1

OASIS/ebXML Registry Services Specification v2.5

–Committee Approved Specification

OASIS/ebXML Registry Technical Committee

June 2003

2 ***This page intentionally left blank.***

1 Status of this Document

This document is an OASIS ebXML Registry Technical Committee Approved Specification - June 2003.

Distribution of this document is unlimited.

The document formatting is based on the Internet Society's Standard RFC format.

This version:

http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebrs-2.5.pdf

***Latest Technical Committee Approved version*:**

http://www.oasis-open.org/committees/regrep/documents/2.1/specs/ebrs.pdf

***Latest OASIS Approved Standard*:**

http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf

2 OASIS/ebXML Registry Technical Committee

This is an OASIS/ebXML Registry Technical Committee draft document. The following persons are members of the OASIS/ebXML Registry Technical Committee:

Kathryn Breininger, Boeing
Joseph M. Chiusano, Booz Allen Hamilton
Suresh Damodaran, Sterling Commerce
Sally Fuger, Individual Member
Peter Kacandes, Adobe Systems Incorporated
Michael Kass, NIST
Paul Macias, LMI
Matthew MacKenzie, Individual Member
Monica Martin, Sun Microsystems
Farrukh Najmi, Sun Microsystems
Sanjay Patil, IONA
Nikola Stojanovic, Individual Member
Uday Subarayan, Sun Microsystems

Contributors

The following persons contributed to the content of this document, but were not a voting member of the OASIS/ebXML Registry Technical Committee.

John Bekisz, Software AG, Inc.
Lisa Carnahan, NIST
Anne Fischer, Individual
Len Gallagher, NIST
Duane Nickull, XML Global
John Silva, Philips Medical
Sekhar Vajjhala, Sun Microsystems

Table of Contents

1 Status of this Document 3
2 OASIS/ebXML Registry Technical Committee 4
Table of Contents 5
Table of Figures 11
Table of Tables 13
3 Introduction 14
3.1 Summary of Contents of Document 14
3.2 General Conventions 14
3.3 Audience 14
4 Design Objectives 15
4.1 Goals 15
4.2 Caveats and Assumptions 15
5 System Overview 16
5.1 What The ebXML Registry Does 16
5.2 How The ebXML Registry Works 16
5.2.1 Schema Documents Are Submitted 16
5.2.2 Business Process Documents Are Submitted 16
5.2.3 Seller's Collaboration Protocol Profile Is Submitted 16
5.2.4 Buyer Discovers The Seller 16
5.2.5 CPA Is Established 17
5.3 Registry Users 17
5.4 Where the Registry Services May Be Implemented 18
5.5 Implementation Conformance 18
5.5.1 Conformance as an ebXML Registry 18
5.5.2 Conformance as an ebXML Registry Client 19
6 ebXML Registry Architecture 20
6.1 Registry Service Described 20
6.2 Abstract Registry Service 21
6.2.1 LifeCycleManager Interface 21
6.2.2 QueryManager Interface 22
6.3 Concrete Registry Services 22
6.4 SOAP Binding 23
6.4.1 WSDL Terminology Primer 23
6.4.2 Concrete Binding for SOAP 24
6.5 ebXML Message Service Binding 24
6.5.1 Service and Action Elements 24
6.5.2 Synchronous and Asynchronous Responses 24
6.5.3 ebXML Registry Collaboration Profiles and Agreements 25
6.6 HTTP Binding 26
6.6.1 Standard URI Parameters 26
6.6.2 QueryManager HTTP Interface 26
6.6.3 LifeCycleManager HTTP Interface 28
6.6.4 Security Considerations 30
6.6.5 Exception Handling 30
6.7 Registry Clients 30

6.7.1 Registry Client Described 30
6.7.2 Registry Communication Bootstrapping 31
6.7.3 RegistryClient Interface 32
6.7.4 Registry Response 32
6.8 Interoperability Requirements 33
6.8.1 Client Interoperability 33
6.9 Registry Requests and Responses 33
6.9.1 RegistryRequestType 33
6.9.2 RegistryResponseType 34
6.9.3 RegistryResponse 35
6.9.4 RegistryErrorList 35
6.9.5 RegistryError 36
6.9.6 ErrorType 36

7 Lifecycle Management Service 38

7.1 Lifecycle of a RegistryObject 38
7.2 RegistryObject Attributes 38
7.3 The Submit Objects Protocol 39
7.3.1 SubmitObjectsRequest 39
7.3.2 RegistryResponse 40
7.3.3 Universally Unique ID Generation 40
7.3.4 ID Attribute And Object References 41
7.3.5 Audit Trail 41
7.3.6 Sample SubmitObjectsRequest 41
7.4 The Update Objects Protocol 45
7.4.1 UpdateObjectsRequest 45
7.4.2 Audit Trail 46
7.5 The Add Slots Protocol 46
7.5.1 AddSlotsRequest 47
7.6 The Remove Slots Protocol 48
7.6.1 RemoveSlotsRequest 48
7.7 The Approve Objects Protocol 49
7.7.1 ApproveObjectsRequest 50
7.7.2 Audit Trail 51
7.8 The Deprecate Objects Protocol 51
7.8.1 DeprecateObjectsRequest 51
7.8.2 Audit Trail 52
7.9 The Undeprecate Objects Protocol 52
7.9.1 UndeprecateObjectsRequest 53
7.9.2 Audit Trail 54
7.10 The Remove Objects Protocol 54
7.10.1 RemoveObjectsRequest 54

8 Query Management Service 57

8.1 Ad Hoc Query Request/Response 57
8.1.1 AdhocQueryRequest 58
8.1.2 AdhocQueryResponse 59
8.1.3 ReponseOption 59
8.1.4 Iterative Query Support 61
8.2 Filter Query Support 61

8.2.1 FilterQuery 63
8.2.2 RegistryObjectQuery 65
8.2.3 RegistryEntryQuery 77
8.2.4 AssociationQuery 80
8.2.5 AuditableEventQuery 83
8.2.6 ClassificationQuery 86
8.2.7 ClassificationNodeQuery 87
8.2.8 ClassificationSchemeQuery 92
8.2.9 RegistryPackageQuery 94
8.2.10 ExtrinsicObjectQuery 96
8.2.11 OrganizationQuery 98
8.2.12 ServiceQuery 101
8.2.13 FederationQuery 104
8.2.14 RegistryQuery 105
8.2.15 SubscriptionQuery 106
8.2.16 UserQuery 107
8.2.17 Registry Filters 109
8.2.18 XML Clause Constraint Representation 113
8.3 SQL Query Support 118
8.3.1 SQL Query Syntax Binding To [ebRIM] 118
8.3.2 Semantic Constraints On Query Syntax 120
8.3.3 SQL Query Results 120
8.3.4 Simple Metadata Based Queries 120
8.3.5 RegistryObject Queries 121
8.3.6 RegistryEntry Queries 121
8.3.7 Classification Queries 121
8.3.8 Association Queries 122
8.3.9 Package Queries 123
8.3.10 ExternalLink Queries 123
8.3.11 Audit Trail Queries 124
8.3.12 Object Export Queries 124
8.4 Content Retrieval 124
8.4.1 GetContentRequest 125
8.4.2 GetContentResponse 126
8.4.3 Identification Of Content Payloads 126
8.4.4 GetContentResponse Message Structure 126
9 Content Management Services 129
9.1 Content Validation 129
9.1.1 Content Validation: Use Cases 129
9.2 Content Cataloging 130
9.2.1 Content-based Discovery: Use Cases 130
9.3 Abstract Content Management Service 131
9.3.1 Inline Invocation Model 131
9.3.2 De-coupled Invocation Model 132
9.4 Content Management Service Protocol 133
9.4.1 ContentManagementServiceRequestType 133
9.4.2 ContentManagementServiceResponseType 135
9.5 Publishing / Configuration of a Content Management Service 135

9.5.1 Multiple Content Management Services and Invocation Control Files137
9.6 Invocation of a Content Management Service 137
9.6.1 Resolution Algorithm For Service and Invocation Control File 137
9.6.2 Audit Trail and Cataloged Content 138
9.6.3 Referential Integrity 138
9.6.4 Error Handling 138
9.7 Validate Content Protocol 138
9.7.1 ValidateContentRequest 139
9.7.2 ValidateContentResponse 140
9.8 Catalog Content Protocol 141
9.8.1 CatalogContentRequest 141
9.8.2 CatalogContentResponse 142
9.9 Illustrative Example: Default XML Cataloging Service 143
9.10 Default XML Content Cataloging Service 144
9.10.1 Publishing of Default XML Content Cataloging Service 145
10 Event Notification Service 146
10.1 Use Cases 146
10.1.1 CPP Has Changed 146
10.1.2 New Service is Offered 146
10.1.3 Monitor Download of Content 146
10.1.4 Monitor Price Changes 146
10.1.5 Keep Replicas Consistent With Source Object 146
10.2 Registry Events 147
10.3 Subscribing to Events 147
10.3.1 Event Selection 148
10.3.2 Notification Action 148
10.3.3 Subscription Authorization 148
10.3.4 Subscription Quotas 148
10.3.5 Subscription Expiration 148
10.3.6 Subscription Rejection 149
10.4 Unsubscribing from Events 149
10.5 Notification of Events 149
10.6 Retrieval of Events 150
10.6.1 GetNotificationsRequest 150
10.6.2 NotificationType 150
10.6.3 ObjectRefsNotification 151
10.6.4 ObjectsNotification 151
10.7 Purging of Events 152
11 Cooperating Registries Support 153
11.1 Cooperating Registries Use Cases 153
11.1.1 Inter-registry Object References 153
11.1.2 Federated Queries 153
11.1.3 Local Caching of Data from Another Registry 153
11.1.4 Object Relocation 154
11.2 Registry Federations 154
11.2.1 Federation Metadata 154
11.2.2 Local Vs. Federated Queries 155
11.2.3 Federated Lifecycle Management Operations 156

11.2.4 Federations and Local Caching of Remote Data 156
11.2.5 Caching of Federation Metadata 156
11.2.6 Time Synchronization Between Registry Peers 156
11.2.7 Federations and Security 157
11.2.8 Federation Lifecycle Management Protocols 157
11.3 Object Replication 158
11.3.1 Use Cases for Object Replication 158
11.3.2 Queries And Replicas 159
11.3.3 Lifecycle Operations And Replicas 159
11.3.4 Object Replication and Federated Registries 159
11.3.5 Creating a Local Replica 159
11.3.6 Transactional Replication 159
11.3.7 Keeping Replicas Current 160
11.3.8 Write Operations on Local Replica 160
11.3.9 Tracking Location of a Replica 160
11.3.10 Remote Object References to a Replica 160
11.3.11 Removing a Local Replica 160
11.4 Object Relocation Protocol 160
11.4.1 RelocateObjectsRequest 163
11.4.2 AcceptObjectsRequest 164
11.4.3 Object Relocation and Remote ObjectRefs 164
11.4.4 Notification of Object Relocation To ownerAtDestination 165
11.4.5 Notification of Object Commit To sourceRegistry 165
11.4.6 Object Relocation and Timeouts 166
12 Registry Security 167
12.1 Security Concerns 167
12.2 Integrity of Registry Content 167
12.2.1 Message Payload Signature 167
12.2.2 Payload Signature Requirements 168
12.3 Authentication 169
12.3.1 Message Header Signature 170
12.4 Key Distribution and KeyInfo Element 171
12.5 Confidentiality 172
12.5.1 On-the-wire Message Confidentiality 172
12.5.2 Confidentiality of Registry Content 172
12.6 Access Control and Authorization 172
12.6.1 Actors / Role Mapping 172
Appendix A Web Service Architecture 174
A.1 Registry Service Abstract Specification 174
A.2 Registry Service SOAP Binding 174
Appendix B ebXML Registry Schema Definitions 175
B.1 RIM Schema 175
B.2 Registry Services Interface Base Schema 175
B.3 QueryManager Service Schema 175
B.4 LifecycleManager Service Schema 175
B.5 Content Management Service Schema 175
B.6 Examples of Instance Documents 175

Appendix C Interpretation of UML Diagrams 176
C.1 UML Class Diagram 176
C.2 UML Sequence Diagram 176
Appendix D SQL Query 177
D.1 SQL Query Syntax Specification 177
D.2 Non-Normative BNF for Query Syntax Grammar 177
D.3 Relational Schema For SQL Queries 178
Appendix E Security Implementation Guideline 179
E.1 Security Concerns 179
E.2 Authentication 180
E.3 Authorization 180
E.4 Registry Bootstrap 180
E.5 Content Submission – Client Responsibility 180
E.6 Content Submission – Registry Responsibility 180
E.7 Content Remove/Deprecate – Client Responsibility 181
E.8 Content Remove/Deprecate – Registry Responsibility 181
E.9 Using ds:KeyInfo Field 181
Appendix F Native Language Support (NLS) 183
F.1 Definitions 183
F.1.1 Coded Character Set (CCS): 183
F.1.2 Character Encoding Scheme (CES): 183
F.1.3 Character Set (charset): 183
F.2 NLS And Request / Response Messages 183
F.3 NLS And Storing of RegistryObject 183
F.3.1 Character Set of *LocalizedString* 184
F.3.2 Language Information of *LocalizedString* 184
F.4 NLS And Storing of Repository Items 184
F.4.1 Character Set of Repository Items 184
F.4.2 Language information of repository item 184
13 References 185
14 Disclaimer 187
15 Contact Information 188
16 Copyright Statement 189

Table of Figures

Figure 1: Actor Relationships 18
Figure 2: ebXML Registry Service Architecture 20
Figure 3: The Abstract ebXML Registry Service 21
Figure 4: A Concrete ebXML Registry Service 23
Figure 5: Registry Architecture Supports Flexible Topologies 31
Figure 6: RegistryRequestType Syntax 33
Figure 7: RegistryResponseType Syntax 34
Figure 8: RegistryErrorList Syntax 35
Figure 9: RegistryError Syntax 36
Figure 10: Lifecycle of a RegistryObject 38
Figure 11: Submit Objects Sequence Diagram 39
Figure 12: SubmitObjectsRequest Syntax 39
Figure 13: Update Objects Sequence Diagram 45
Figure 14: UpdateObjectsRequest Syntax 45
Figure 15: Add Slots Sequence Diagram 47
Figure 16: AddSlotsRequest Syntax 47
Figure 17: Remove Slots Sequence Diagram 48
Figure 18: RemoveSlotsRequest Syntax 49
Figure 19: Approve Objects Sequence Diagram 50
Figure 20: ApproveObjectsRequest Syntax 50
Figure 21: Deprecate Objects Sequence Diagram 51
Figure 22: DeprecateObjectsRequest Syntax 52
Figure 23: Undeprecate Objects Sequence Diagram 53
Figure 24: UndeprecateObjectsRequest Syntax 53
Figure 25: Remove Objects Sequence Diagram 54
Figure 26: RemoveObjectsRequest Syntax 55
Figure 27: Submit Ad Hoc Query Sequence Diagram 57
Figure 28: AdhocQueryRequest Syntax 58
Figure 29: AdhocQueryResponse Syntax 59
Figure 30: ResponseOption Syntax 60
Figure 31: Example ebRIM Binding 62
Figure 32: ebRIM Binding for RegistryObjectQuery 65
Figure 33: ebRIM Binding for RegistryEntryQuery 78
Figure 34: ebRIM Binding for AssociationQuery 81
Figure 35: ebRIM Binding for AuditableEventQuery 83
Figure 36: ebRIM Binding for ClassificationQuery 86
Figure 37: ebRIM Binding for ClassificationNodeQuery 88
Figure 38: ebRIM Binding for ClassificationSchemeQuery 93
Figure 39: ebRIM Binding for RegistryPackageQuery 94
Figure 40: ebRIM Binding for ExtrinsicObjectQuery 96

Figure 41: ebRIM Binding for OrganizationQuery....98
Figure 42: ebRIM Binding for ServiceQuery....102
Figure 43: ebRIM Binding for FederationQuery....104
Figure 44: ebRIM Binding for RegistryQuery....105
Figure 45: ebRIM Binding for SubscriptionQuery....106
Figure 46: ebRIM Binding for OrganizationQuery Add PersonName under User and line up 107
Figure 47: The Clause Structure....113
Figure 48: Content Retrieval Sequence Diagram....125
Figure 49: GetContentRequest Syntax....125
Figure 50: GetContentResponse Syntax....126
Figure 51: Content Validation Service....129
Figure 52: Content Cataloging Service....130
Figure 53: Content Management Service: Inline Invocation Model....132
Figure 54: Content Management Service: De-coupled Invocation Model....133
Figure 55: ContentManagementServiceRequestType Syntax....134
Figure 56: Content ContentManagementServiceResponseType Syntax....135
Figure 57: Cataloging Service Configuration....136
Figure 58: Validate Content Protocol....139
Figure 59: ValidateContentRequest Syntax....139
Figure 60: ValidateContentResponse Syntax....140
Figure 61: Catalog Content Protocol....141
Figure 62: CatalogContentRequest Syntax....142
Figure 63: CatalogContentResponse Syntax....143
Figure 64: Example of CPP cataloging using Default XML Cataloging Service....144
Figure 65: GetNotificationsRequest Syntax....150
Figure 66: NotificationType Syntax....151
Figure 67: ObjectRefsNotification Syntax....151
Figure 68: ObjectsNotification Syntax....152
Figure 69: Inter-registry Object References....153
Figure 70: Registry Federations....154
Figure 71: Federation Metadata Example....155
Figure 72: Object Replication....158
Figure 73: Object Relocation....161
Figure 74: Relocate Objects Protocol....162
Figure 75: RelocateObjectsRequest XML Schema....163
Figure 76: AcceptObjectsRequest Syntax....164

Table of Tables

Table 1: Registry Users 17
Table 2: LifeCycle Manager Summary 21
Table 3: Query Manager 22
Table 4: Standard URI Parameters 26
Table 5: QueryManager HTTP Interface 27
Table 6: LifeCycleManager HTTP Interface 29
Table 7: RegistryClient Summary 32
Table 8: Path Filter Expressions for Use Cases 91
Table 9: Default Actor to Role Mappings 173

3 Introduction

3.1 Summary of Contents of Document

This document defines the interface to the ebXML Registry Services as well as interaction protocols, message definitions and XML schema.

A separate document, ebXML Registry Information Model [ebRIM], provides information on the types of metadata that are stored in the Registry as well as the relationships among the various metadata classes.

3.2 General Conventions

The following conventions are used throughout this document:

UML diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific *Implementation* or methodology requirements.

The term "*repository item*" is used to refer to an object (e.g., an XML document or a DTD) that resides in a repository for storage and safekeeping. Each repository item is described by a RegistryObject instance. The RegistryObject catalogs the RepositoryItem with metadata.

The term "*ExtrinsicObject*" is used to refer to an object that provides metadata about a *repository item*.

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

Software practitioners MAY use this document in combination with other ebXML specification documents when creating ebXML compliant software.

3.3 Audience

The target audience for this specification is the community of software developers who are:

- Implementers of ebXML Registry Services
- Implementers of ebXML Registry Clients

Related Documents

The following specifications provide some background and related information to the reader:

a) *ebXML Registry Information Model* [ebRIM]

b) *ebXML Message Service Specification* [ebMS]

c) *ebXML Business Process Specification Schema* [ebBPSS]

d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

- Communicate functionality of Registry services to software developers
- Specify the interface for Registry clients and the Registry
- Provide a basis for future support of more complete ebXML Registry requirements
- Be compatible with other ebXML specifications

4.2 Caveats and Assumptions

This version of the Registry Services Specification is the second in a series of phased deliverables. Later versions of the document will include additional capability as deemed appropriate by the OASIS/ebXML Registry Technical Committee. It is assumed that:

Interoperability requirements dictate that at least one of the normative interfaces as referenced in this specification must be supported.

1. All access to the Registry content is exposed via the interfaces defined for the Registry Services.
2. The Registry makes use of a Repository for storing and retrieving persistent information required by the Registry Services. This is an implementation detail that will not be discussed further in this specification.

5 System Overview

5.1 What The ebXML Registry Does

The ebXML Registry provides a set of services that enable sharing of information between interested parties for the purpose of enabling business process integration between such parties based on the ebXML specifications. The shared information is maintained as objects in a repository and managed by the ebXML Registry Services defined in this document.

5.2 How The ebXML Registry Works

This section describes at a high level some use cases illustrating how Registry clients may make use of Registry Services to conduct B2B exchanges. It is meant to be illustrative and not prescriptive.

The following scenario provides a high level textual example of those use cases in terms of interaction between Registry clients and the Registry. It is not a complete listing of the use cases that could be envisioned. It assumes for purposes of example, a buyer and a seller who wish to conduct B2B exchanges using the RosettaNet PIP3A4 Purchase Order business protocol. It is assumed that both buyer and seller use the same Registry service provided by a third party. Note that the architecture supports other possibilities (e.g. each party uses its own private Registry).

5.2.1 Schema Documents Are Submitted

A third party such as an industry consortium or standards group submits the necessary schema documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the LifeCycleManager service of the Registry described in Section 7.3.

5.2.2 Business Process Documents Are Submitted

A third party, such as an industry consortium or standards group, submits the necessary business process documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the LifeCycleManager service of the Registry described in Section 7.3.

5.2.3 Seller's Collaboration Protocol Profile Is Submitted

The seller publishes its Collaboration Protocol Profile or CPP as defined by [ebCPP] to the Registry. The CPP describes the seller, the role it plays, the services it offers and the technical details on how those services may be accessed. The seller classifies their Collaboration Protocol Profile using the Registry's flexible Classification capabilities.

