindObjectById&id=urn:acme:pictures:danyal.jpg
```

2477

2478 12.2.1 Parameter queryId

2479 The queryId parameter MUST specify the id of a parameterized stored query while zero or more additional
2480 parameters MAY provide parameter name and value pairs for parameters supported by the query. If the
2481 queryId is unspecified then it implicitly specifies the value "urn:oasis:names:tc:ebxml-
2482 regrep:query:FindObjectById" as the default queryId.

2483 12.2.2 Query Specific Parameters

2484 A parameterized query MAY define any number of query-specific parameters. A client MAY specify values
2485 for these parameters MAY as additional options to the URL. For example, the
2486 **id=urn:acme:pictures:danyal.jpg** part in example URL above supplies a value for the id query-specific
2487 parameter defined by the FindObjectsByIdAndType query.

2488 In addition to query-specific parameters, every query invocation URL MUST also support one or more
2489 canonical query parameters. These are described in subsequent sections.

2490 12.2.3 Canonical Query Parameter: depth

2491 This canonical query parameter represents the same named attribute and associated semantics as
2492 defined for [Query Request](#).

2493

```
2494 #Example: Find objects matching specifies keywords and also return  
2495 #related objects reachable by up to 10 levels of references  
2496 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2497 regrep:query:FindObjectByKeywords&keywords=automobile;japan&depth=10
```

2498 **12.2.4 Canonical Query Parameter: format**

2499 This canonical query parameter represents the same named attribute and associated semantics as
2500 defined for [Query Request](#).

2501

```
2502 #Example: Find 10 resources by keywords using en-us language and ebRS format  
2503 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2504 regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-  
2505 us&format=application/x-ebxml+xml
```

2506

2507 **12.2.5 Canonical Query Parameter: federated**

2508 This canonical query parameter represents the same named attribute and associated semantics as
2509 defined for [Query Request](#).

2510

```
2511 #Example: Perform a federated query across members of all configured  
2512 federations  
2513 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2514 regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true
```

2515

2516 **12.2.6 Canonical Query Parameter: federation**

2517 This canonical query parameter represents the same named attribute and associated semantics as
2518 defined for [Query Request](#).

2519

```
2520 #Example: Perform a federated query across members of specified federation  
2521 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2522 regrep:query:FindObjectByKeywords&keywords=automobile;japan&federated=true&fed-  
2523 eration=urn:acme:federation:acme-partners
```

2524

2525 **12.2.7 Canonical Query Parameter: matchOlderVersions**

2526 This canonical query parameter represents the same named attribute and associated semantics as
2527 defined for [Query Request](#).

2528

```
2529 #Example: Find objects matching specified name and include older versions of  
2530 matched objects if they match  
2531 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2532 regrep:query:BasicQuery&name=TestRegister1&matchOlderVersionsOnQuery=true
```

2533 12.2.8 Canonical Query Parameter: startIndex

2534 This canonical query parameter represents the same named attribute and associated semantics as
2535 defined for [Query Request](#).

2536

```
2537 #Example: Find 10 resources by keywords starting at index 30  
2538 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2539 regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10&star  
2540 tIndex=30
```

2541

2542 12.2.9 Canonical Query Parameter: lang

2543 This canonical query parameter represents the same named attribute and associated semantics as
2544 defined for [Query Request](#).

2545

```
2546 #Example: Find resources by keywords using en-us language  
2547 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2548 regrep:query:FindObjectByKeywords&keywords=automobile;japan&lang=en-us
```

2549

2550 12.2.10 Canonical Query Parameter: maxResults

2551 This canonical query parameter represents the same named attribute and associated semantics as
2552 defined for [Query Request](#).

2553

```
2554 #Example: Find 10 resources by keywords  
2555 /rest/search/?queryId=urn:oasis:names:tc:ebxml-  
2556 regrep:query:FindObjectByKeywords&keywords=automobile;japan&maxResults=10
```

2557 12.2.11 Use of Functions in Query Parameters

2558 Query functions may be used in query parameters as defined in [Query Function](#). The only caveat is that
2559 the special characters such as the special sequences “#@” and “@#”, special characters “(, ”) etc.
2560 MUST be specified in their URL encoded representation as defined by RFC 3986 and RFC 3629.

2561 For example a query parameter “#@’@#rs:currentTime#@’@#” would evaluate to the current time as a
2562 quoted timestamp string in ISO 8601 format such as “#@’@#2010-08-05T17:14:18.866#@’@#”. Such a
2563 query parameter in REST interface would have to be URL encoded to be as shown in the following
2564 example:

```
2565 http://localhost:8080/omar-server/rest/search?  
2566 queryId=urn:ogc:specification:regrep:profile:ISO19139:query:DatasetDiscoveryQu  
2567 ery&title=%23%40%2740%23ebers:currentTime%28%29%23%40%2740%23
```

2568 12.2.12 Query Response

2569 The response document returned by the Query Protocol REST binding MUST be a [QueryResponse](#)
2570 document. If the format parameter value is unspecified or if it is specified as “application/x-ebxml+xml” then
2571 the response document must have query:QueryResponse element as its root element.

2573

13 SOAP Binding

2574 This chapter specifies the requirements for SOAP Binding that a regrep server or client must adhere to.
2575 The normative definition of service endpoint, protocols and their SOAP binding is contained within the
2576 WSDL 1.1 definitions defined by [regrep-wsdl-v4.0]. A WSDL 2.0 definition is also available in [regrep-
2577 wsdl-v4.0].

2578 The following additional requirements are defined by this specification for the SOAP binding:

- 2579 ● A server MUST use WS-Addressing SOAP Headers when sending a Notification message to a
2580 SOAP endpoint as defined [here](#).

2581 13.1 WS-Addressing SOAP Headers

2582 The following rules apply to a server when sending a Notification message to a SOAP endpoint for the
2583 NotificationListener.

- 2584 ● Use of WS-Addressing SOAP headers MUST conform to [WSA-SOAP].
- 2585 ● A server MUST set the content of the wsa:MessageID element to a unique id. A server SHOULD
2586 generate a universally unique id value that conform to the format of a URN that specifies a DCE
2587 128 bit UUID as specified in [UUID] (e.g. *urn:uuid:a2345678-1234-1234-123456789012*).
- 2588 ● A server MUST set the wsa:ReplyTo SOAP header element
 - 2589 ○ The wsa:Address elements content MUST be set to the base URL for the server.
- 2590 ● A server MUST set the content of the wsa:To element to the SOAP endpoint URL where the
2591 message is being sent to.
- 2592 ● A server MUST set the content of the wsa:Action element to the value of the soapAction attribute
2593 of the soap:operation element for the operation defined for the SOAP binding for the interface's
2594 WSDL.

2595 The following example shows a SOAP message containing a Notification intended for a
2596 NotificationListener SOAP endpoint.

2597

```
2598 <env:Envelope>  
2599   <env:Header>  
2600     <wsa:MessageID>  
2601       urn:uuid:3e79348f-d696-4fac-a015-a4bae0bf83c5  
2602     </wsa:MessageID>  
2603     <wsa:ReplyTo>  
2604       <wsa:Address>http://www.acme.com/regrep</wsa:Address>  
2605     </wsa:ReplyTo>  
2606     <wsa:To>http://www.client.com/notificationListener</wsa:To>  
2607     <wsa:Action>urn:oasis:names:tc:ebxml-  
2608 regrep:wsdl:NotificationListener:bindings:4.0:NotificationListener:onNotificat  
2609 ion</wsa:Action>  
2610   </env:Header>  
2611   <env:Body>  
2612     <rim:Notification .../>  
2613   </env:Body>  
2614 </env:Envelope>
```

2615

Appendix A. Protocol Exceptions

2616 This appendix defines the standard exception that may be returned by various protocols defined in this
 2617 specification. These exceptions MUST be returned as SOAP fault messages in the SOAP binding for the
 2618 protocols. Implementations SHOULD provide relevant details regarding the exception within the Detail
 2619 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. |

2620

2621