5.2.4 Buyer Discovers The Seller

The buyer browses the Registry using Classification schemes defined within the Registry using a Registry Browser GUI tool to discover a suitable seller. For example the buyer may look for all parties that are in the Automotive Industry, play a seller role, support the RosettaNet PIP3A4 process and sell Car Stereos.

The buyer discovers the seller's CPP and decides to engage in a partnership with the seller.

5.2.5 CPA Is Established

The buyer unilaterally creates a Collaboration Protocol Agreement or CPA as defined by [ebCPP] with the seller using the seller's CPP and their own CPP as input. The buyer proposes a trading relationship to the seller using the unilateral CPA. The seller accepts the proposed CPA and the trading relationship is established.

Once the seller accepts the CPA, the parties may begin to conduct B2B transactions as defined by [ebMS].

5.3 Registry Users

We describe the actors who use the registry below. Some of the actors are defined in Section **Error! Reference source not found.**. Note that the same entity may represent different actors. For example, a Registration Authority and Registry Administrator may have the same identity.

Table 1: Registry Users

Actor	Function	ISO/IEC 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY have the same identity as Registration Authority
Registered User	Has a contract with the Registration Authority and MUST be authenticated by Registration Authority.		The contract could be a ebXML CPA or some other form of contract.
Registry Guest	Has no contract with Registration Authority. Does not have to be authenticated for Registry access. Cannot change contents of the Registry (MAY be permitted to read some RegistryObjects.)		Note that a Registry Guest is not a Registry Reader.
Submitting Organization	A Registered User who does lifecycle operations on permitted RegistryObjects.	Submitting Organization (SO)	
Registry Reader	A Registered User who has only *read* access		
Responsible Organization	Creates Registry Objects	Responsible Organization (RO)	RO MAY have the same identity as SO

Registry Client	Registered User or Registered Guest		

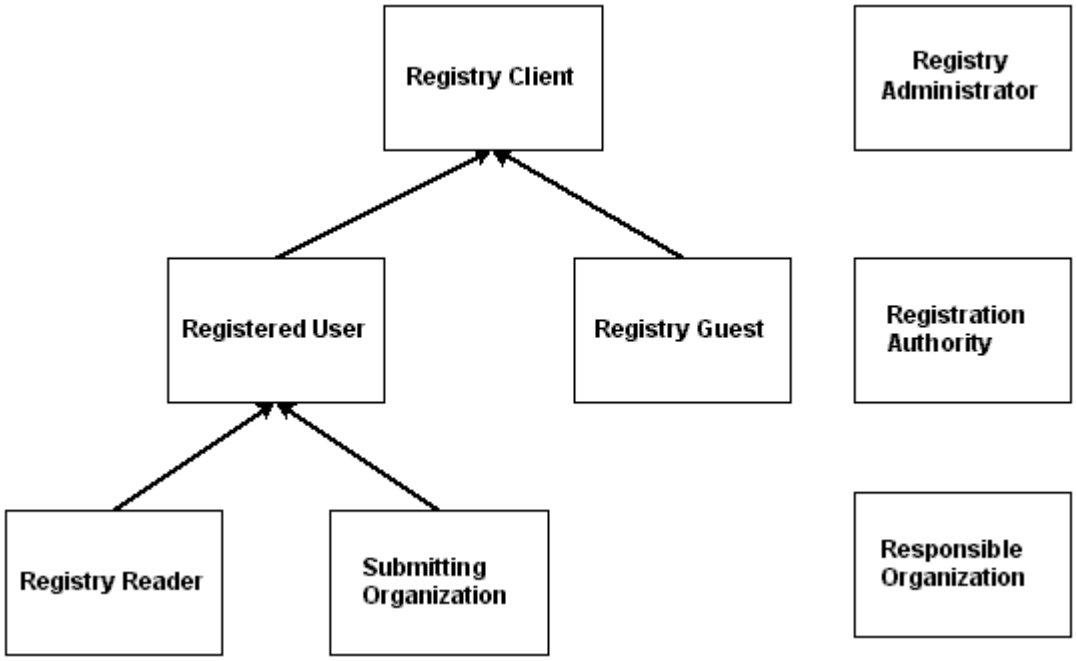


Figure 1: Actor Relationships

Note:
In the current version of the specification the following are true.

A Submitting Organization and a Responsible Organization are the same.

Registration of a user happens out-of-band, i.e, by means not specified in this specification.

A Registry Administrator and Registration Authority are the same.

5.4 Where the Registry Services May Be Implemented

The Registry Services may be implemented in several ways including, as a public web site, as a private web site, hosted by an ASP or hosted by a VPN provider.

5.5 Implementation Conformance

An implementation is a *conforming* ebXML Registry if the implementation meets the conditions in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the implementation meets the conditions in Section 5.5.2. An implementation is a conforming ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and Registry Client.

5.5.1 Conformance as an ebXML Registry

An implementation conforms to this specification as an ebXML Registry if it meets the following conditions:

1. Conforms to the ebXML Registry Information Model [ebRIM].
2. Supports the syntax and semantics of the Registry Interfaces and Security Model.
3. Supports the defined ebXML Registry Schema (Appendix B).
4. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

5.5.2 Conformance as an ebXML Registry Client

An implementation conforms to this specification, as an ebXML Registry Client if it meets the following conditions:

1. Supports the ebXML CPA and bootstrapping process.
2. Supports the syntax and the semantics of the Registry Client Interfaces.
3. Supports the defined ebXML Error Message DTD.
4. Supports the defined ebXML Registry Schema (Appendix B).

6 ebXML Registry Architecture

The ebXML Registry architecture consists of an ebXML Registry Service and ebXML Registry Clients. The ebXML Registry Service provides the methods for managing a repository. An ebXML Registry Client is an application used to access the Registry.

Figure 2: ebXML Registry Service Architecture

6.1 Registry Service Described

The ebXML Registry Service is comprised of a robust set of interfaces designed to fundamentally manage the objects and inquiries associated with the ebXML Registry. The two primary interfaces for the Registry Service consist of:

- A Lifecycle Management interface that provides a collection of methods for managing objects within the Registry.
- A Query Management Interface that controls the discovery and retrieval of information from the Registry.

A registry client program utilizes the services of the registry by invoking methods on one of the above interfaces defined by the Registry Service. This specification defines the interfaces exposed by the Registry Service as well as the interface for the Registry Client.

6.2 Abstract Registry Service

The architecture defines the ebXML Registry as an abstract registry service that is defined as:

1. A set of interfaces that must be supported by the registry.
2. The set of methods that must be supported by each interface.
3. The parameters and responses that must be supported by each method.

The abstract registry service neither defines any specific implementation for the ebXML Registry, nor does it specify any specific protocols used by the registry. Such implementation details are described by concrete registry services that realize the abstract registry service.

The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide two key functional interfaces called **`QueryManager`**[1] (QM) and **`LifeCycleManager`**[2] (LM).

Figure 3: The Abstract ebXML Registry Service

0 provides hyperlinks to the abstract service definition in the Web Service Description Language (WSDL) syntax.

6.2.1 LifeCycleManager Interface

This is the interface exposed by the Registry Service that implements the object lifecycle management functionality of the Registry. Its methods are invoked by the Registry Client. For example, the client may use this interface to submit objects, to classify and associate objects and to deprecate and remove objects. For this specification the semantic meaning of submit, classify, associate, deprecate and remove is found in [ebRIM].

Table 2: LifeCycle Manager Summary

Method Summary of LifeCycleManager	
`RegistryResponse`	**`acceptObjects`**`(AcceptObjectsRequest req)` Accepts one or more objects to a registry during object relocation.
`RegistryResponse`	**`approveObjects`**`(ApproveObjectsRequest req)` Approves one or more previously submitted objects.
`RegistryResponse`	**`deprecateObjects`**`(DeprecateObjectsRequest req)` Deprecates one or more previously submitted objects.

[1] Known as ObjectQueryManager in V1.0

[2] Known as ObjectManager in V1.0

RegistryResponse	**removeObjects**(RemoveObjectsRequest req) Removes one or more previously submitted objects from the Registry.
RegistryResponse	**submitObjects**(SubmitObjectsRequest req) Submits one or more objects and possibly related metadata such as Associations and Classifications.
RegistryResponse	**updateObjects**(UpdateObjectsRequest req) Updates one or more previously submitted objects.
RegistryResponse	**addSlots**(AddSlotsRequest req) Add slots to one or more registry entries.
RegistryResponse	**relocateObjects**(RelocateObjectsRequest req) Relocate one or more objects from one registry to another.
RegistryResponse	**removeSlots**(RemoveSlotsRequest req) Remove specified slots from one or more registry entries.

6.2.2 QueryManager Interface

This is the interface exposed by the Registry that implements the Query management service of the Registry. Its methods are invoked by the Registry Client. For example, the client may use this interface to perform browse and drill down queries or ad hoc queries on registry content.

Table 3: Query Manager

Method Summary of QueryManager	
GetContentResponse	**getContent**(GetContentRequest req) Submit an ad hoc query request. This method is being deprecated and may go away in version 4.
GetNotificationsResponse	**getNotifications**(GetNotificationsRequest req) Submit a request to get event notifications.
AdhocQueryResponse	**submitAdhocQuery**(AdhocQueryRequest req) Submit an ad hoc query request.
RegistryObject	**getRegistryObject**(String id) Submit a request to get the RegistryObject that matches the specified id.
RepositoryItem	**getRepositoryItem**(String id) Submit a request to get the repository item that matches the specified id. This is the same as the id of the ExtrinsicObject that catalogs this repository item.

6.3 Concrete Registry Services

The architecture allows the abstract registry service to be mapped to one or more concrete registry services defined as:

- Implementations of the interfaces defined by the abstract registry service.
- Bindings of these concrete interfaces to specific communication protocols.

This specification describes the following concrete bindings for the abstract registry service:

- A SOAP binding using the HTTP protocol
- An ebXML Messaging Service (ebMS) binding
- An HTTP binding

A registry must implement at least one concrete binding between SOAP and ebMS concrete bindings for the abstract registry service as shown in Figure 5. In addition a registry must implement the HTTP binding for the abstract registry service as shown in Figure 5.

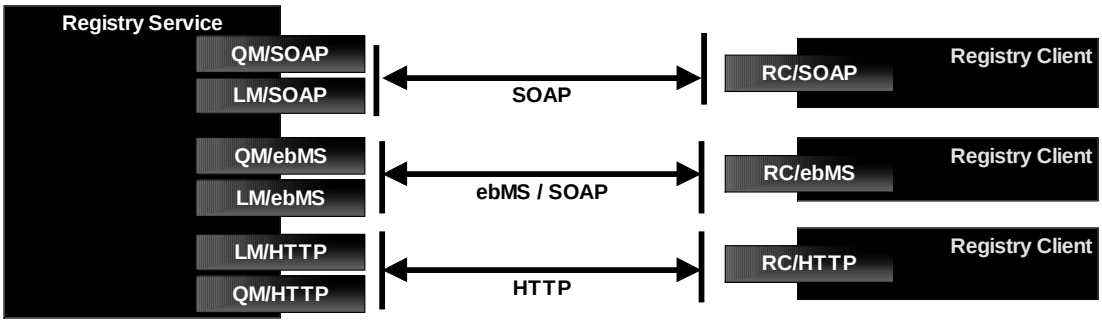


Figure 5: A Concrete ebXML Registry Service

Figure 5 shows a concrete implementation of the abstract ebXML Registry (RegistryService) on the left side. The RegistryService provides the QueryManager and LifeCycleManager interfaces available with multiple protocol bindings (SOAP and ebMS).

Figure 5 also shows two different clients of the ebXML Registry on the right side. The top client uses SOAP interface to access the registry while the lower client uses ebMS interface. Clients use the appropriate concrete interface within the RegistryService service based upon their protocol preference.

6.4 SOAP Binding

6.4.1 WSDL Terminology Primer

This section provides a brief introduction to Web Service Description Language (WSDL) since the SOAP binding is described using WSDL syntax. WSDL provides the ability to describe a web service in abstract as well as with concrete bindings to specific protocols. In WSDL, an abstract service consists of one or more `port types` or end-points. Each port type consists of a collection of `operations`. Each operation is defined in terms of `messages` that define what data is exchanged as part of that operation. Each message is typically defined in terms of elements within an XML Schema definition.

An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract service may be used to define a concrete service by binding it to a specific protocol. This binding is done by providing a `binding` definition for each abstract port type that defines additional

protocols specific details. Finally, a concrete `service` definition is defined as a collection of `ports`, where each port simply adds address information such as a URL for each concrete port.

6.4.2 Concrete Binding for SOAP

This section assumes that the reader is somewhat familiar with SOAP and WSDL. The SOAP binding to the ebXML Registry is defined as a web service description in WSDL as follows:

- A single service element with name "RegistryService" defines the concrete SOAP binding for the registry service.
- The service element includes two port definitions, where each port corresponds with one of the interfaces defined for the abstract registry service. Each port includes an HTTP URL for accessing that port.
- Each port definition also references a binding element, one for each interface defined in the WSDL for the abstract registry service.

```
<service name = "RegistryService">
       <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
            <soap:address location = "http://your_URL_to_your_QueryManager"/>
       </port>

       <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
            <soap:address location = "http://your_URL_to_your_QueryManager"/>
       </port>
</service>
```

The complete WSDL description for the SOAP binding can be obtained via a hyperlink in 0.

6.5 ebXML Message Service Binding

6.5.1 Service and Action Elements

When using the ebXML Messaging Services Specification, ebXML Registry Service elements correspond to Messaging Service elements as follows:

- The value of the Service element in the MessageHeader is an ebXML Registry Service interface name (e.g., "LifeCycleManager"). The type attribute of the Service element should have a value of "ebXMLRegistry".
- The value of the Action element in the MessageHeader is an ebXML Registry Service method name (e.g., "submitObjects").

```
<eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
<eb:Action>submitObjects</eb:Action>
```

Note that the above allows the Registry Client only one interface/method pair per message. This implies that a Registry Client can only invoke one method on a specified interface for a given request to a registry.

6.5.2 Synchronous and Asynchronous Responses

All methods on interfaces exposed by the registry return a response message.

Asynchronous response

When a message is sent asynchronously, the Registry will return two response messages. The first message will be an immediate response to the request and does not reflect the actual response for the request. This message will contain:

- MessageHeader
- RegistryResponse element including:
  - status attribute with value **Unavailable**

The Registry delivers the actual Registry response element with non-empty content asynchronously at a later time. The delivery is accomplished by the Registry invoking the onResponse method on the RegistryClient interface as implemented by the registry client application. The onResponse method includes a RegistryResponse element as shown below:

- MessageHeader
- RegistryResponse element including:
  - Status attribute (Success, Failure)
  - Optional RegistryErrorList

Synchronous response

When a message is sent synchronously, the Message Service Handler will hold open the communication mechanism until the Registry returns a response. This message will contain:

- MessageHeader
- RegistryResponse element including:
  - Status attribute (Success, Failure)
  - Optional RegistryErrorList

6.5.3 ebXML Registry Collaboration Profiles and Agreements

The ebXML CPP specification [ebCPP] defines a Collaboration-Protocol Profile (CPP) and a Collaboration-Protocol Agreement (CPA) as mechanisms for two parties to share information regarding their respective business processes. That specification assumes that a CPA has been agreed to by both parties in order for them to engage in B2B interactions.

This specification does not mandate the use of a CPA between the Registry and the Registry Client. However if the Registry does not use a CPP, the Registry shall provide an alternate mechanism for the Registry Client to discover the services and other information provided by a CPP. This alternate mechanism could be a simple URL.

The CPA between clients and the Registry should describe the interfaces that the Registry and the client expose to each other for Registry-specific interactions. The definition of the Registry CPP template and a Registry Client CPP template are beyond the scope of this document.

6.6 HTTP Binding

The ebXML Registry abstract interface defines a HTTP binding that enables access to the registry over HTTP protocol. The HTTP binding maps the abstract registry interfaces to an HTTP interface. It defines the URI parameters and their usage patterns that must be used to specify the interface, method and invocation parameters in order to invoke a method on a registry interface such as the QueryManager interface.

The HTTP binding also defines the return values that are synchronously sent back to the client as the HTTP response for the HTTP request.

6.6.1 Standard URI Parameters

This section defines the normative URI parameters that must be supported by the HTTP Interface. A Registry may implement additional URI parameters in addition to these parameters.

URI Parameter Name	Required	Description	Example
interface	YES	Defines the interface or object to call methods on.	Example: QueryManager
method	YES	Defines the method to be carried out on the given interface.	Example: submitAdhocQueryRequest
param-<key>	NO	Defines named parameters to be passed into a method call.	Example: param-id=888-999-8877h

Table 4: Standard URI Parameters

6.6.2 QueryManager HTTP Interface

The HTTP Interface to QueryManager *must* be supported by all registries.

The HTTP Interface to QueryManager defines that the interface parameter must be "QueryManager". In addition the following method parameters are defined by the QueryManager HTTP Interface.

Method	Parameters	Return Value	HTTP Request Type
getNotifications	GetNotificationsRequest	GetNotificationsResponse	POST
getRegistryObject	Id	An instance of a leaf class that is a concrete sub-class of RegistryObject that matches the specified id.	GET
getRepositoryItem	Id	RepositoryItem that matches	GET

Method	Parameters	Return Value	HTTP Request Type
		the specified id. Note that a RepositoryItem may be arbitrary content (e.g. a GIF image).	
submitAdhocQueryRequest	AdhocQueryRequest	AdhocQueryResponse for the specified AdhocQueryRequest.	POST

Table 5: QueryManager HTTP Interface

Note that in the examples that follow name space declarations are omitted to conserve space. Also note that some lines may be wrapped due to lack of space.

Sample getRegistryObject Request

The following example shows a getRegistryObject request.

```
GET /http?interface=QueryManager&method=getRegistryObject&param-id=
urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6 HTTP/1.1
```

Sample getRegistryObject Response

The following example shows an ExtrinsicObject, which is a specialized sub-class of RegistryObject being returned as a response to the getRegistryObject method invocation.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<ExtrinsicObject id = "urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6"
   objectType="urn:uuid:32bbb291-0291-486d-a80d-cdd6cd625c57">
   <Name>
      <LocalizedString value = "Sample Object"/>
   </Name>
</ExtrinsicObject>
```

Sample getRepositoryItem Request

The following example shows a getRepositoryItem request.

```
GET /http?interface=QueryManager&method=getRepositoryItem&param-id=
urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6 HTTP/1.1
```

Sample getRepositoryItem Response

The following example assumes that the repository item was a Collaboration Protocol Profile as defined by [ebCPP].

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
< CollaborationProtocolProfile>
...
</CollaborationProtocolProfile>
```

Sample submitAdhocQueryRequest Request

The following example shows how an HTTP POST request is used to invoke the submitAdhocQueryRequest method of QueryManager.

```
POST /http?interface=QueryManager&method=submitAdhocQueryRequest HTTP/1.1
User-Agent: Foo-ebXML/1.0
Host: www.registryserver.com
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<AdhocQueryRequest>
...
</AdhocQueryRequest>
```

Sample submitAdhocQueryRequest Response

The following example shows an AdhocQueryResponse that is returned in response to an AdhocQueryRequest.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<AdhocQueryResponse>
...
</AdhocQueryResponse>
```

6.6.3 LifeCycleManager HTTP Interface

The HTTP Interface to LifeCycleManager *may* be supported by a registry.

The HTTP Interface to LifeCycleManager defines that the interface parameter must be "LifeCycleManager". In addition the following method parameters are defined by the LifeCycleManager HTTP Interface.

Method	Parameters	Return Value	HTTP Request Type
acceptObjects	AcceptObjectsRequest	RegistryResponse	POST
approveObjects	ApproveObjectsRequest	RegistryResponse	POST
deprecateObjects	DeprecateObjectsRequest	RegistryResponse	POST
relocateObjects	RelocateObjectsRequest	RegistryResponse	POST
removeObjects	RemoveObjectsRequest	RegistryResponse	POST
submitObjects	SubmitObjectsRequest	RegistryResponse	POST
updateObjects	UpdateObjectsRequest	RegistryResponse	POST
addSlots	AddSlotsRequest	RegistryResponse	POST
removeSlots	RemoveSlotsRequest	RegistryResponse	POST

Table 6: LifeCycleManager HTTP Interface

Note that in the examples that follow name space declarations are omitted to conserve space. Also note that some lines may be wrapped due to lack of space.

Sample submitObjects Request

The following example shows how an HTTP POST request is used to invoke the submitObjects method in LifeCycleManager.

```
POST /http?interface=LifeCycleManager&method=submitObjects HTTP/1.1
User-Agent: Foo-ebXML/1.0
Host: www.registryserver.com
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<SubmitObjectsRequest>
...
</SubmitObjectRequest>
```

Sample submitObjects Response

The following example shows a sample response returned by the submitObjects method in LifeCycleManager.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<RegistryResponse>
...
</RegistryResponse>
```


6.6.4 Security Considerations

The HTTP interface supports the same mechanisms that are specified in chapter 12. Authentication may be performed by the registry on a per message basis by verifying any digital signatures present, as well as at the HTTP transport level using Basic or Digest authentication. When using the HTTP binding, authentication credentials are specified using the SignatureList element within a request or response as defined by the RegistryRequestType (6.9.1) and RegistryResponseType (6.9.2) elements in the registry XML schema.

6.6.5 Exception Handling

Exception handling is consistent with exception handling in other registry interface bindings. Errors must be reported in a `RegistryErrorList`, and sent back to the client on the same connection as the request.

When errors occur, the HTTP status code and message should correspond to the error(s) being reported in the `RegistryErrorList`. For example, if the `RegistryErrorList` reports that an object wasn't found, therefore cannot be returned, an appropriate error code should be 404, with a message of "ObjectNotFoundException". A detailed list of HTTP status codes can be found in [RFC2616]. The mapping between registry exceptions and HTTP status codes is currently unspecified.

6.7 Registry Clients

6.7.1 Registry Client Described

The Registry Client interfaces may be local to the registry or local to the user. Figure 7 depicts the two possible topologies supported by the registry architecture with respect to the Registry and Registry Clients. The picture on the left side shows the scenario where the Registry provides a web based “thin client” application for accessing the Registry that is available to the user using a common web browser. In this scenario the Registry Client interfaces reside across the Internet and are local to the Registry from the user’s view. The picture on the right side shows the scenario where the user is using a “fat client” Registry Browser application to access the registry. In this scenario the Registry Client interfaces reside within the Registry Browser tool and are local to the Registry from the user’s view. The Registry Client interfaces communicate with the Registry over the Internet in this scenario.

A third topology made possible by the registry architecture is where the Registry Client interfaces reside in a server side business component such as a Purchasing business component. In this topology there may be no direct user interface or user intervention involved. Instead, the Purchasing business component may access the Registry in an automated manner to select possible sellers or service providers based on current business needs.

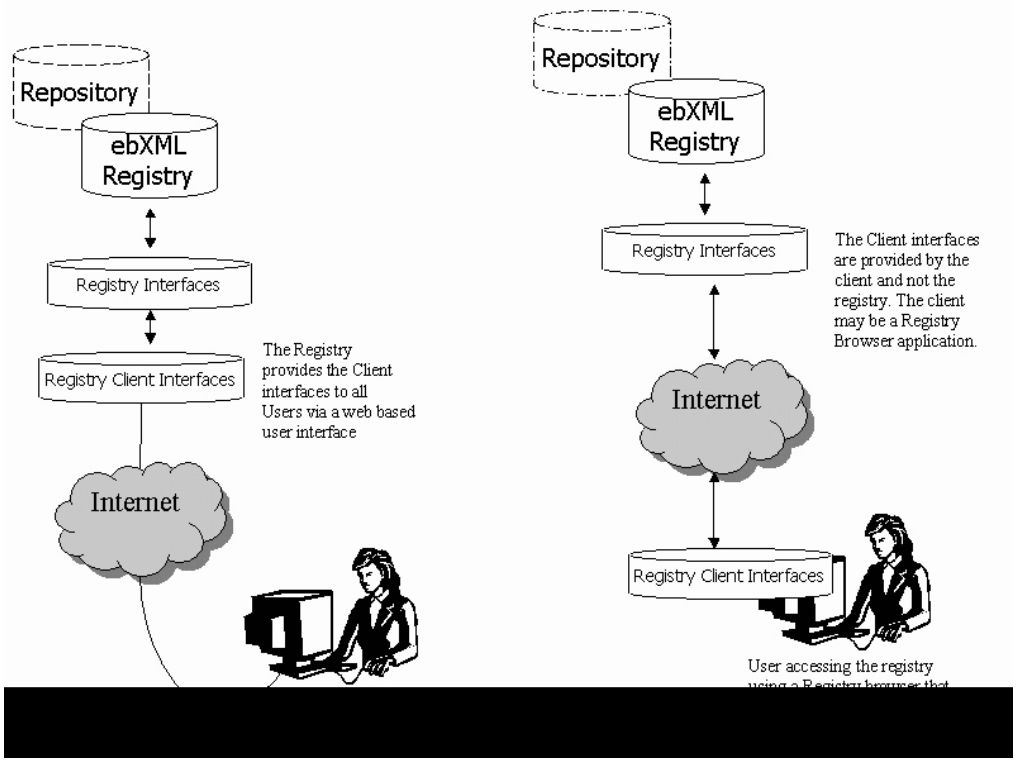


Figure 7: Registry Architecture Supports Flexible Topologies

6.7.2 Registry Communication Bootstrapping

Before a client can access the services of a Registry, there must be some communication bootstrapping between the client and the registry. The most essential aspect of this bootstrapping process is for the client to discover addressing information (e.g. an HTTP URL) to each of the concrete service interfaces of the Registry. The client may obtain the addressing information by discovering the ebXML Registry in a public registry such as UDDI or within another ebXML Registry.

- In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available in a WSDL desription for the registry. This WSDL conforms to the template WSDL description in Appendix A.1. This WSDL description may be discovered in a in a registry of registries.
- In case of ebMS binding, the information exchange between the client and the registry may be accomplished in a registry specific manner, which may involve establishing a CPA between the client and the registry. Once the information exchange has occurred the Registry and the client will have addressing information (e.g. URLs) for the other party.
- In case of HTTP binding the client may obtain the base URL to the registry by a lookup in a registry of registries.

Communication Bootstrapping for SOAP Binding

Each ebXML Registry must provide a WSDL description for its RegistryService as defined by Appendix A.1. A client uses the WSDL description to determine the address information of the RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the

RegistryService may be accessed via a URL specified in the WSDL for the registry.

The use of WSDL enables the client to use automated tools such as a WSDL compiler to generate stubs that provide access to the registry in a language specific manner.

At minimum, any client may access the registry over SOAP/HTTP using the address information within the WSDL, with minimal infrastructure requirements other than the ability to make synchronous SOAP call to the SOAP based ports on the RegistryService.

Communication Bootstrapping for ebXML Message Service Binding

Since there is no previously established CPA between the Registry and the RegistryClient, the client must know at least one Transport-specific communication address for the Registry. This communication address is typically a URL to the Registry, although it could be some other type of address such as an email address. For example, if the communication used by the Registry is HTTP, then the communication address is a URL. In this example, the client uses the Registry's public URL to create an implicit CPA with the Registry. When the client sends a request to the Registry, it provides a URL to itself. The Registry uses the client's URL to form its version of an implicit CPA with the client. At this point a session is established within the Registry. For the duration of the client's session with the Registry, messages may be exchanged bi-directionally as required by the interaction protocols defined in this specification.

Communication Bootstrapping for HTTP Binding

Communication between a client and the HTTP interface is established based upon the base URL of the HTTP interface to the registry. No other communication bootstrapping is required.

6.7.3 RegistryClient Interface

This is the principal interface implemented by a Registry client. The client provides this interface when creating a connection to the Registry. It provides the methods that are used by the Registry to deliver asynchronous responses to the client. Note that a client need not provide a RegistryClient interface if the [CPA] between the client and the registry does not support asynchronous responses.

The registry sends all asynchronous responses to operations via the onResponse method.

Table 7: RegistryClient Summary

Method Summary of RegistryClient	
void	**onResponse**(RegistryResponse resp) Notifies client of the response sent by registry to previously submitted request.

6.7.4 Registry Response

The RegistryResponse is a common class defined by the Registry interface that is used by the registry to provide responses to client requests.

6.8 Interoperability Requirements

6.8.1 Client Interoperability

The architecture requires that any ebXML compliant registry client can access any ebXML compliant registry service in an interoperable manner. An ebXML Registry must implement a HTTP binding and either or both of the ebMS and SOAP/HTTP bindings.

6.9 Registry Requests and Responses

This section describes the generic aspects that are common to all requests/responses sent/received by registry clients/registry to the registry/registry clients.

Each registry request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of "Success" back to the client. In the event of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of "Unavailable" back to the client. Failure occurs when one or more Error conditions are raised in the processing of the submitted objects. Warning messages do not result in failure of the request.

6.9.1 RegistryRequestType

The RegistryRequestType is used as a common base type for all registry requests.

Syntax:

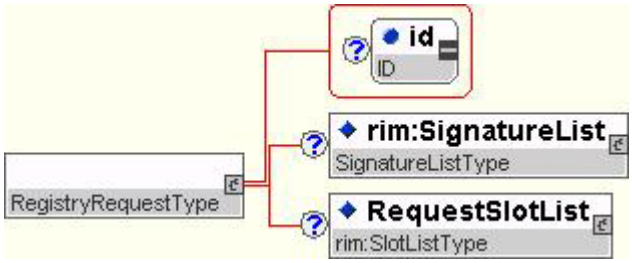


Figure 9: RegistryRequestType Syntax

Parameters:

- *id:* This parameter specifies a request identifier that is used by the corresponding response to correlate the response with its request. It may also be used to correlate a request with another related request.
- *RequestSlotList:* This parameter specifies a collection of Slot instances. A RegistryReuqestType may include Slots as an extensibility mechanism that provides a means of adding dynamic attributes in form of Slots.
- *SignatureList:* This parameter specifies a collection of Signature elements as defined by [XMLDSIG]. Each Signature specified in the SignatureList must be verified by the registry before processing the request.

940 **Returns:**

941 All RegistryRequests returns a response derived from the common RegistryResponseType base
942 type.

943 **Exceptions:**

944 The following exceptions are common to all requests:

945 ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an
946 operation for which she was not authorized.

947 ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an
948 operation that was semantically invalid.

949 ▪ SignatureValidationException: Indicates that a Signature specified for the request
950 failed to validate.

951 ▪ TimeoutException: Indicates that the processing time for the request exceeded a
952 registry specific limit.

953 ▪ UnsupportedCapabilityException: Indicates that this registry did not support the
954 capability required to service the request.

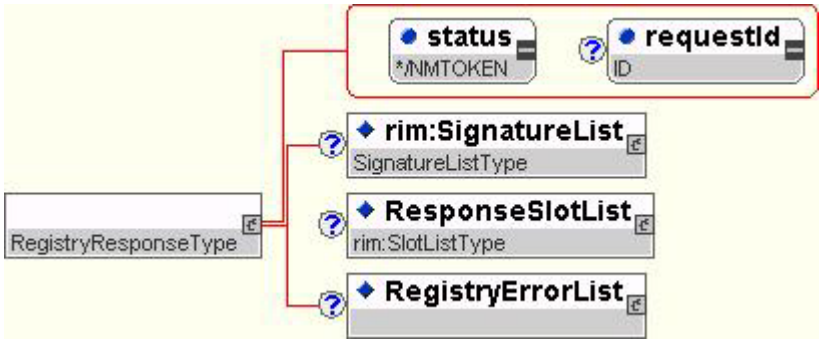
955

956 6.9.2 RegistryResponseType

957 The RegistryResponseType is used as a common base type for all registry responses.

958

959 **Syntax:**



960
961 **Figure 10: RegistryResponseType Syntax**

962 **Parameters:**

963 ▪ *requestId:* This parameter specifies the id of the request for which this is a
964 response. It matches value of the id attribute of the corresponding
965 RegistryRequestType.

966 ▪ *RegistryErrorList***:** This parameter specifies an optional collection of
967 RegistryError elements in the event that there are one or more errors that were
968 encountered while the registry processed the request for this response. This is
969 described in more detail in 6.9.3.

- *ResponseSlotList:* This parameter specifies a collection of Slot instances. A RegistryResponseType may include Slots as an extensibility mechanism that provides a means of adding dynamic attributes in form of Slots.
- *SignatureList:* This parameter specifies a collection of Signature elements as defined by [DSIG]. Each Signature specified in the SignatureList should be verified by the receiver before processing the response.
- *status*: This enumerated value is used to indicate the status of the request. Values for status are as follows:

  - Success - This status specifies that the request was successful.
  - Failure - This status specifies that the request encountered a failure. One or more errors must be included in the RegistryErrorList in this case.
  - Unavailable – This status specifies that the response is not yet available. This may be the case if this RegistryResponseType represents an immediate response to an asynchronous request where the actual response is not yet available.

.

6.9.3 RegistryResponse

RegistryResponse is an element whose base type is RegistryResponseType. It adds no additional elements or attributes beyond those described in RegistryResponseType. RegistryResponse is used by many requests as their response.

6.9.4 RegistryErrorList

A RegistryErrorList specifies an optional collection of RegistryError elements in the event that there are one or more errors that were encountered while the registry processes a request.

Syntax:

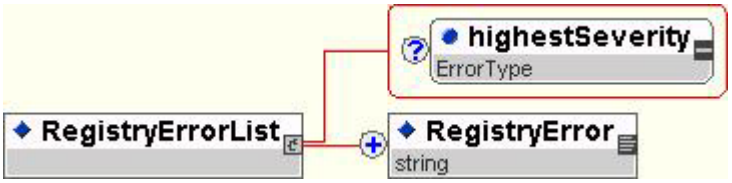


Figure 11: RegistryErrorList Syntax

Parameters:

- *highestSeverity:* This parameter specifies the ErrorType for the highest severity RegistryError in the RegistryErrorList. Values for highestSeverity are defined by ErrorType in 6.9.6.
- *RegistryError***:** A RegistryErrorList has one or more RegistryErrors. A RegistryError specifies an error or warning message that is encountered while the

registry processes a request. RegistryError is defined in 6.9.5.

6.9.5 RegistryError

A RegistryError specifies an error or warning message that is encountered while the registry processes a request.

Syntax:

Figure 12: RegistryError Syntax

Parameters:

- *codeContext:* This parameter specifies a string that indicates contextual text that provides additional detail to the errorCode. For example, if the errorCode is InvalidRequestException the codeContext may provide the reason why the request was invalid.
- *errorCode*: This parameter specifies a string that indicates the error that was encountered. Implementations must set this parameter to the Exception or Error as defined by this specification (e.g. InvalidRequestException).
- *location*: This parameter specifies a string that indicated where in the code the error occured. Implementations should show the stack trace and/or, code module and line number information where the error was encountered in code.
- *severity*: This parameter specifies an enumerated value of type ErrorType which indicates the severity of error that was encountered. ErrorType is described in 6.9.6.

6.9.6 ErrorType

The ErrorType type defines a set of enumerated values that indicate the different type of errors that a registry may encounter while processing a request. The possible values are Warning and Error.

Warning

A Warning is a non-fatal error encountered by the registry while processing a request. A registry must return a status of Success in the RegistryResponse for a request that only encountered Warnings during its processing and encountered no Errors.

Error

An Error is a fatal error encountered by the registry while processing a request. A registry must

1038 return a status of Failure in the RegistryResponse for a request that encountered Errors during its
1039 processing.
1040
1041

7 Lifecycle Management Service

This section defines the Lifecycle Management service of the Registry. The Lifecycle Management Service is a sub-service of the Registry service. It provides the functionality required by RegistryClients to manage the lifecycle of repository items (e.g. XML documents required for ebXML business processes). The Lifecycle Management Service can be used with all types of repository items as well as the metadata objects specified in [ebRIM] such as Classification and Association.

The minimum-security policy for an ebXML registry is to accept content from any client if a certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the content.

7.1 Lifecycle of a RegistryObject

The main purpose of the LifeCycleManagement service is to manage the lifecycle of RegistryObjects. Figure 13 shows the typical lifecycle of a RegistryObject.

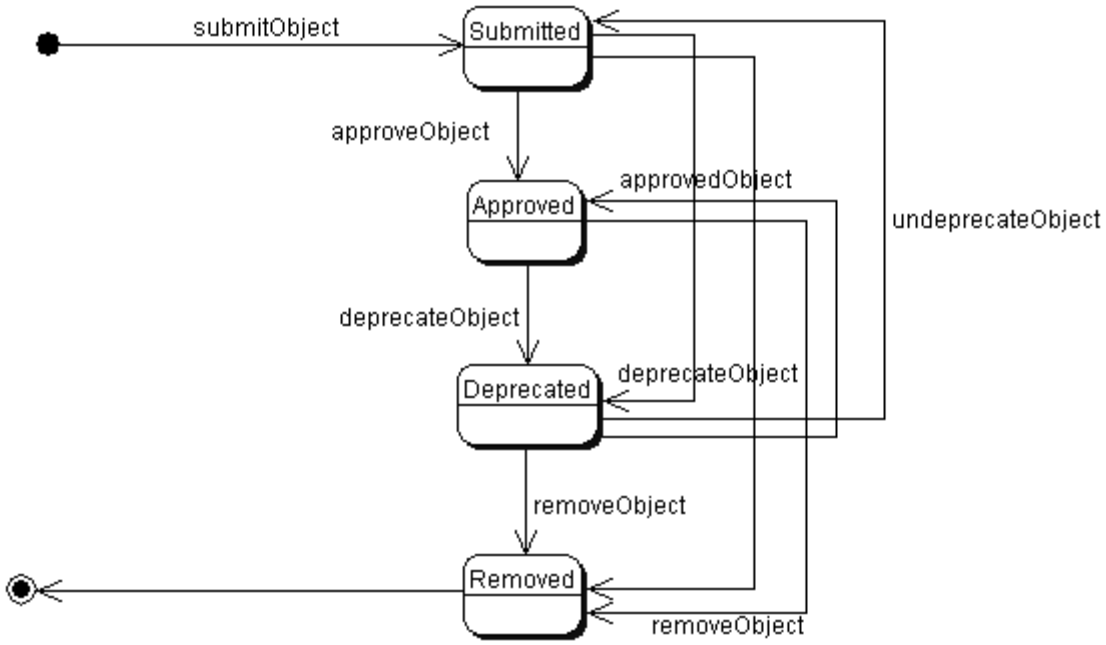


Figure 13: Lifecycle of a RegistryObject

7.2 RegistryObject Attributes

A repository item is associated with a set of standard metadata defined as attributes of the RegistryObject class and its sub-classes as described in [ebRIM]. These attributes reside outside of the actual repository item and catalog descriptive information about the repository item. XML elements called ExtrinsicObject and other elements (See Appendix B.1 for details) encapsulate all object metadata attributes defined in [ebRIM] as XML attributes.

7.3 The Submit Objects Protocol

This section describes the protocol of the Registry Service that allows a RegistryClient to submit one or more RegistryObjects and/or repository items using the LifeCycleManager on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

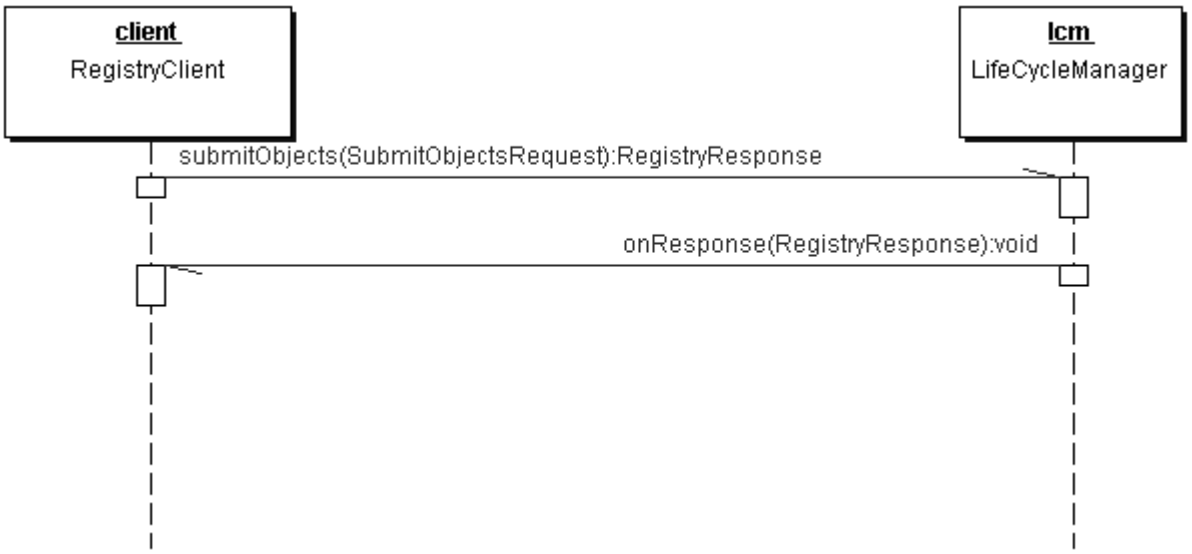


Figure 15: Submit Objects Sequence Diagram

For details on the schema for the Business documents shown in this process refer to Appendix B.

7.3.1 SubmitObjectsRequest

The SubmitObjectsRequest is used by a client to submit RegistryObjects and/or repository items to the registry.

Syntax:

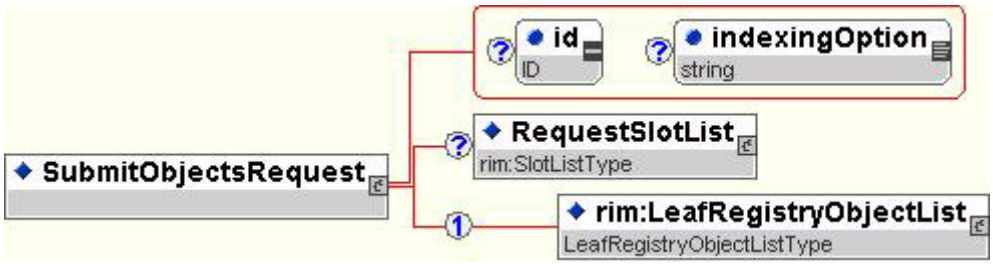


Figure 16: SubmitObjectsRequest Syntax

Parameters:

- *LeafRegistryObjectsList:* This parameter specifies a collection of RegistryObject instances that are being submitted to the registry. The RegistryObjects in the list may be brand new objects being submitted to the registry or they may be current objects already existing in the registry. In case of existing objects the registry

must treat them in the same manner as UpdateObjectsRequest and simply update the existing objects.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException:* Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException:* Indicates that the requestor referenced an object within the request that was not found.
- *ObjectExistsException:* Indicates that the requestor tried to submit an object using an id that matched the id of an existing object in the registry.
- *InvalidRequestException:* Indicates that the requestor attempted to perform an operation which was semantically invalid.
- *UnsupportedCapabilityException:* Indicates that the requestor attempted to submit some content that is not supported by the registry.
- *QuotaExceededException:* Indicates that the requestor attempted to submit more content than the quota allowed for them by the registry.

7.3.2 RegistryResponse

The RegistryResponse is sent by the registry as a response to several different requests. It is a simple response that can signal the status of a request and any errors or exceptions that may have occurred during the processing of that request. The details of RegistryResponse are decsribed by the RegistryResponseType in 6.9.2.

7.3.3 Universally Unique ID Generation

As specified by [ebRIM], all objects in the registry have a unique id contained within the value of the "id" attribute of a RegistryObject instance. The id must be a Universally Unique Identifier (UUID) and must conform to the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID].

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

The registry usually generates this id. The client may optionally supply the id attribute for submitted objects. If the client supplies the `id` and it conforms to the format of a URN that specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the id for the object. In this case, the registry must honour a client-supplied id and use it as the id attribute of the object in the registry. If the id is not unique within the registry, the registry must return ObjectExistsException.

If the client does not supply an id for a submitted object then the registry must generate a universally unique id . Whether the client generates the id or whether the registry generates it, it must be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

7.3.4 ID Attribute And Object References

The id attribute of an object may be used by other objects to reference the first object. Such references are common both within the SubmitObjectsRequest as well as within the registry. Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the SubmitObjectsRequest as well as to refer to an object within the registry. An object in the SubmitObjectsRequest that needs to be referred to within the request document may be assigned an id by the submitter so that it can be referenced within the request. The submitter may give the object a proper UUID URN, in which case the id is permanently assigned to the object within the registry. Alternatively, the submitter may assign an arbitrary id (not a proper UUID URN) as long as the id is a unique anyURI value within the request document. In this case the id serves as a linkage mechanism within the request document but must be ignored by the registry and replaced with a registry generated UUID upon submission.

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

7.3.5 Audit Trail

The RS must create AuditableEvent objects with eventType *Created* for each RegistryObject created via a SubmitObjectsRequest.

7.3.6 Sample SubmitObjectsRequest

The following example shows several different use cases in a single SubmitObjectsRequest. It does not show the complete SOAP or [ebMS] Message with the message header and additional payloads in the message for the repository items.

A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects that are being submitted. It may also contain any number of ObjectRefs to link objects being submitted to objects already within the registry.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<SubmitObjectsRequest
  xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
  xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
  xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
  >

  <rim:LeafRegistryObjectList>

    <!--
    The following 3 objects package specified ExtrinsicObject in specified
      RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
      being  submitted
      -->
```



```
    <rim:RegistryPackage id = "acmePackage1" >
      <rim:Name>
        <rim:LocalizedString value = "RegistryPackage #1"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's package #1"/>
      </rim:Description>
    </rim:RegistryPackage>

    <rim:ExtrinsicObject id = "acmeCPP1"  >
      <rim:Name>
        <rim:LocalizedString value = "Widget Profile" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's profile for selling widgets" />
      </rim:Description>
    </rim:ExtrinsicObject>

    <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
= "acmePackage1" targetObject = "acmeCPP1" />

    <!--
      The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
      Where the RegistryPackage is being submitted and the ExtrinsicObject is
      already in registry
      -->

    <rim:RegistryPackage id = "acmePackage2" >
      <rim:Name>
        <rim:LocalizedString value = "RegistryPackage #2"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's package #2"/>
      </rim:Description>
    </rim:RegistryPackage>

    <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!--
      The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
      where the RegistryPackage and the ExtrinsicObject are already in registry
      -->

    <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
    <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>

    <!-- id is unspecified implying that registry must create a uuid for this object -->

    <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>

    <!--
      The following 3 objects externally link specified ExtrinsicObject using
      specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
      are being submitted
      -->

    <rim:ExternalLink id = "acmeLink1" externalURI="http://www.acme.com">
      <rim:Name>
        <rim:LocalizedString value = "Link #1"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's Link #1"/>
      </rim:Description>
    </rim:ExternalLink>

    <rim:ExtrinsicObject id = "acmeCPP2"  >
      <rim:Name>
        <rim:LocalizedString value = "Sprockets Profile" />
      </rim:Name>
```



```
      <rim:Description>
        <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
      </rim:Description>
    </rim:ExtrinsicObject>

    <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>

    <!--
      The following 2 objects externally link specified ExtrinsicObject using specified
      ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
      is already in registry. Note that the targetObject points to an ObjectRef in a
      previous line
      -->

    <rim:ExternalLink id = "acmeLink2" externalURI="http://www.acme2.com">
      <rim:Name>
        <rim:LocalizedString value = "Link #2"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's Link #2"/>
      </rim:Description>
    </rim:ExternalLink>

    <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
"ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>

    <!--
      The following 3 objects externally identify specified ExtrinsicObject using specified
      ExternalIdentifier, where the ExternalIdentifier is being submitted and the
      ExtrinsicObject is already in registry. Note that the targetObject points to an
      ObjectRef in a previous line
      -->

    <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
      <rim:Name>
        <rim:LocalizedString value = "DUNS"/>
      </rim:Name>

      <rim:Description>
        <rim:LocalizedString value = "This is the DUNS scheme"/>
      </rim:Description>
    </rim:ClassificationScheme>

    <rim:ExternalIdentifier id = "acmeDUNSId"  identificationScheme="DUNS-id" value =
"13456789012">
      <rim:Name>
        <rim:LocalizedString value = "DUNS" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "DUNS ID for ACME"/>
      </rim:Description>
    </rim:ExternalIdentifier>

    <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType =
"ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>

    <!--
      The following show submission of a brand new classification scheme in its entirety
      -->
    <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
      <rim:Name>
        <rim:LocalizedString value = "Geography"/>
      </rim:Name>

      <rim:Description>
        <rim:LocalizedString value = "This is a sample Geography scheme"/>
      </rim:Description>

      <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
"NorthAmerica" >
```



```
        <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
"UnitedStates" />
        <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
      </rim:ClassificationNode>

      <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
        <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
          <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
        </rim:ClassificationNode>
      </rim:ClassificationNode>
    </rim:ClassificationScheme>


    <!--
      The following show submission of a Automotive sub-tree of ClassificationNodes that
      gets added to an existing classification scheme named 'Industry'
      that is already in the registry
      -->

    <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
    <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
123456789012">
      <rim:Name>
        <rim:LocalizedString value = "Automotive" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
      </rim:Description>
    </rim:ClassificationNode>

    <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
      <rim:Name>
        <rim:LocalizedString value = "Parts Supplier" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
      </rim:Description>
    </rim:ClassificationNode>

    <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
      <rim:Name>
        <rim:LocalizedString value = "Engine Supplier" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
      </rim:Description>
    </rim:ClassificationNode>

    <!--
      The following show submission of 2 Classifications of an object  that is already in
      the registry using 2 ClassificationNodes. One ClassificationNode
      is being submitted in this request (Japan) while the other is already in the registry.
      -->

    <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
1234-123456789012" classificationNode = "Japan-id">
      <rim:Description>
        <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
      </rim:Description>
    </rim:Classification>

    <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
"urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
123456789012">
      <rim:Description>
        <rim:LocalizedString value = "Classifies object using a node in the registry" />
      </rim:Description>
    </rim:Classification>

    <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
  </rim:LeafRegistryObjectList>
</SubmitObjectsRequest>
```


7.4 The Update Objects Protocol

This section describes the protocol of the Registry Service that allows a Registry Client to update one or more existing RegistryObjects and/or repository items in the registry on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

Figure 17: Update Objects Sequence Diagram

7.4.1 UpdateObjectsRequest

The UpdateObjectsRequest is used by a client to update RegistryObjects and/or repository items that already exist within the registry.

Syntax:

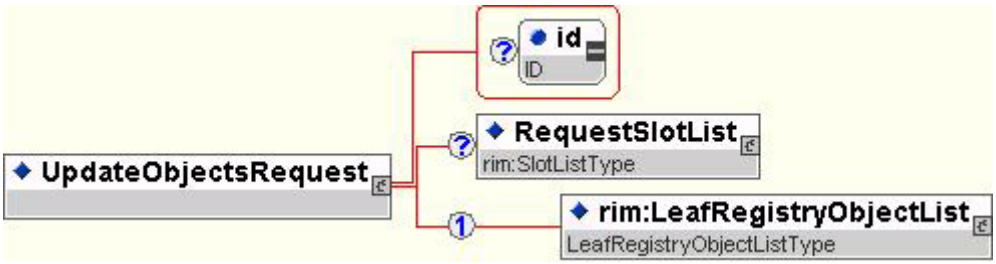


Figure 18: UpdateObjectsRequest Syntax

Parameters:

- *LeafRegistryObjectsList:* This parameter specifies a collection of RegistryObject instances that are being updated within the registry. All immediate RegistryObject children of the LeafRegistryObjectList must be a current RegistryObject already in the registry. RegistryObjects must include all required attributes, even those the

user does not intend to change. A missing attribute is interpreted as a request to set that attribute to NULL or in case it has a default value, the default value will be assumed. If this collection contains an immediate child RegistryObject that does not already exists in the registry, then the registry must return an InvalidRequestException. If the user wishes to submit a mix of new and updated objects then she should use a SubmitObjectsRequest.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException:* Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException:* Indicates that the requestor referenced an object within the request that was not found.
- *ObjectExistsException:* Indicates that the requestor tried to submit an object using an id that matched the id of an existing object in the registry.
- *InvalidRequestException:* Indicates that the requestor attempted to perform an operation which was semantically invalid.
- *UnsupportedCapabilityException:* Indicates that the requestor attempted to submit some content that is not supported by the registry.
- *QuotaExceededException:* Indicates that the requestor attempted to submit more content than the quota allowed for them by the registry.

7.4.2 Audit Trail

The RS must create AuditableEvents object with eventType *Updated* for each RegistryObject updated via an UpdateObjectsRequest.

7.5 The Add Slots Protocol

This section describes the protocol of the Registry Service that allows a client to add slots to a previously submitted RegistryObject using the LifeCycleManager. Slots provide a dynamic mechanism for extending RegistryObjects as defined by [ebRIM].

Figure 19: Add Slots Sequence Diagram

7.5.1 AddSlotsRequest

The AddSlotsRequest is used by a client to add slots to an existing RegistryObject in the registry.

Syntax:

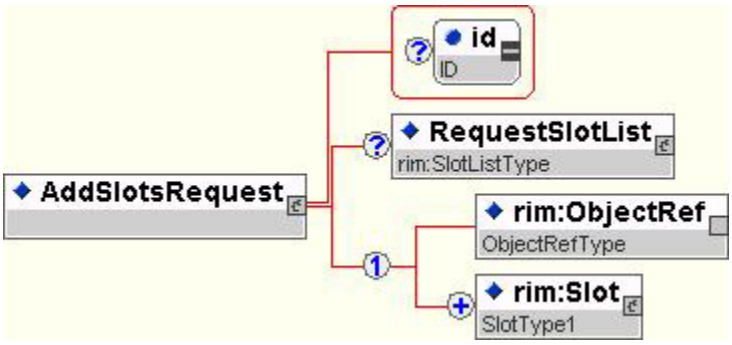


Figure 20: AddSlotsRequest Syntax

Parameters:

- *ObjectRef:* This parameter specifies a reference to a RegistryObject instance to which the requestor wishes to add slots via this request.
- *Slot:* This parameter specifies one or more Slot objects. Each Slot contains a ValueList with one or more Values. Each Slot also has a slot name and a slotType as described [ebRIM].

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *SlotExistsException:* Indicates that the requestor tried to add a Slot using a name that matched the name of an existing Slot in the RegistryObject.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation which was semantically invalid.

7.6 The Remove Slots Protocol

This section describes the protocol of the Registry Service that allows a client to remove slots to a previously submitted RegistryObject using the LifeCycleManager.

Figure 21: Remove Slots Sequence Diagram

7.6.1 RemoveSlotsRequest

The RemoveSlotsRequest is used by a client to remove slots from an existing RegistryObject in the registry.

Syntax:

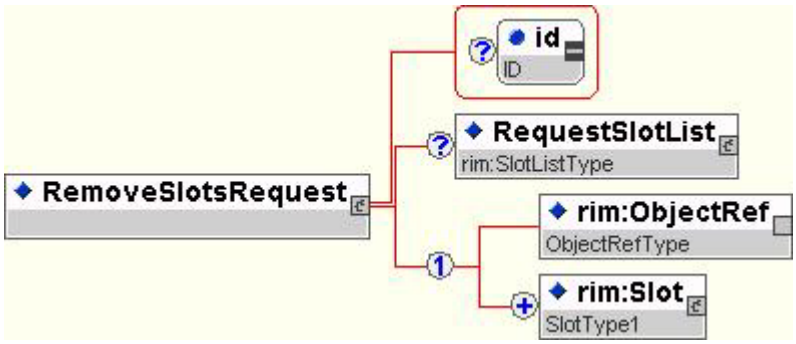


Figure 22: RemoveSlotsRequest Syntax

Parameters:

- *ObjectRef:* This parameter specifies a reference to a RegistryObject instance from which the requestor wishes to remove slots via this request.
- *Slot:* This parameter specifies one or more Slot objects. Each slot being removed is identified by its name attribute. Any Values specified with the ValueList for the Slot can be silently ignored.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *SlotNotFoundException*: Indicates that the requestor attempted to remove a Slot by name where no Slot existed that matches the specified name.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation which was semantically invalid.

7.7 The Approve Objects Protocol

This section describes the protocol of the Registry Service that allows a client to approve one or more previously submitted RegistryObject objects using the LifeCycleManager.

Figure 23: Approve Objects Sequence Diagram

7.7.1 ApproveObjectsRequest

The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject instances in the registry.

Syntax:

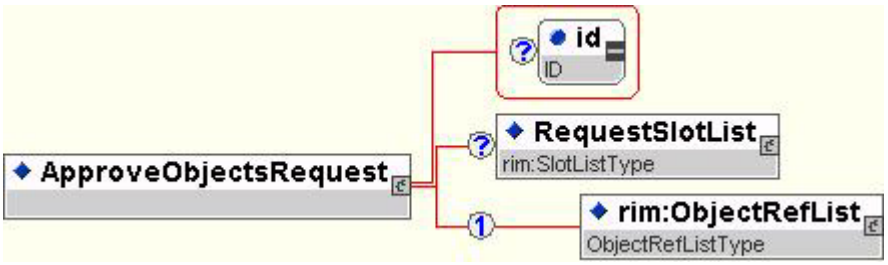


Figure 24: ApproveObjectsRequest Syntax

Parameters:

- *ObjectRefList:* This parameter specifies a collection of reference to existing RegistryObject instances in the registry. These are the objects that the requestor wishes to approve via this request.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an

operation for which she was not authorized.

- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation that was semantically invalid.

7.7.2 Audit Trail

The RS must create AuditableEvent objects with eventType *Approved* for each RegistryObject instance approved via an ApproveObjectsRequest.

7.8 The Deprecate Objects Protocol

This section describes the protocol of the Registry Service that allows a client to deprecate one or more previously submitted RegistryObject instances using the LifeCycleManager. Once a RegistryObject is deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that object can be submitted. However, existing references to a deprecated object continue to function normally.

Figure 25: Deprecate Objects Sequence Diagram

7.8.1 DeprecateObjectsRequest

The DeprecateObjectsRequest is used by a client to deprecate one or more existing RegistryObject instances in the registry.

Syntax:

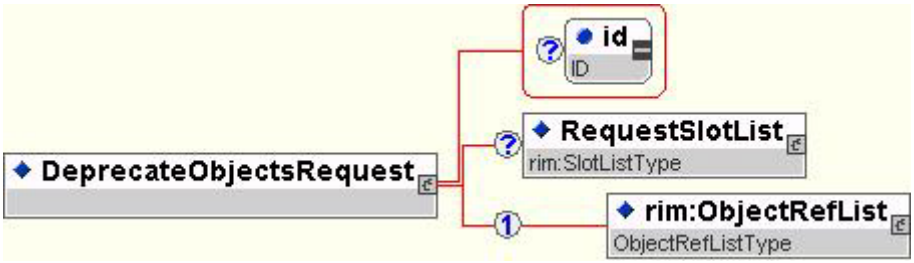


Figure 26: DeprecateObjectsRequest Syntax

Parameters:

- *ObjectRefList:* This parameter specifies a collection of reference to existing RegistryObject instances in the registry. These are the objects that the requestor wishes to deprecate via this request.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation which was semantically invalid.

7.8.2 Audit Trail

The RS must create AuditableEvents object with eventType *Deprecated* for each RegistryObject deprecated via a DeprecateObjectsRequest.

7.9 The Undeprecate Objects Protocol

This section describes the protocol of the Registry Service that allows a client to undo the deprecation of one or more previously deprectaed RegistryObject instances using the LifeCycleManager. When a RegistryObject is un-deprecated, it goes back to the Submitted status and new references (e.g. new Associations, Classifications and ExternalLinks) to that object can be submitted.

Figure 27: Undeprecate Objects Sequence Diagram

7.9.1 UndeprecateObjectsRequest

The UndeprecateObjectsRequest is used by a client to un-deprecate one or more existing RegistryObject instances in the registry.

Syntax:

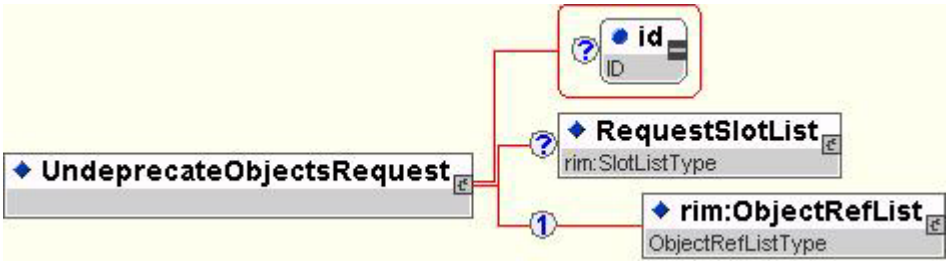


Figure 28: UndeprecateObjectsRequest Syntax

Parameters:

- *ObjectRefList:* This parameter specifies a collection of reference to existing RegistryObject instances in the registry. These are the objects that the requestor wishes to un-deprecate via this request. The registry should silently ignore any reference to a RegistryObject that is not deprecated.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an

operation for which she was not authorized.

- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation which was semantically invalid.

7.9.2 Audit Trail

The RS must create AuditableEvents object with eventType *Undeprecated* for each RegistryObject undeprecated via an UndeprecateObjectsRequest.

7.10 The Remove Objects Protocol

This section describes the protocol of the Registry Service that allows a client to remove one or more RegistryObject instances and/or repository items using the LifeCycleManager.

The remove object protocol is expressed in UML notation as described in Appendix C.

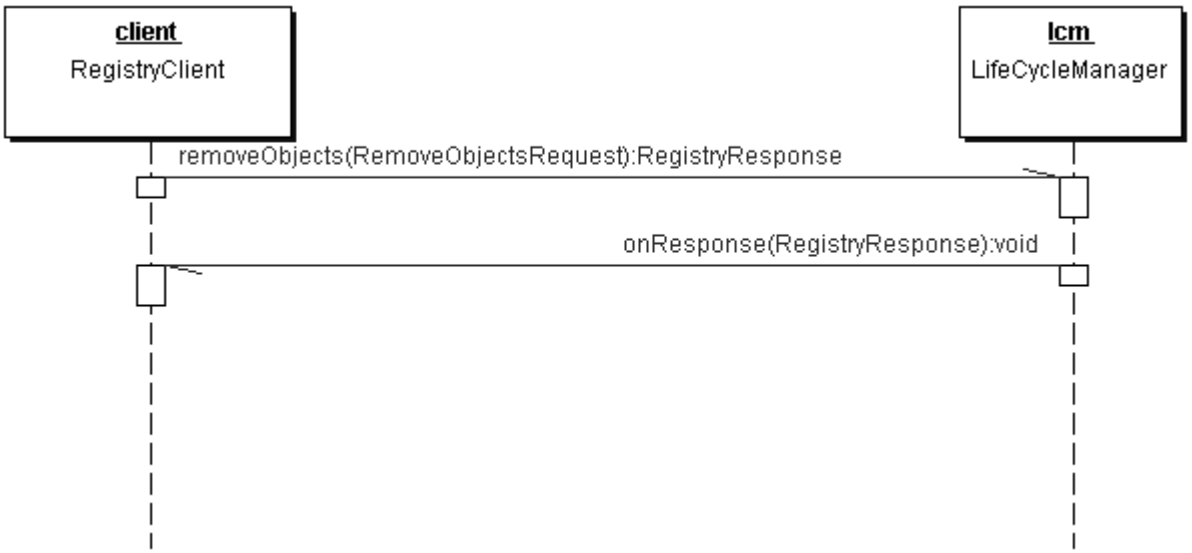


Figure 29: Remove Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix B.

7.10.1 RemoveObjectsRequest

The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject and/or repository items from the registry.

Syntax:

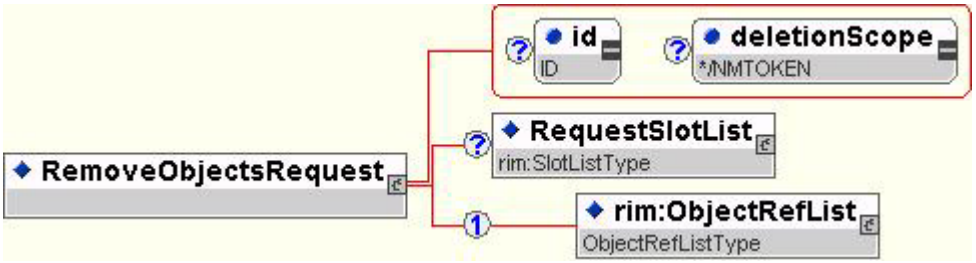


Figure 30: RemoveObjectsRequest Syntax

Parameters:

- *deletionScope:* This parameter indicates the scope of impact of the RemoveObjectsRequest. Its valid values may be as follows:

  DeleteRepositoryItemOnly: This deletionScope specifies that the request should delete the repository items for the specified registry entries but not delete the specified registry entries. This is useful in keeping references to the registry entries valid. A registry *must* set the status of the ExtrinsicObject instance to *Withdrawn* in this case.

  DeleteAll: This deletionScope specifies that the request should delete both the RegistryObject and the repository item for the specified registry entries. Only if all references (e.g. Associations, Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject then be removed using a RemoveObjectsRequest with deletionScope DeleteAll.

- *ObjectRefList:* This parameter specifies a collection of reference to existing RegistryObject instances in the registry. These are the objects that the requestor wishes to remove via this request.

Returns:

This request returns a RegistryResponse. See section 7.3.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *ObjectNotFoundException*: Indicates that the requestor referenced an object within the request that was not found.
- *InvalidRequestException*: Indicates that the requestor attempted to perform an operation that was semantically invalid. A specific case where this exception must be returned is if the requestor attempts to specify deletionScope of *DeleteRepositoryItemOnly* when the object is not an instance of an ExtrinsicObject.

1636 *ReferencesExistException*: Indicates that the requestor attempted to remove a RegistryObject
1637 while references to it still exist.

8 Query Management Service

This section describes the capabilities of the Registry Service that allow a client (QueryManagerClient) to search for or query different kinds of registry objects in the ebXML Registry using the QueryManager interface of the Registry. The Registry supports the following query capabilities:

- Filter Query
- SQL Query

The Filter Query mechanism in Section 8.2 *must* be supported by every Registry implementation. The SQL Query mechanism is an optional feature and *may* be provided by a registry implementation. However, if a vendor provides an SQL query capability to an ebXML Registry it *must* conform to this document. As such this capability is a normative yet optional capability.

In a future version of this specification, the W3C XQuery syntax may be considered as another query syntax.

8.1 Ad Hoc Query Request/Response

A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a query in one of the supported Registry query mechanisms.

The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously back to the client. The AdhocQueryResponse returns a collection of objects whose element type depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate classes in [ebRIM] such as RegistryObject and RegistryEntry.

Any errors in the query request messages are indicated in the corresponding AdhocQueryResponse message.

Figure 31: Submit Ad Hoc Query Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix

Error! Reference source not found..

8.1.1 AdhocQueryRequest

The AdhocQueryRequest is used to submit queries to the registry.

Syntax:

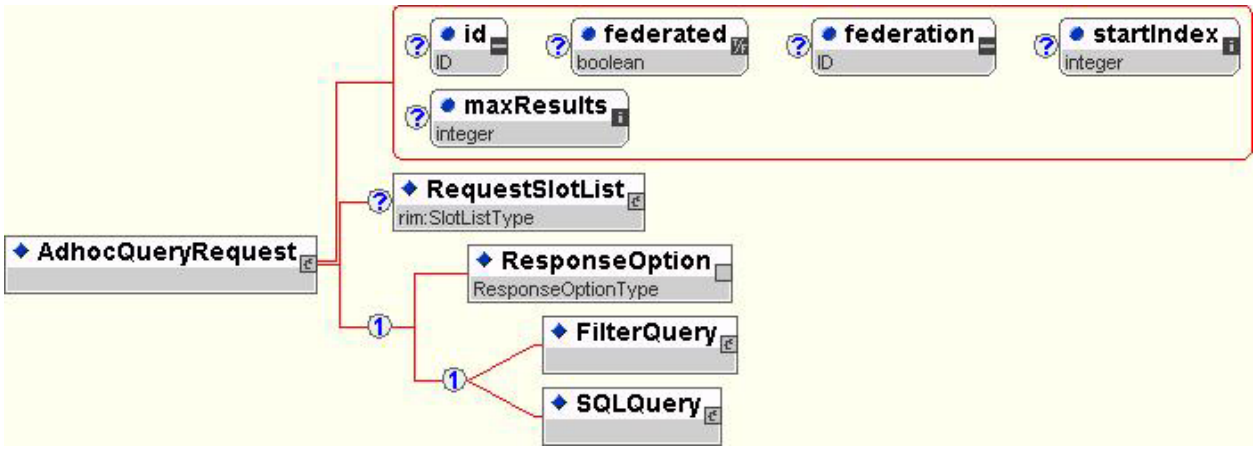


Figure 32: AdhocQueryRequest Syntax

Parameters:

- *federated:* This optional parameter specifies that the registry must process this query as a federated query. By default its value is *false*.
- *federation:* This optional parameter specifies the id of the target Federation for a federated query in case the registry is a member of multiplke federations. In the absence of this parameter a registry must route the federated query to all federations that it is a member of.
- *FilterQuery:* This parameter specifies a registry Filter Query.
- *maxResults*: This optional parameter specifies a limit on the maximum number of results the client wishes the query to return. If unspecified, the registry should return either all the results, or in case the result set size exceeds a registry specific limit, the registry should return a sub-set of results that are within the bounds of the registry specific limit. See section 0 for an illustrative example.
- *ResponseOption*: This required parameter allows the client to control the format and content of the AdhocQueryResponse to this request. See section 8.1.3 for details.
- *SQLQuery***:** This parameter specifies a registry SQL Query.
- *startIndex*: This optional integer value is used to indicate which result *must* be returned as the first result when iterating over a large result set. The default value is 0, which returns the result set starting with index 0 (first result). See section 0 for an illustrative example.

Returns:

This request returns an AdhocQueryResponse. See section 8.1.2 for details.

Exceptions:

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

- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
- *InvalidQueryException:* signifies that the query syntax or semantics was invalid. Client must fix the query syntax or semantic and re-submit the query.

8.1.2 AdhocQueryResponse

The AdhocQueryResponse is sent by the registry as a response to AdhocQueryRequest.

Syntax:

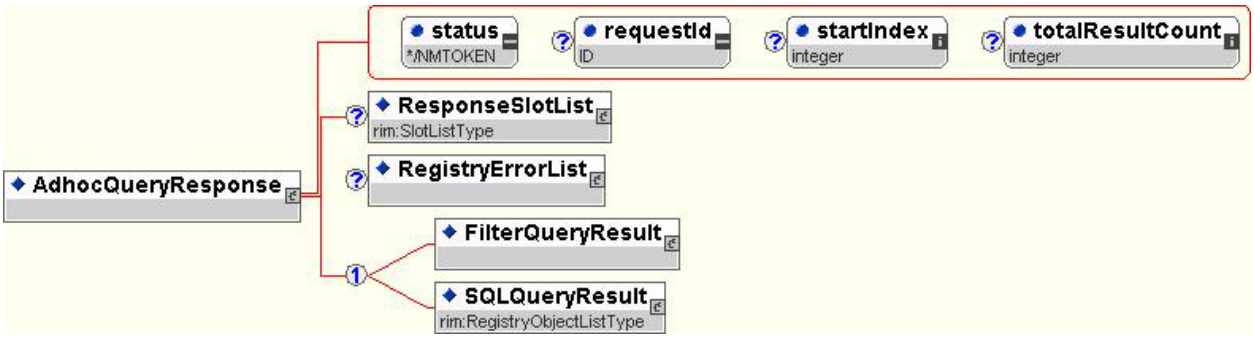


Figure 33: AdhocQueryResponse Syntax

Parameters:

- *FilterQueryResult:* This parameter specifies the result of a registry Filter Query.
- *SQLQueryResult***:** This parameter specifies the result of a registry SQL Query.
- *startIndex*: This optional integer value is used to indicate the index for the first result in the result set returned by the query, within the complete result set matching the query. By default, this value is 0. See section 0 for an illustrative example.
- *totalResultCount*: This optional parameter specifies the size of the complete result set matching the query within the registry. When this value is unspecified, the client should assume that value is the size of the result set contained within the result. See section 0 for an illustrative example.

8.1.3 ReponseOption

A client specifies an ResponseOption structure within an AdhocQueryRequest to indicate the

format of the results within the corresponding AdhocQueryResponse.

Syntax:

Figure 34: ResponseOption Syntax

Parameters:

- *returnComposedObjects:* This optional parameter specifies whether the RegistryObjects returned should include composed objects as defined by Figure 1 in [ebRIM]. The default is to return all composed objects.
- *returnType:* This optional enumeration parameter specifies the type of RegistryObject to return within the response. Values for returnType are as follows:

  - ObjectRef - This option specifies that the AdhocQueryResponse may contain a collection of ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return references to registry objects.
  - RegistryObject - This option specifies that the AdhocQueryResponse may contain a collection of RegistryObject XML elements as defined in [ebRIM Schema]. In this case all attributes of the registry objects are returned (objectType, name, description, …).
  - RegistryEntry - This option specifies that the AdhocQueryResponse may contain a collection of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which correspond to RegistryEntry or RegistryObject attributes.
  - LeafClass - This option specifies that the AdhocQueryResponse may contain a collection of XML elements that correspond to leaf classes as defined in [ebRIM Schema].
  - LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse may contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema] accompanied with their repository items or RegistryEntry or RegistryObject and their attributes. Linking of ExtrinsicObject and its repository item is accomplished using the technique explained in Section 8.4 -Content Retrieval.

  If "returnType" specified does not match a result returned by the query, then the registry *must* use the closest matching semantically valid returnType that matches the result.

  This can be illustrated with a case when OrganizationQuery is asked to return LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass option instead. If OrganizationQuery is asked to retrieve a

RegistryEntry as a return type then RegistryObject metadata will be returned.

8.1.4 Iterative Query Support

The iterative query feature is a normative optional feature of the registry. The AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large result set matching a logical query by allowing multiple AdhocQueryRequest requests to be submitted such that each query requests a different sliding window within the result set. This feature enables the registry to handle queries that match a very large result set, in a scalable manner.

The iterative queries feature is not a true Cursor capability as found in databases. The registry is not required to maintain transactional consistency or state between iterations of a query. Thus it is possible for new objects to be added or existing objects to be removed from the complete result set in between iterations. As a consequence it is possible to have a result set element be skipped or duplicated between iterations.

Note that while it is not required, it may be possible for implementations to be smart and implement a transactionally consistent iterative query feature. It is likely that a future version of this specification will require a transactionally consistent iterative query capability.

Query Iteration Example

Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a query that matches all 1007 Organizations. The user wishes to do the query iteratively such that Organizations are retrieved in chunks of 100. The following table illustrates the parameters of the AdhocQueryRequest and those of the AdhocQueryResponses for each iterative query in this example.

AdhocQueryRequest Parameters		AdhocQueryResponse Parameters		
startIndex	maxResults	startIndex	totalResultCount	# of Results
0	100	0	1007	100
100	100	100	1007	100
200	100	200	1007	100
300	100	300	1007	100
400	100	400	1007	100
500	100	500	1007	100
600	100	600	1007	100
700	100	700	1007	100
800	100	800	1007	100
900	100	900	1007	100
1000	100	1000	1007	7

8.2 Filter Query Support

FilterQuery is an XML syntax that provides simple query capabilities for any ebXML conforming Registry implementation. Each query alternative is directed against a single class

defined by the ebXML Registry Information Model (ebRIM). There are two types of filter queries depending on which classes are queried on.

- Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic queries that might return different subclasses of the class that is queried on. The result of such a query is a set of XML elements that correspond to instances of any class that satisfies the responseOption defined previously in Section 8.1.3. An example might be that RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances that satisfy the query. This implies that response might return XML elements that correspond to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact traversals of these classes. Responses to these queries are accordingly constrained.

A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.1.

Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated with Z. The ebRIM Binding for C might be as in Figure 35

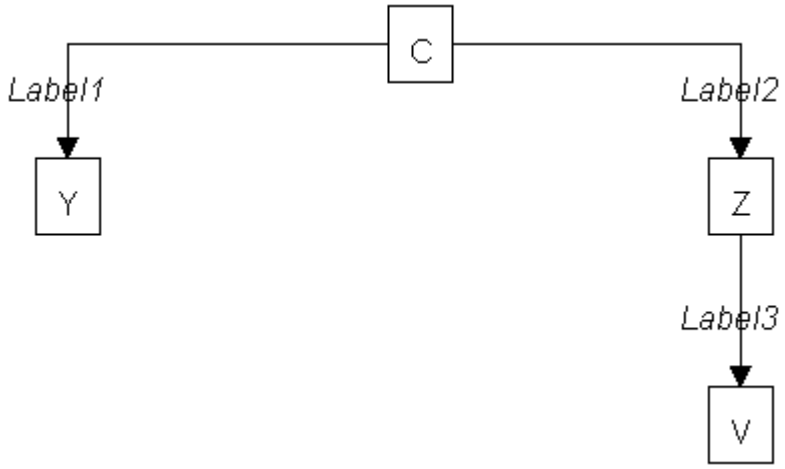


Figure 35: Example ebRIM Binding

Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to which ebRIM association is intended. The name of the query is determined by the root class, i.e. this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are limited to instances that are linked to their parent node by the identified association.

Each FilterQuery alternative depends upon one or more class filters, where a class filter is a restricted predicate clause over the attributes of a single class. Class methods that are defined in ebRIM and that return simple types constitute “visible attributes” that are valid choices for predicate clauses. Names of those attributes will be same as name of the corresponding method just without the prefix ‘get’. For example, in case of “getLevelNumber” method the

corresponding visible attribute is “levelNumber”. The supported class filters are specified in Section 8.2.17 and the supported predicate clauses are defined in Section 8.2.18. A FilterQuery will be composed of elements that traverse the tree to determine which branches satisfy the designated class filters, and the query result will be the set of instances that support such a branch.

In the above example, the CQuery element will have three subelements, one a CFilter on the C class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on the Y class to eliminate branches from C to Y where the target of the association does not satisfy the YFilter, and a third to eliminate branches along a path from C through Z to V. The third element is called a branch element because it allows class filters on each class along the path from C to V. In general, a branch element will have subelements that are themselves class filters, other branch elements, or a full-blown query on the class in the path.

If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one branch, filter or query element on Y is allowed. However, if the association is one-to-many, then multiple branch, filter or query elements are allowed. This allows one to specify that an instance of C must have associations with multiple instances of Y before the instance of C is said to satisfy the branch element.

The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their respective superclass queries. Structures of XML elements that match the ebRIM classes are explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in ebRIM whenever possible.

The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify the effect of that binding on query semantics.

8.2.1 FilterQuery

Purpose

To identify a set of queries that traverse specific registry class. Each alternative assumes a specific binding to ebRIM. The status is a success indication or a collection of warnings and/or exceptions.

Definition

```
<complexType name="FilterQueryType">
  <complexContent>
    <extension base="rim:AdhocQueryType">
      <choice minOccurs="1" maxOccurs="1">
        <element ref="tns:RegistryObjectQuery"/>
        <element ref="tns:RegistryEntryQuery"/>
        <element ref="tns:AssociationQuery"/>
        <element ref="tns:AuditableEventQuery"/>
        <element ref="tns:ClassificationQuery"/>
        <element ref="tns:ClassificationNodeQuery"/>
        <element ref="tns:ClassificationSchemeQuery"/>
```



```
            <element ref="tns:RegistryPackageQuery"/>
            <element ref="tns:ExtrinsicObjectQuery"/>
            <element ref="tns:OrganizationQuery"/>
            <element ref="tns:ServiceQuery"/>
            <element ref="tns:FederationQuery"/>
            <element ref="tns:RegistryQuery"/>
            <element ref="tns:SubscriptionQuery"/>
            <element ref="tns:UserQuery"/>
          </choice>
      </extension>
  </complexContent>
</complexType>
<element name="FilterQuery" type="tns:FilterQueryType"/>

<element name="FilterQueryResult">
          <complexType>
              <choice minOccurs="1" maxOccurs="1">
                  <element ref="tns:RegistryObjectQueryResult" />
                  <element ref="tns:RegistryEntryQueryResult" />
                  <element ref=”tns:AssociationQueryResult” />
                  <element ref="tns:AuditableEventQueryResult" />
                  <element ref=”tns:ClassificationQueryResult” />
                  <element ref="tns:ClassificationNodeQueryResult" />
                  <element ref="tns:ClassificationSchemeQueryResult" />
                  <element ref="tns:RegistryPackageQueryResult" />
                  <element ref="tns:ExtrinsicObjectQueryResult" />
                  <element ref="tns:OrganizationQueryResult" />
                  <element ref="tns:ServiceQueryResult" />
                  <element ref="tns:FederationQueryResult" />
                  <element ref="tns:RegistryQueryResult" />
                  <element ref="tns:SubscriptionQueryResult" />
                  <element ref="tns:UserQueryResult" />
              </choice>
          </complexType>
</element>
```

Semantic Rules

1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
2. Semantic rules specify the procedure for implementing the evaluation of Filter Queries. Implementations do not necessarily have to follow the same procedure provided that the same effect is achieved.
3. Each FilterQueryResult is a set of XML elements to identify each instance of the result set. Each XML attribute carries a value derived from the value of an attribute specified in the Registry Information Model [ebRIM Schema].
4. For each FilterQuery subelement there is only one corresponding FilterQueryResult subelement that must be returned as a response. Class name of the FilterQueryResult subelement has to match the class name of the FilterQuery subelement.
5. If a Branch or Query element for a class has no sub-elements then every persistent instance of that class satisfies the Branch or Query.

6. If an error condition is raised during any part of the execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to “failure” and no AdHocQueryResult element is returned; instead, a RegistryErrorList element must be returned with its highestSeverity element set to “error”. At least one of the RegistryError elements in the RegistryErrorList will have its severity attribute set to “error”.
7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to “success” and an appropriate FilterQueryResult element must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of the RegistryErrorList is set to “warning” and the serverity attribute of each RegistryError is set to “warning”.

8.2.2 RegistryObjectQuery

Purpose

To identify a set of registry object instances as the result of a query over selected registry metadata.

ebRIM Binding

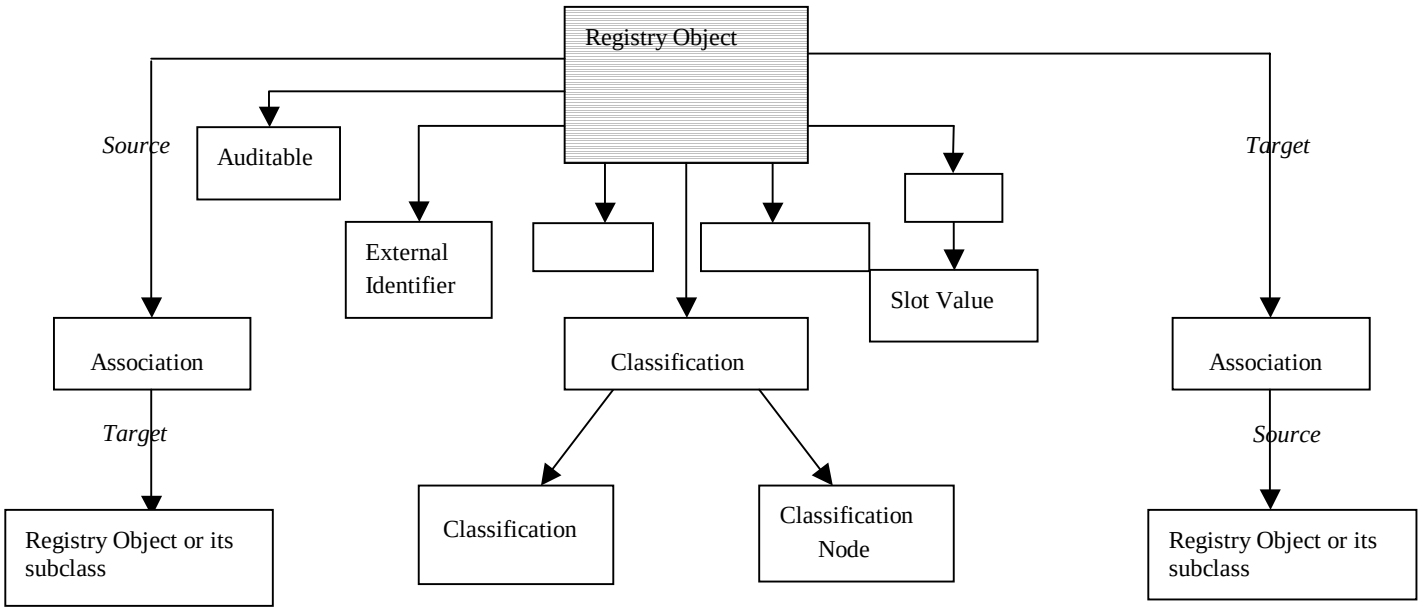


Figure 37: ebRIM Binding for RegistryObjectQuery

Definition

```
<complexType name="RegistryObjectQueryType">
  <sequence>
    <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
    <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
    <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
    <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
```



```
      <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
      <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
</complexType>
<element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />

<element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />

<complexType name="InternationalStringBranchType">
    <sequence>
      <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
</complexType>

<complexType name="AssociationBranchType">
    <sequence>
      <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
      <choice minOccurs="0" maxOccurs="1">
        <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
        <element ref=”tns:AssociationQuery” minOccurs=”0” maxOccurs=”1” />
        <element ref=”tns:ClassificationQuery” minOccurs=”0” maxOccurs=”1” />
        <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:FederationQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:SubscriptionQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:UserQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
        <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
      </choice>
    </sequence>
</complexType>
<element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
<element name="TargetAssociationBranch" type="tns:AssociationBranchType" />

<element name="ClassifiedByBranch">
    <complexType>
      <sequence>
        <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
      </sequence>
    </complexType>
</element>

<element name="SlotBranch">
    <complexType>
      <sequence>
        <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
```

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

<complexType name="ServiceBindingBranchType">
    <sequence>
      <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
      <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
    </sequence>
</complexType>
<element name=”ServiceBindingBranch” type=”tns:ServiceBindingBranchType” />
<element name=”ServiceBindingTargetBranch” type=”tns:ServiceBindingBranchType” />

<element name="SpecificationLinkBranch">
    <complexType>
      <sequence>
        <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
      </sequence>
    </complexType>
</element>
```

Semantic Rules

1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The following steps will eliminate instances in RO that do not satisfy the conditions of the specified filters.
   a) If RO is empty then go to number 2 below.
   b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from RO. If RO is empty then continue to the next numbered rule.
   c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one ExternalIdentifier instance, then remove x from RO; otherwise, treat each ExternalIdentifierFilter element separately as follows: Let EI be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
   d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x doesn’t have an auditable event that satisfy AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is empty then continue to the next numbered rule.

e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a name then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise treat NameBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by all of the LocalizedStrings that constitute the name of the registry object then remove x from RO. If RO is empty then continue to the next numbered rule.

f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a description then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise treat DescriptionBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by all of the LocalizedStrings that constitute the description of the registry object then remove x from RO. If RO is empty then continue to the next numbered rule.

g) If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the classifiedObject of at least one Classification instance, then remove x from RO; otherwise, treat each ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified within the ClassifiedByBranch, then let CL be the set of all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then remove x from RO and continue to the next numbered rule. Otherwise, if CL is not empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of remaining Classification instances in CL whose defining classification scheme satisfies the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and continue to the next numbered rule. Otherwise, if CL remains not empty, and if a ClassificationNodeQuery is specified, then replace CL by the set of remaining Classification instances in CL for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one Slot instance, then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise, treat each SlotBranch element separately as follows: If a SlotFilter is not specified within the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty, then remove x from RO and continue to the next numbered rule. Otherwise, if SL remains not empty, and if a SlotValueFilter is specified, replace SL by the set of remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the source object of at least one Association instance, then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise, treat each SourceAssociationBranch element separately as follows:

If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be the set of all Association instances that have x as a source object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the source object. If AF is empty, then remove x from RO.

If RO is empty then continue to the next numbered rule.

If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let ROT be the set of ClassificationScheme instances that satisfy the ClassificationSchemeQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Organization instances that satisfy the OrganizationQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AuditableEventQuery is specified within the SourceAssociationBranch, then let ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the set of Service instances that satisfy the ServiceQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Classification instances that satisfy the ClassificationQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2).

If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let ROT be the set of ServiceBinding instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a SpecificationLinkBranch is specified within the ServiceBindingBranch then consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in ROT. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for at least one registry object in RO, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery element is

specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and are the target service binding of some element of ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let ROT be the set of SpecificationLink instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Association instances that satisfy the AssociationQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2).

If a Federation Query is specified within the SourceAssociationBranch, then let ROS be the set of Federation instances that satisfy the FederationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryQuery is specified within the SourceAssociationBranch, then let ROS be the set of Registry instances that satisfy the RegistryQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a SubscriptionQuery is specified within the SourceAssociationBranch, then let ROS be the set of Subscription instances that satisfy the SubscriptionQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a UserQuery is specified within the SourceAssociationBranch, then let ROS be the set of User instances that satisfy the UserQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

j) If a TargetAssociationBranch element is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the target object of some Association instance, then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise, treat each TargetAssociationBranch element separately as follows:

If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be the set of all Association instances that have x as a target object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the target object. If AF is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS be the set of

RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let ROS be the set of ClassificationScheme instances that satisfy the ClassificationSchemeQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Organization instances that satisfy the OrganizationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AuditableEventQuery is specified within the TargetAssociationBranch, then let ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the set of Service instances that satisfy the ServiceQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Classification instances that satisfy the ClassificationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2).

If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS be the set of ServiceBinding instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a SpecificationLinkBranch is specified within the ServiceBindingBranch then consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in ROS. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule.

If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let ROS be the set of SpecificationLink instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and are the target service binding of some element of ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Association instances that satisfy the AssociationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2).

If a Federation Query is specified within the TargetAssociationBranch, then let ROS be the set of Federation instances that satisfy the FederationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a RegistryQuery is specified within the TargetAssociationBranch, then let ROS be the set of Registry instances that satisfy the RegistryQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a SubscriptionQuery is specified within the TargetAssociationBranch, then let ROS be the set of Subscription instances that satisfy the SubscriptionQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a UserQuery is specified within the TargetAssociationBranch, then let ROS be the set of User instances that satisfy the UserQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set RO to be the result of the RegistryObjectQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application needs all items that are classified by two different classification schemes, one based on "Industry" and another based on "Geography". Both schemes have been defined by ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The following query identifies registry entries for all registered items that are classified by Industry as any subnode of "Automotive" and by Geography as any subnode of "Asia/Japan".

```
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryEntry"/>
  <FilterQuery>
    <RegistryObjectQuery>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "path">
              <StringClause stringPredicate = "Equal">//Automotive</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassifiedByBranch>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "path">
              <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
```



```
                    <SimpleClause leftArgument = "value">
                      <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
                    </SimpleClause>
                  </Clause>
                </LocalizedStringFilter>
              </NameBranch>
            </ClassificationSchemeQuery>
          </ClassifiedByBranch>
        </RegistryObjectQuery>
      </FilterQuery>
</AdhocQueryRequest>
```

A client application wishes to identify all RegistryObject instances that are classified by some internal classification scheme and have some given keyword as part of the description of one of the classification nodes of that classification scheme. The following query identifies all such RegistryObject instances. The query takes advantage of the knowledge that the classification scheme is internal, and thus that all of its nodes are fully described as ClassificationNode instances.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryObject"/>
  <FilterQuery>
    <RegistryObjectQuery>
      <ClassifiedByBranch>
        <ClassificationNodeQuery>
          <DescriptionBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">transistor</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </DescriptionBranch>
        </ClassificationNodeQuery>
      </ClassifiedByBranch>
    </RegistryObjectQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.3 RegistryEntryQuery

Purpose

To identify a set of registry entry instances as the result of a query over selected registry metadata.

ebRIM Binding

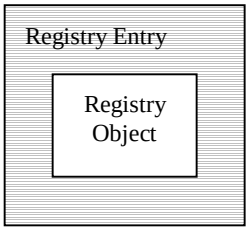


Figure 38: ebRIM Binding for RegistryEntryQuery

Definition

```
<complexType name="RegistryEntryQueryType">
    <complexContent>
        <extension base="tns:RegistryObjectQueryType">
            <sequence>
                <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
            </sequence>
        </extension>
    </complexContent>
</complexType>
<element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />

<element name="RegistryEntryQueryResult">
    <complexType>
        <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="rim:ObjectRef" />
            <element ref="rim:ClassificationScheme" />
            <element ref="rim:ExtrinsicObject" />
            <element ref="rim:RegistryEntry" />
            <element ref="rim:RegistryObject" />
            <element ref="rim:RegistryPackage" />
            <element ref="rim:Service" />
            <element ref="rim:Federation" />
            <element ref="rim:Registry" />
        </choice>
    </complexType>
</element>
```

Semantic Rules

1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
    a) If RE is empty then continue to the next numbered rule.
    b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE. If RE is empty then continue to the next numbered rule.
    c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited RegistryObjectQuery over RE as explained in Section 8.2.2.
2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, set RE to be the result of the RegistryEntryQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client wishes to establish a trading relationship with XYZ Corporation and wants to know if they have registered any of their business documents in the Registry. The following query returns a set of registry entry identifiers for currently registered items submitted by any organization whose name includes the string "XYZ". It does not return any registry entry identifiers for superseded, replaced, deprecated, or withdrawn items.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "ObjectRef"/>
  <FilterQuery>
    <RegistryEntryQuery>
      <TargetAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument = "associationType">
              <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Contains">XYZ</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </OrganizationQuery>
      </TargetAssociationBranch>
      <RegistryEntryFilter>
        <Clause>
          <SimpleClause leftArgument = "status">
            <StringClause stringPredicate = "Equal">Approved</StringClause>
          </SimpleClause>
        </Clause>
      </RegistryEntryFilter>
    </RegistryEntryQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client is using the United Nations Standard Product and Services Classification (UNSPSC) scheme and wants to identify all companies that deal with products classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.


```
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryEntry"/>
  <FilterQuery>
    <RegistryEntryQuery>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "nodeRepresentation">
              <StringClause stringPredicate = "Equal">321118</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassifiedByBranch>
      <RegistryEntryFilter>
        <Clause>
          <CompoundClause connectivePredicate = "And">
            <Clause>
              <SimpleClause leftArgument = "objectType">
                <StringClause stringPredicate = "Equal">CPP</StringClause>
              </SimpleClause>
            </Clause>
            <Clause>
              <SimpleClause leftArgument = "status">
                <StringClause stringPredicate = "Equal">Approved</StringClause>
              </SimpleClause>
            </Clause>
          </CompoundClause>
        </Clause>
      </RegistryEntryFilter>
    </RegistryEntryQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.4 AssociationQuery

Purpose

To identify a set of association instances as the result of a query over selected registry metadata.

ebRIM Binding

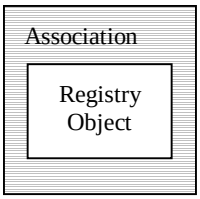


Figure 39: ebRIM Binding for AssociationQuery

Definition

```
<complexType name = "AssociationQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
      <sequence>
        <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name = "AssociationQuery" type = "tns:AssociationQueryType"/>

<element name="AssociationQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:Association" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let A denote the set of all persistent Association instances in the Registry. The following steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
   a) If A is empty then continue to the next numbered rule.
   b) If an AssociationFilter element is not directly contained in the AssociationQuery element, then go to the next step; otherwise let x be an association instance in A. If x does not satisfy the AssociationFilter then remove x from A. If A is empty then continue to the next numbered rule.
   c) Let A be the set of remaining Association instances. Evaluate inherited RegistryObjectQuery over A as explained in Section 8.2.2.
2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to be the result of the AssociationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify a set of associations that are ‘equivalentTo’ a set of other associations.


```
<AdhocQueryRequest">
  <ResponseOption returnType="LeafClass" />
  <FilterQuery>
    <AssociationQuery>
      <SourceAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument="associationType">
              <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <AssociationQuery>
          <AssociationFilter>
            <Clause>
              <SimpleClause leftArgument="associationType">
                <StringClause stringPredicate="StartsWith">Sin</StringClause>
              </SimpleClause>
            </Clause>
          </AssociationFilter>
        </AssociationQuery>
      </SourceAssociationBranch>
      <AssociationFilter>
        <Clause>
          <SimpleClause leftArgument="associationType">
            <StringClause stringPredicate="StartsWith">Son</StringClause>
          </SimpleClause>
        </Clause>
      </AssociationFilter>
    </AssociationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```


8.2.5 AuditableEventQuery

Purpose

To identify a set of auditable event instances as the result of a query over selected registry metadata.

ebRIM Binding

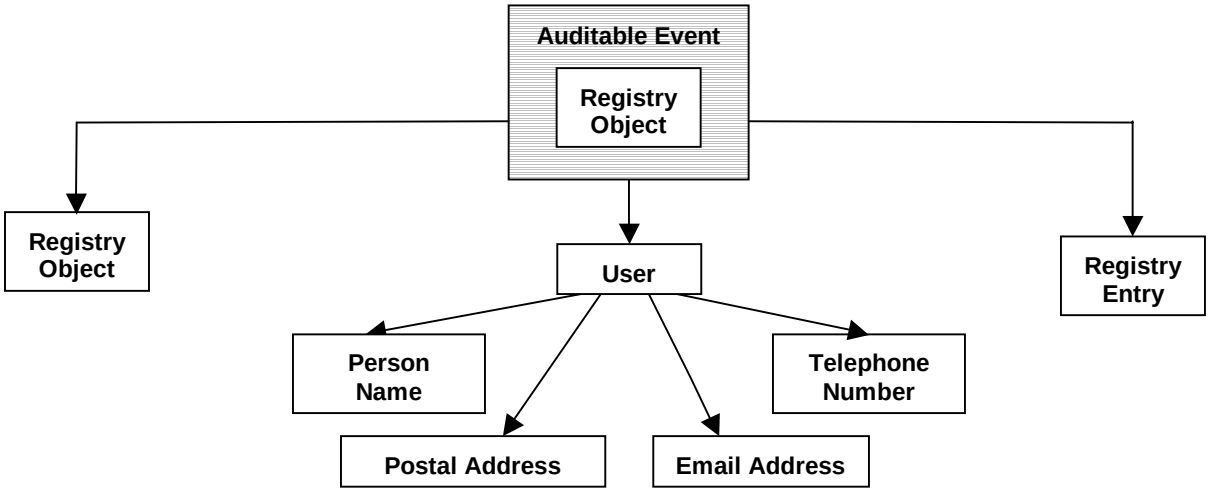


Figure 40: ebRIM Binding for AuditableEventQuery

Definition

```
<complexType name="AuditableEventQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:AuditableEventFilter" minOccurs="0" />
        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
        <element ref="tns:UserQuery" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />

<element name="AuditableEventQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:AuditableEvent" />
    </choice>
  </complexType>
</element>
```


Semantic Rules

1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The following steps will eliminate instances in AE that do not satisfy the conditions of the specified filters.
   a) If AE is empty then continue to the next numbered rule.
   b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from AE. If AE is empty then continue to the next numbered rule.
   c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not an auditable event for some registry object in RO, then remove x from AE. If AE is empty then continue to the next numbered rule.
   d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an auditable event for some registry entry in RE, then remove x from AE. If AE is empty then continue to the next numbered rule.
   e) If a UserQuery is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. If the defining user of x does not satisfy the UserQuery, then remove x from AE. If AE is empty then continue to the next numbered rule.
   f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited RegistryObjectQuery over AE as explained in Section 8.2.2.
2. If AE is empty, then raise the warning: *auditable event query result is empty*; otherwise set AE to be the result of the AuditableEventQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The client is now interested in all events since the beginning of the year that have impacted that item. The following query will return a set of AuditableEvent instances for all such events.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <AuditableEventQuery>
      <AuditableEventFilter>
        <Clause>
          <SimpleClause leftArgument = "timestamp">
            <RationalClause logicalPredicate = "GE">
              DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
            </RationalClause>
          </SimpleClause>
```



```
          </Clause>
        </AuditableEventFilter>
        <RegistryEntryQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </RegistryEntryQuery>
      </AuditableEventQuery>
  </FilterQuery>
</AdhocQueryRequest
```

A client company has many registered objects in the Registry. The Registry allows events submitted by other organizations to have an impact on your registered items, e.g. new classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <AuditableEventQuery>
      <RegistryEntryQuery>
        <TargetAssociationBranch>
          <AssociationFilter>
            <Clause>
              <SimpleClause leftArgument = "associationType">
               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
              </SimpleClause>
            </Clause>
          </AssociationFilter>
          <OrganizationQuery>
            <NameBranch>
              <LocalizedStringFilter>
                <Clause>
                  <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">myorg</StringClause>
                  </SimpleClause>
                </Clause>
              </LocalizedStringFilter>
            </NameBranch>
          </OrganizationQuery>
        </TargetAssociationBranch>
      </RegistryEntryQuery>
      <UserBranch>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "NotEqual">myorg</StringClause>
```



```
                    </SimpleClause>
                  </Clause>
                </LocalizedStringFilter>
              </NameBranch>
            </OrganizationQuery>
          </UserBranch>
        </AuditableEventQuery>
      </FilterQuery>
</AdhocQueryRequest>
```

8.2.6 ClassificationQuery

Purpose

To identify a set of classification instances as the result of a query over selected registry metadata.

ebRIM Binding

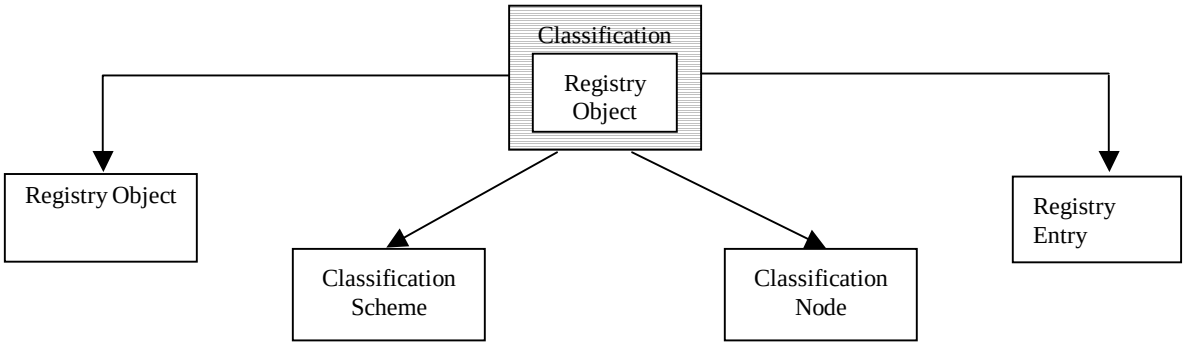


Figure 41: ebRIM Binding for ClassificationQuery

Definition

```
<complexType name = "ClassificationQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
      <sequence>
        <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>

<element name="ClassificationQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:Classification" />
    </choice>
```



```
  </complexType>
</element>
```

Semantic Rules

1. Let C denote the set of all persistent Classification instances in the Registry. The following steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
   a) If C is empty then continue to the next numbered rule.
   b) If a ClassificationFilter element is not directly contained in the ClassificationQuery element, then go to the next step; otherwise let x be an classification instance in C. If x does not satisfy the ClassificationFilter then remove x from C. If C is empty then continue to the next numbered rule.
   c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C. If C is empty then continue to the next numbered rule.
   d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the classification node of x does not satisfy the ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is empty then continue to the next numbered rule.
   e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a classification of at least one registry object in RO, then remove x from C. If C is empty then continue to the next numbered rule.
   f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a classification of at least one registry entry in RE, then remove x from C. If C is empty then continue to the next numbered rule.
2. If C is empty, then raise the warning: *classification query result is empty*; otherwise otherwise, set C to be the result of the ClassificationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.7 ClassificationNodeQuery

Purpose

To identify a set of classification node instances as the result of a query over selected registry metadata.

ebRIM Binding

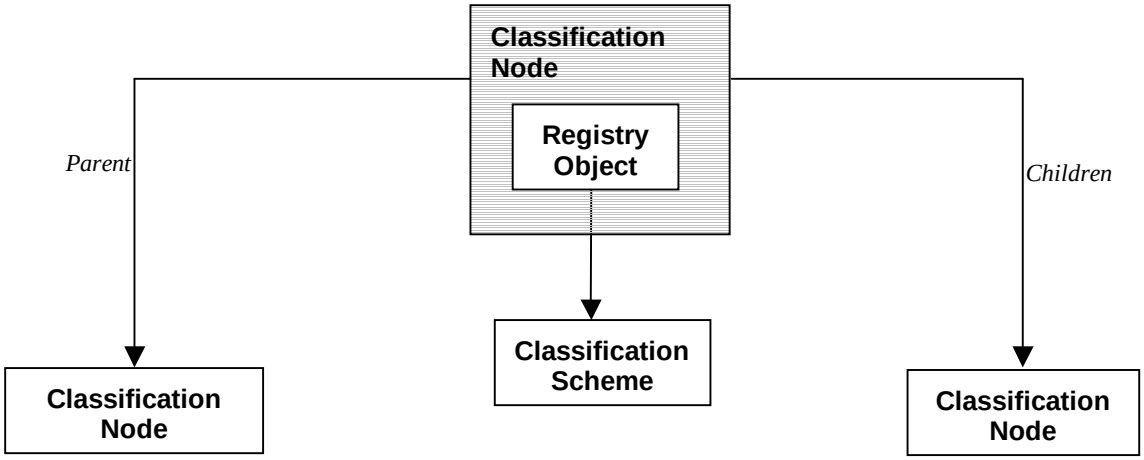


Figure 42: ebRIM Binding for ClassificationNodeQuery

Definition

```
<complexType name="ClassificationNodeQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
        <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
          maxOccurs="1" />
        <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />

<element name="ClassificationNodeQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:ClassificationNode" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The following steps will eliminate instances in CN that do not satisfy the conditions of the specified filters.
   a) If CN is empty then continue to the next numbered rule.
   b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove x from CN. If CN is empty then continue to the next numbered rule.

c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification node in CN. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from CN. If CN is empty then continue to the next numbered rule.

d) If a ClassificationNodeParentBranch element is not specified, then go to the next step; otherwise, let x be a remaining classification node in CN and execute the following paragraph with n=x.

Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base level node), then remove x from CN and go to the next step; otherwise, let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in the ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter, then remove x from CN. If CN is empty then continue to the next numbered rule. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeParentBranch and if defining classification scheme of p does not satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue to the next numbered rule.

If another ClassificationNodeParentBranch element is directly contained within this ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.

e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the parent node of some ClassificationNode instance, then remove x from CN and if CN is empty continue to the next numbered rule; otherwise, treat each ClassificationNodeChildrenBranch element separately and execute the following paragraph with n = x.

Let n be a classification node instance. If a ClassificationNodeFilter element is not specified within the ClassificationNodeChildrenBranch element then let CNC be the set of all classification nodes that have n as their parent node; otherwise, let CNC be the set of all classification nodes that satisfy the ClassificationNodeFilter and have n as their parent node. If CNC is empty, then remove x from CN and if CN is empty continue to the next numbered rule; otherwise, let c be any member of CNC. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeChildrenBranch and if the defining classification scheme of c does not satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then remove x from CN. If CN is empty then continue to the next numbered rule; otherwise, let y be an element of CNC and continue with the next paragraph.

If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly contain another ClassificationNodeChildrenBranch element, then continue to the next numbered rule; otherwise, repeat the previous paragraph with the new ClassificationNodeChildrenBranch element and with n = y.

f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited RegistryObjectQuery over CN as explained in Section 8.2.2.

2. If CN is empty, then raise the warning: *classification node query result is empty*; otherwise set CN to be the result of the ClassificationNodeQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Path Filter Expression usage in ClassificationNodeFilter

The path filter expression is used to match classification nodes in ClassificationNodeFilter elements involving the path attribute of the ClassificationNode class as defied by the getPath method in [ebRIM].

The path filter expressions are based on a very small and proper sub-set of location path syntax of XPath.

The path filter expression syntax includes support for matching multiple nodes by using wild card syntax as follows:

- Use of '*' as a wildcard in place of any path element in the pathFilter.
- Use of '//' syntax to denote any descendent of a node in the pathFilter. Support for '//' syntax is optional.

It is defined by the following BNF grammar:

```
pathFilter   ::= '/' schemeId nodePath
nodePath   ::= slashes nodeCode
                        |   slashes '*'
                        |   slashes nodeCode ( nodePath )?
Slashes     ::= '/' | '//'
```

In the above grammer, schemeId is the id attribute of the ClassificationScheme instance. In the above grammar nodeCode is defined by NCName production as defined by http://www.w3.org/TR/REC-xml-names/#NT-NCName.

The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause is a PATH Filter expression.

This is illustrated in the following example that matches all second level nodes in ClassificationScheme with id 'Geography-id' and with code 'Japan':

```
<ClassificationNodeQuery>
  <ClassificationNodeFilter>
    <Clause>
      <SimpleClause leftArgument = "path">
        <StringClause stringPredicate = "Equal">//Geography-id/*/Japan</StringClause>
      </SimpleClause>
    </Clause>
  </ClassificationNodeFilter>
</ClassificationNodeQuery>
```

Use Cases and Examples of Path Filter Expressions

The following table lists various use cases and examples using the sample Geography scheme below:

```
<ClassificationScheme id='Geography-id' name="Geography"/>

<ClassificationNode id="NorthAmerica-id" parent="Geography-id" code=NorthAmerica" />
<ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />

<ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
<ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
<ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```


Table 8: Path Filter Expressions for Use Cases

Use Case	PATH Expression	Description
Match all nodes in first level that have a specified value	/Geography-id/NorthAmerica	Find all first level nodes whose code is 'NorthAmerica'
Find all children of first level node whose code is “NorthAmerica”	/Geography-id/NorthAmerica/*	Match all nodes whose first level path element has code "NorthAmerica"
Match all nodes that have a specified value regardless of level	/ Geography-id//Japan	Find all nodes with code "Japan”
Match all nodes in the second level that have a specified value	/Geography-id/*/Japan	Find all second level nodes with code 'Japan'
Match all nodes in the 3rd level that have a specified value	/ Geography-id/*/*/Tokyo	Find all third level nodes with code 'Tokyo'

Examples

A client application wishes to identify all of the classification nodes in the first three levels of a classification scheme hierarchy. The client knows that the name of the underlying classification scheme is “urn:ebxml:cs:myscheme”. The following query identifies all nodes at the first three levels.

```
<AdhocQueryRequest>
   <ResponseOption returnType = "LeafClass"/>
   <FilterQuery>
      <ClassificationNodeQuery>
         <ClassificationNodeFilter>
            <Clause>
               <SimpleClause leftArgument = "levelNumber">
                  <RationalClause logicalPredicate = "LE">
                    <IntClause>3</IntClause>
                  </RationalClause>
               </SimpleClause>
            </Clause>
         </ClassificationNodeFilter>
         <ClassificationSchemeQuery>
            <NameBranch>
               <LocalizedStringFilter>
                  <Clause>
                     <SimpleClause leftArgument = "value">
                       <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
                     </SimpleClause>
                  </Clause>
```



```
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassificationNodeQuery>
    </FilterQuery>
</AdhocQueryRequest>
```

If, instead, the client wishes all levels returned, they could simply delete the ClassificationNodeFilter element from the query.

The following query finds all children nodes of a first level node whose code is NorthAmerica.

```
<AdhocQueryRequest>
   <ResponseOption returnType = "LeafClass"/>
   <FilterQuery>
      <ClassificationNodeQuery>
         <ClassificationNodeFilter>
           <Clause>
            <SimpleClause leftArgument = "path">
              <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
            </SimpleClause>
           </Clause>
         </ClassificationNodeFilter>
      </ClassificationNodeQuery>
   </FilterQuery>
</AdhocQueryRequest>
```

The following query finds all third level nodes with code of Tokyo.

```
<AdhocQueryRequest>
   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
   <FilterQuery>
      <ClassificationNodeQuery>
         <ClassificationNodeFilter>
            <Clause>
              <SimpleClause leftArgument = "path">
                <StringClause stringPredicate = "Equal">/Geography-id/*/*/Tokyo</StringClause>
              </SimpleClause>
            </Clause>
         </ClassificationNodeFilter>
      </ClassificationNodeQuery>
   </FilterQuery>
</AdhocQueryRequest>
```

8.2.8 ClassificationSchemeQuery

Purpose

To identify a set of classification scheme instances as the result of a query over selected registry metadata.

ebRIM Binding

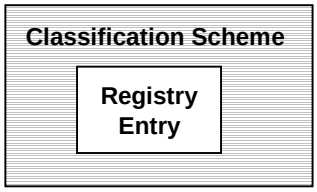


Figure 43: ebRIM Binding for ClassificationSchemeQuery

Definition

```
<complexType name="ClassificationSchemeQueryType">
  <complexContent>
    <extension base="tns:RegistryEntryQueryType">
      <sequence>
        <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />

<element name = "ClassificationSchemeQueryResult">
  <complexType>
    <choice minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "rim:ObjectRef"/>
      <element ref = "rim:RegistryObject"/>
      <element ref = "rim:RegistryEntry"/>
      <element ref = "rim:ClassificationScheme"/>
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The following steps will eliminate instances in CS that do not satisfy the conditions of the specified filters.
   a) If CS is empty then continue to the next numbered rule.
   b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter, then remove x from CS. If CS is empty then continue to the next numbered rule.
   c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited RegistryEntryQuery over CS as explained in Section 8.2.3.
2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise, set CS to be the result of the ClassificationSchemeQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify all classification scheme instances in the Registry.


```
<AdhocQueryRequest>
   <ResponseOption returnType = "LeafClass"/>
   <FilterQuery>
      <ClassificationSchemeQuery/>
   </FilterQuery>
</AdhocQueryRequest>
```

8.2.9 RegistryPackageQuery

Purpose

To identify a set of registry package instances as the result of a query over selected registry metadata.

ebRIM Binding

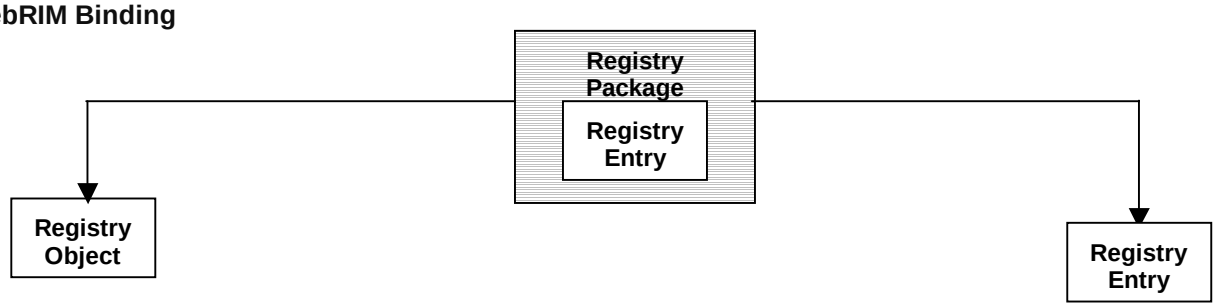


Figure 44: ebRIM Binding for RegistryPackageQuery

Definition

```
<complexType name="RegistryPackageQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
            <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
            <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />

<element name="RegistryPackageQueryResult">
   <complexType>
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="rim:ObjectRef" />
         <element ref="rim:RegistryEntry" />
         <element ref="rim:RegistryObject" />
         <element ref="rim:RegistryPackage" />
      </choice>
   </complexType>
</element>
```


Semantic Rules

1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The following steps will eliminate instances in RP that do not satisfy the conditions of the specified filters.
   a) If RP is empty then continue to the next numbered rule.
   b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule; otherwise, let x be a registry package instance in RP. If x does not satisfy the RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next numbered rule.
   c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryObjectQuery as follows: let RO be the set of RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2 and let PO be the subset of RO that are members of the package x. If PO is empty, then remove x from RP. If RP is empty then continue to the next numbered rule. If a RegistryEntryQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be the subset of RE that are members of the package x. If PE is empty, then remove x from RP. If RP is empty then continue to the next numbered rule.
   d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited RegistryEntryQuery over RP as explained in Section 8.2.3.
2. If RP is empty, then raise the warning: *registry package query result is empty*; otherwise set RP to be the result of the RegistryPackageQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify all package instances in the Registry that contain an Invoice extrinsic object as a member of the package.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <RegistryPackageQuery>
      <RegistryEntryQuery>
        <RegistryEntryFilter>
          <Clause>
            <SimpleClause leftArgument = "objectType">
              <StringClause stringPredicate = "Equal">Invoice</StringClause>
            </SimpleClause>
          </Clause>
        </RegistryEntryFilter>
      </RegistryEntryQuery>
    </RegistryPackageQuery>
  </FilterQuery>
</AdhocQueryRequest>
```


A client application wishes to identify all package instances in the Registry that are not empty.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <RegistryPackageQuery>
      <RegistryObjectQuery/>
    </RegistryPackageQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client application wishes to identify all package instances in the Registry that are empty. Since the RegistryPackageQuery is not set up to do negations, clients will have to do two separate RegistryPackageQuery requests, one to find all packages and another to find all non-empty packages, and then do the set difference themselves. Alternatively, they could do a more complex RegistryEntryQuery and check that the packaging association between the package and its members is non-existent.

Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However, the equivalent RegistryEntryQuery is often more complicated to write.

8.2.10 ExtrinsicObjectQuery

Purpose

To identify a set of extrinsic object instances as the result of a query over selected registry metadata.

ebRIM Binding

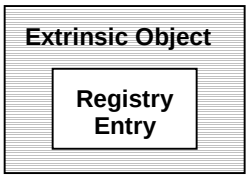


Figure 45: ebRIM Binding for ExtrinsicObjectQuery

Definition

```
<complexType name="ExtrinsicObjectQueryType">
  <complexContent>
    <extension base="tns:RegistryEntryQueryType">
      <sequence>
        <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
```



```
<element name="ExtrinsicObjectQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryEntry" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:ExtrinsicObject" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The following steps will eliminate instances in EO that do not satisfy the conditions of the specified filters.
   a) If EO is empty then continue to the next numbered rule.
   b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from EO. If EO is empty then continue to the next numbered rule.
   c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited RegistryEntryQuery over EO as explained in Section 8.2.3.
2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set EO to be the result of the ExtrinsicObjectQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.11 OrganizationQuery

Purpose

To identify a set of organization instances as the result of a query over selected registry metadata.

ebRIM Binding

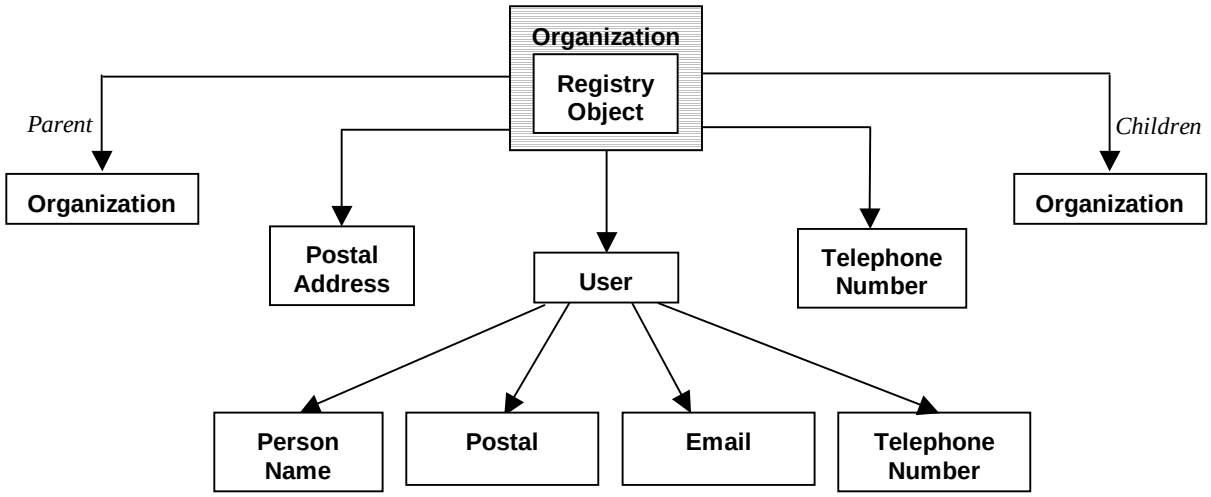


Figure 46: ebRIM Binding for OrganizationQuery

Definition

```
<complexType name="OrganizationQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:OrganizationFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
        <element ref="tns:UserQuery" minOccurs="0" maxOccurs="1" />
        <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0
          " maxOccurs="1" />
        <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="OrganizationQuery" type="tns:OrganizationQueryType" />

<element name="OrganizationQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
```



```
        <element ref="rim:RegistryObject" />
        <element ref="rim:Organization" />
      </choice>
    </complexType>
</element>
```

Semantic Rules

1. Let ORG denote the set of all persistent Organization instances in the Registry. The following steps will eliminate instances in ORG that do not satisfy the conditions of the specified filters.

   a) If ORG is empty then continue to the next numbered rule.

   b) If an OrganizationFilter element is not directly contained in the OrganizationQuery element, then go to the next step; otherwise let x be an organization instance in ORG. If x does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then continue to the next numbered rule.

   c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an organization in ORG. If postal address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is empty then continue to the next numbered rule.

   d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an organization in ORG. If any of the TelephoneNumberFilters isn't satisfied by all of the telephone numbers of x then remove x from ORG. If ORG is empty then continue to the next numbered rule.

   e) If a UserQuery is not specified then go to the next step; otherwise, let x be a remaining organization in ORG. If the defining primary contact of x does not satisfy the UserQuery, then remove x from ORG. If ORG is empty then continue to the next numbered rule.

   f) If a OrganizationParentBranch element is not specified within the OrganizationQuery, then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the following paragraph with o = x:

   Let o be an organization instance. If an OrganizationFilter is not specified within the OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the Organization hierarchy), then remove x from ORG; otherwise, let p be the parent organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG. If ORG is empty then continue to the next numbered rule.

   If another OrganizationParentBranch element is directly contained within this OrganizationParentBranch element, then repeat the previous paragraph with o = p.

   g) If a OrganizationChildrenBranch element is not specified, then continue to the next numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent node of some organization instance, then remove x from ORG and if ORG is empty continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch element separately and execute the following paragraph with n = x.

Let n be an organization instance. If an OrganizationFilter element is not specified within the OrganizationChildrenBranch element then let ORGC be the set of all organizations that have n as their parent node; otherwise, let ORGC be the set of all organizations that satisfy the OrganizationFilter and have n as their parent node. If ORGC is empty, then remove x from ORG and if ORG is empty continue to the next numbered rule; otherwise, let c be any member of ORGC. If a PostalAddressFilter element is directly contained in the OrganizationChildrenBranch and if the postal address of c does not satisfy the PostalAddressFilter then remove c from ORGC. If ORGC is empty then remove x from ORG. If ORG is empty then continue to the next numbered rule. If no TelephoneNumberFilter element is directly contained in the OrganizationChildrenBranch and if any of the TelephoneNumberFilters isn't satisfied by all of the telephone numbers of c then remove c from ORGC. If ORGC is empty then remove x from ORG. If ORG is empty then continue to the next numbered rule; otherwise, let y be an element of ORGC and continue with the next paragraph.

If the OrganizationChildrenBranch element is terminal, i.e. if it does not directly contain another OrganizationChildrenBranch element, then continue to the next numbered rule; otherwise, repeat the previous paragraph with the new OrganizationChildrenBranch element and with n = y.

h) Let ORG be the set of remaining Organization instances. Evaluate inherited RegistryObjectQuery over ORG as explained in Section 8.2.2.

2. If ORG is empty, then raise the warning: *organization query result is empty*; otherwise set ORG to be the result of the OrganizationQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify a set of organizations, based in France, that have submitted a PartyProfile extrinsic object this year.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
  <FilterQuery>
        <OrganizationQuery>
            <SourceAssociationBranch>
                <AssociationFilter>
                    <Clause>
                        <SimpleClause leftArgument = "associationType">
                            <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
                        </SimpleClause>
                    </Clause>
                </AssociationFilter>
                <RegistryObjectQuery>
                    <RegistryObjectFilter>
                        <Clause>
                            <SimpleClause leftArgument = "objectType">
                                <StringClause stringPredicate = "Equal">CPP</StringClause>
                            </SimpleClause>
                        </Clause>
                    </RegistryObjectFilter>
                    <AuditableEventQuery>
```



```
                        <AuditableEventFilter>
                            <Clause>
                                <SimpleClause leftArgument = "timestamp">
                                  <RationalClause logicalPredicate = "GE">
                                    <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
                                  </RationalClause>
                                </SimpleClause>
                            </Clause>
                        </AuditableEventFilter>
                    </AuditableEventQuery>
                </RegistryObjectQuery>
            </SourceAssociationBranch>
            <PostalAddressFilter>
                <Clause>
                    <SimpleClause leftArgument = "country">
                        <StringClause stringPredicate = "Equal">France</StringClause>
                    </SimpleClause>
                </Clause>
            </PostalAddressFilter>
        </OrganizationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client application wishes to identify all organizations that have Corporation named XYZ as a parent.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <OrganizationQuery>
      <OrganizationParentBranch>
        <NameBranch>
          <LocalizedStringFilter>
            <Clause>
              <SimpleClause leftArgument = "value">
                <StringClause stringPredicate = "Equal">XYZ</StringClause>
              </SimpleClause>
            </Clause>
          </LocalizedStringFilter>
        </NameBranch>
      </OrganizationParentBranch>
    </OrganizationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.12 ServiceQuery

Purpose

To identify a set of service instances as the result of a query over selected registry metadata.

ebRIM Binding

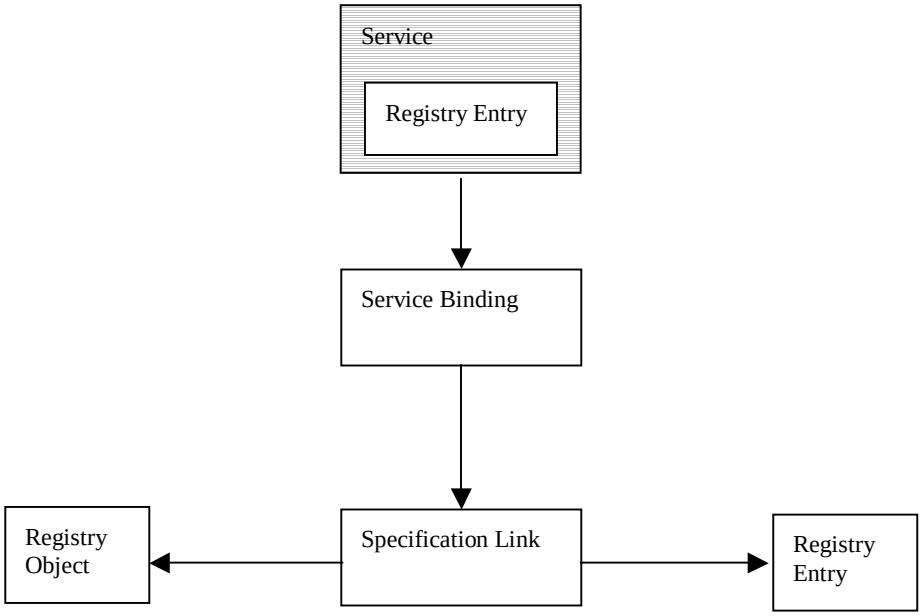


Figure 47: ebRIM Binding for ServiceQuery

Definition

```
<complexType name="ServiceQueryType">
  <complexContent>
    <extension base="tns:RegistryEntryQueryType">
      <sequence>
        <element ref="tns:ServiceFilter" minOccurs="0"
          maxOccurs="1" />
        <element ref="tns:ServiceBindingBranch" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ServiceQuery" type="tns:ServiceQueryType" />

<element name="ServiceQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:RegistryEntry" />
      <element ref="rim:Service" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let S denote the set of all persistent Service instances in the Registry. The following steps will eliminate instances in S that do not satisfy the conditions of the specified filters.

   a) If S is empty then continue to the next numbered rule.

b) If a ServicetFilter is not specified then go to the next step; otherwise, let x be a service in S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then continue to the next numbered rule.

c) If a ServiceBindingBranch is not specified then continue to the next numbered rule; otherwise, consider each ServiceBindingBranch element separately as follows:

Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the member of SB. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue to the next numbered rule. If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in SB. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue to the next numbered rule. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue to the next numbered rule.

d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery over AE as explained in Section 8.2.3.

2. If S is empty, then raise the warning: *service query result is empty*; otherwise set S to be the result of the ServiceQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.13 FederationQuery

Purpose

To identify a set of federation instances as the result of a query over selected registry metadata.

ebRIM Binding

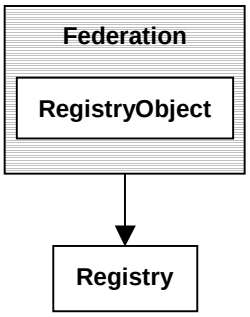


Figure 48: ebRIM Binding for FederationQuery

Definition

```
<complexType name = "FederationQueryType">
  <complexContent>
    <extension base = "tns:RegistryEntryQueryType">
      <sequence>
        <element ref = "tns:FederationFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryQuery" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name = "FederationQuery" type = "tns:FederationQueryType"/>

<element name = "FederationQueryResult">
  <complexType>
    <choice minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "rim:ObjectRef"/>
      <element ref = "rim:RegistryObject"/>
      <element ref = "rim:RegistryEntry"/>
      <element ref = "rim:Federation"/>
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let F denote the set of all persistent Federation instances in the Registry. The following steps will eliminate instances in F that do not satisfy the conditions of the specified filters.
   a) If F is empty then continue to the next numbered rule.

b) If a FederationFilter is not specified, then continue to the next numbered rule; otherwise, let x be a federation instance in F. If x does not satisfy the FederationFilter then remove x from F. If F is empty then continue to the next numbered rule.

c) If a RegistryQuery element is directly contained in the FederationQuery element then treat each RegistryQuery as follows: let R be the set of Registry instances returned by the RegistryQuery as defined in Section and let FR be the subset of R that are members of the federation x. If FR is empty, then remove x from F. If F is empty then continue to the next numbered rule.

d) Let F be the set of remaining Federation instances. Evaluate inherited RegistryEntryQuery over F as explained in Section.

2. If F is empty, then raise the warning: *federation query result is empty*; otherwise set F to be the result of the FederationQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.14 RegistryQuery

Purpose

To identify a set of registry instances as the result of a query over selected registry metadata.

ebRIM Binding

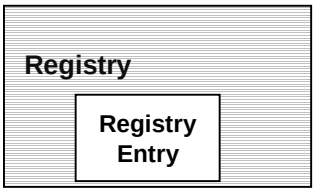


Figure 49: ebRIM Binding for RegistryQuery

Definition

```
<complexType name="RegistryQueryType">
  <complexContent>
    <extension base="tns:RegistryEntryQueryType">
      <sequence>
        <element ref="tns:RegistryFilter" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="RegistryQuery" type="tns:RegistryQueryType" />

<element name = "RegistryQueryResult">
  <complexType>
    <choice minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "rim:ObjectRef"/>
      <element ref = "rim:RegistryObject"/>
      <element ref = "rim:RegistryEntry"/>
      <element ref = "rim:Registry"/>
```



```
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let R denote the set of all persistent Registry instances in the Registry. The following steps will eliminate instances in R that do not satisfy the conditions of the specified filters.
   a) If R is empty then continue to the next numbered rule.
   b) If a RegistryFilter is not specified then go to the next step; otherwise, let x be a registry in R. If x does not satisfy the RegistryFilter, then remove x from R. If R is empty then continue to the next numbered rule.
   c) Let R be the set of remaining Registry instances. Evaluate inherited RegistryEntryQuery over R as explained in Section.
2. If R is empty, then raise the warning: *registry query result is empty*; otherwise, set R to be the result of the RegistryQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.15 SubscriptionQuery

Purpose

To identify a set of subscription instances as the result of a query over selected registry metadata.

ebRIM Binding

Figure 50: ebRIM Binding for SubscriptionQuery

Definition

```
<complexType name = "SubscriptionQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
      <sequence>
        <element ref = "tns: SubscriptionFilter" minOccurs = "0" maxOccurs = "1"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name = "SubscriptionQuery" type = "tns: SubscriptionQueryType"/>
```



```
<element name="SubscriptionQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim: Subscription" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let S denote the set of all persistent Subscription instances in the Registry. The following steps will eliminate instances in S that do not satisfy the conditions of the specified filters.
   a) If S is empty then continue to the next numbered rule.
   b) If a SubscriptionFilter element is not directly contained in the SubscriptionQuery element, then go to the next step; otherwise let x be a subscription instance in S. If x does not satisfy the SubscriptionFilter then remove x from S. If S is empty then continue to the next numbered rule.
   c) Let S be the set of remaining Subscription instances. Evaluate inherited RegistryObjectQuery over S as explained in Section.
2. If S is empty, then raise the warning: *subscription query result is empty*; otherwise, set S to be the result of the SubscriptionQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.16 UserQuery

Purpose

To identify a set of user instances as the result of a query over selected registry metadata.

ebRIM Binding

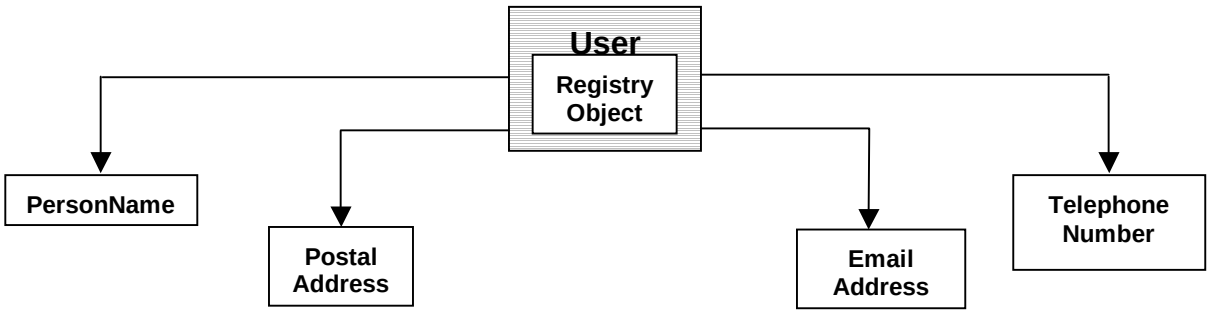


Figure 51: ebRIM Binding for OrganizationQuery Add PersonName under User and line up

Definition

```
<complexType name = "UserQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
          <sequence>
               <element ref = "tns:UserFilter" minOccurs = "0" maxOccurs="1"/>
               <element ref = "tns:EmailAddressFilter" minOccurs = "0" maxOccurs="unbounded"/>
               <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
               <element ref = "tns:PersonNameFilter" minOccurs = "0" maxOccurs="1"/>
               <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="unbounded"/>
          </sequence>
    </extension>
  </complexContent>
</complexType >
<element name = "UserQuery" type = "tns:UserQueryType"/>

<element name = "UserQueryResult">
  <complexType>
    <choice minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "rim:ObjectRef"/>
      <element ref = "rim:RegistryObject"/>
      <element ref = "rim:User"/>
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let U denote the set of all persistent User instances in the Registry. The following steps will eliminate instances in U that do not satisfy the conditions of the specified filters.
   a) If U is empty then continue to the next numbered rule.
   b) If an UserFilter element is not directly contained in the UserQuery element, then go to the next step; otherwise let x be an user instance in U. If x does not satisfy the UserFilter then remove x from U. If U is empty then continue to the next numbered rule.
   c) If a EmailAddressFilter element is not directly contained in the UserQuery element then go to the next step; otherwise, let x be an user in U. If email address of x does not satisfy the EmailAddressFilter then remove x from U. If U is empty then continue to the next numbered rule.
   d) If a PostalAddressFilter element is not directly contained in the UserQuery element then go to the next step; otherwise, let x be an user in U. If postal address of x does not satisfy the PostalAddressFilter then remove x from U. If U is empty then continue to the next numbered rule.
   e) If a PersonNameFilter element is not directly contained in the UserQuery element then go to the next step; otherwise, let x be an user in U. If reson name of x does not satisfy the PersonNameFilter then remove x from U. If U is empty then continue to the next numbered rule.

f) If no TelephoneNumberFilter element is directly contained in the UserQuery element then go to the next step; otherwise, let x be an user in U. If any of the TelephoneNumberFilters isn't satisfied by all of the telephone numbers of x then remove x from U. If U is empty then continue to the next numbered rule.

g) Let U be the set of remaining User instances. Evaluate inherited RegistryObjectQuery over U as explained in Section 8.2.2.

2. If U is empty, then raise the warning: *user query result is empty*; otherwise set U to be the result of the UserQuery.

Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.17 Registry Filters

Purpose

To identify a subset of the set of all persistent instances of a given registry class.

Definition

```
<complexType name="FilterType">
  <sequence>
    <element ref="tns:Clause" />
  </sequence>
</complexType>
<element name="RegistryObjectFilter" type="tns:FilterType" />
<element name="RegistryEntryFilter" type="tns:FilterType" />
<element name="ExtrinsicObjectFilter" type="tns:FilterType" />
<element name="RegistryPackageFilter" type="tns:FilterType" />
<element name="OrganizationFilter" type="tns:FilterType" />
<element name="ClassificationNodeFilter" type="tns:FilterType" />
<element name="AssociationFilter" type="tns:FilterType" />
<element name="ClassificationFilter" type="tns:FilterType" />
<element name="ClassificationSchemeFilter" type="tns:FilterType" />
<element name="ExternalLinkFilter" type="tns:FilterType" />
<element name="ExternalIdentifierFilter" type="tns:FilterType" />
<element name="SlotFilter" type="tns:FilterType" />
<element name="AuditableEventFilter" type="tns:FilterType" />
<element name="UserFilter" type="tns:FilterType" />
<element name="SlotValueFilter" type="tns:FilterType" />
<element name="PostalAddressFilter" type="tns:FilterType" />
<element name="TelephoneNumberFilter" type="tns:FilterType" />
<element name="EmailAddressFilter" type="tns:FilterType" />
<element name="ServiceFilter" type="tns:FilterType" />
<element name="ServiceBindingFilter" type="tns:FilterType" />
<element name="SpecificationLinkFilter" type="tns:FilterType" />
<element name="LocalizedStringFilter" type="tns:FilterType" />
<element name="FederationFilter" type = "tns:FilterType"/>
<element name="PersonNameFilter" type = "tns:FilterType"/>
<element name="RegistryFilter" type = "tns:FilterType"/>
<element name="SubscriptionFilter" type = "tns:FilterType"/>
```


Semantic Rules

1. The Clause element is defined in Section 8.2.18.
2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryObject UML class defined in [ebRIM]. If not, raise exception: *registry object attribute error*. The RegistryObjectFilter returns a set of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the Clause predicate.
3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True* for the Clause predicate.
4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause predicate.
5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for the Clause predicate.
6. For every OrganizationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Organization or PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The OrganizationFilter returns a set of identifiers for Organization instances whose attribute values evaluate to *True* for the Clause predicate.
7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the leftAttribute is the visible attribute “path” then if stringPredicate of the StringClause is not “Equal” then raise exception: *classification node path attribute error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose attribute values evaluate to *True* for the Clause predicate.
8. For every AssociationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Association UML class defined in [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a set of identifiers for Association instances whose attribute values evaluate to *True* for the Clause predicate.
9. For every ClassificationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Classification UML class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter returns a set of identifiers for Classification instances whose attribute values evaluate to *True* for the Clause predicate.

10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances whose attribute values evaluate to *True* for the Clause predicate.

11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalLink UML class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the Clause predicate.

12. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose attribute values evaluate to *True* for the Clause predicate.

13. For every SlotFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot instances whose attribute values evaluate to *True* for the Clause predicate.

14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to *True* for the Clause predicate.

15. For every UserFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User instances whose attribute values evaluate to *True* for the Clause predicate.

16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one SlotValue instance for each “value” in the “values” list of a Slot instance. The visible attribute of SlotValue is“value”. It is a character string. The dynamic instances of SlotValue are derived from the “values” attribute defined in ebRIM for a Slot instance. For every SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception: *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value” attribute evaluates to *True* for the Clause predicate.

17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the PostalAddress UML class defined in [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True* for the Clause predicate.

18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the TelephoneNumber UML class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*. The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances whose attribute values evaluate to *True* for the Clause predicate.

19. For every EmailAddressFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the EmailAddress UML class defined in [ebRIM]. If not, raise exception: *email address attribute error*. The EmailAddressFilter returns a set of identifiers for EmailAddresss instances whose attribute values evaluate to *True* for the Clause predicate.

20. For every ServiceFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM]. If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for Service instances whose attribute values evaluate to *True* for the Clause predicate.

21. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to *True* for the Clause predicate.

22. For every SpecificationLinkFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the SpecificationLink UML class defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose attribute values evaluate to *True* for the Clause predicate.

23. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the LocalizedString UML class defined in [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter returns a set of identifiers for LocalizedString instances whose attribute values evaluate to *True* for the Clause predicate.

24. For every FederationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Federation UML class defined in [ebRIM]. If not, raise exception: *federation attribute error*. The FederationFilter returns a set of identifiers for Federation instances whose attribute values evaluate to *True* for the Clause predicate.

25. For every PersonNameFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the PersonName UML class defined in [ebRIM]. If not, raise exception: *person name attribute error*. The PersonNameFilter returns a set of identifiers for PersonName instances whose attribute values evaluate to *True* for the Clause predicate.

26. For every RegistryFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Registry UML class defined in [ebRIM]. If not, raise exception: *registry attribute error*. The RegistryFilter returns a set of identifiers for Registry instances whose attribute values evaluate to *True* for the Clause predicate.

27. For every SubscriptionFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Subscription UML class defined in [ebRIM]. If not, raise exception: *subscription attribute error*. The SubscriptionFilter returns a set of identifiers for Subscription instances whose attribute values evaluate to *True* for the Clause predicate.

8.2.18 XML Clause Constraint Representation

Purpose

The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses. Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to simply as Clauses in this specification.

Conceptual Diagram

The following is a conceptual diagram outlining the Clause structure.

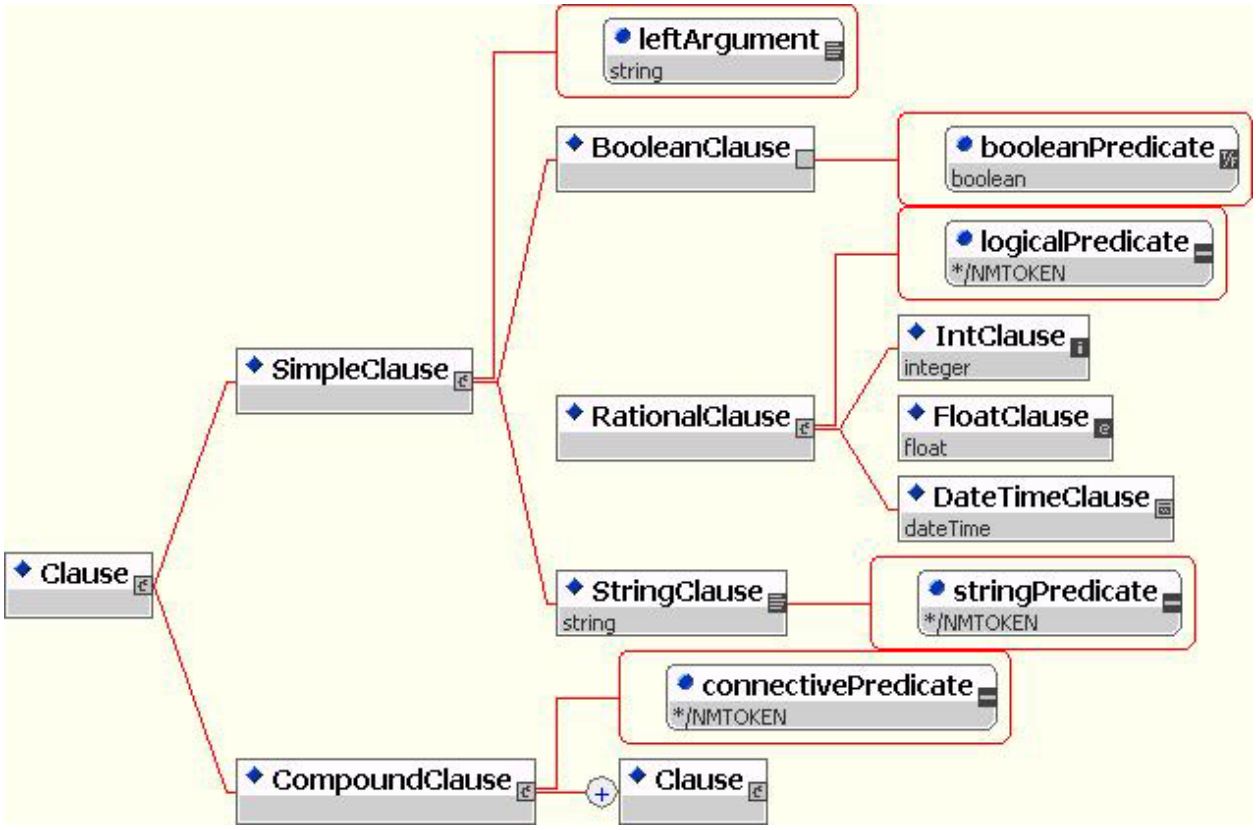


Figure 52: The Clause Structure

Semantic Rules

Predicates and Arguments are combined into a "LeftArgument - Predicate - RightArgument" format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.

SimpleClauses

A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the

Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended. SimpleClause is extended to support BooleanClause, StringClause, and RationalClause (abstract).

BooleanClause

BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a boolean.

StringClauses

StringClause defines the predicate as an enumerated attribute of appropriate string-compare operations and a right argument as the element's text data. String compare operations are defined as follow:

- Contains: Evaluates to true if left argument contains the content of the StringClause. Evaluates to false otherswise.
- NotContains: Evaluates to true if left argument does not contain the content of the StringClause. Evaluates to false otherswise.
- StartsWith: Evaluates to true if left argument starts with the content of the StringClause. Evaluates to false otherswise.
- NotStartsWith: Evaluates to true if left argument does not start with the content of the StringClause. Evaluates to false otherswise.
- Like: Evaluates to true if left argument matches the pattern specified by the content of the StringClause. Evaluates to false otherswise. The pattern for the Like operation is a sub-set of the LIKE syntax in SQL-92. The '*' or '%' character matches any number of characters while the '?' or '_' character matches a single character.
- NotLike: Evaluates to true if left argument does not match the pattern specified by the content of the StringClause. Evaluates to false otherswise.
- Equal: Evaluates to true if left argument is lexically equal to the content of the StringClause. Evaluates to false otherswise.
- NotEqual: Evaluates to true if left argument is lexically not equal to the content of the StringClause. Evaluates to false otherswise.
- EndsWith: Evaluates to true if left argument ends with the content of the StringClause. Evaluates to false otherswise.
- NotEndsWith: Evaluates to true if left argument does not end with the content of the StringClause. Evaluates to false otherswise.

RationalClauses

Rational number support is provided through a common RationalClause providing an enumeration of appropriate rational number compare operations, which is further extended to IntClause and FloatClause, each with appropriate signatures for the right argument.

CompoundClauses

A CompoundClause contains two or more Clauses (Simple or Compound) and a connective predicate. This provides for arbitrarily complex Clauses to be formed.

Definition

```
<element name = "Clause">
  <annotation>
    <documentation xml:lang = "en">
The following lines define the XML syntax for Clause.

    </documentation>
  </annotation>
  <complexType>
    <choice>
      <element ref = "tns:SimpleClause"/>
      <element ref = "tns:CompoundClause"/>
    </choice>
  </complexType>
</element>
<element name = "SimpleClause">
  <complexType>
    <choice>
      <element ref = "tns:BooleanClause"/>
      <element ref = "tns:RationalClause"/>
      <element ref = "tns:StringClause"/>
    </choice>
    <attribute name = "leftArgument" use = "required" type = "string"/>
  </complexType>
</element>
<element name = "CompoundClause">
  <complexType>
    <sequence>
      <element ref = "tns:Clause" maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "connectivePredicate" use = "required">
      <simpleType>
        <restriction base = "NMTOKEN">
          <enumeration value = "And"/>
          <enumeration value = "Or"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>
</element>
<element name = "BooleanClause">
  <complexType>
    <attribute name = "booleanPredicate" use = "required" type = "boolean"/>
  </complexType>
</element>
<element name = "RationalClause">
  <complexType>
    <choice>
      <element ref = "tns:IntClause"/>
      <element ref = "tns:FloatClause"/>
      <element ref = "tns:DateTimeClause"/>
    </choice>
    <attribute name = "logicalPredicate" use = "required">
```



```
          <simpleType>
            <restriction base = "NMTOKEN">
              <enumeration value = "LE"/>
              <enumeration value = "LT"/>
              <enumeration value = "GE"/>
              <enumeration value = "GT"/>
              <enumeration value = "EQ"/>
              <enumeration value = "NE"/>
            </restriction>
          </simpleType>
        </attribute>
      </complexType>
    </element>
    <element name = "IntClause" type = "integer"/>
    <element name = "FloatClause" type = "float"/>
    <element name = "DateTimeClause" type = "dateTime"/>

    <element name = "StringClause">
      <complexType>
        <simpleContent>
          <extension base = "string">
            <attribute name = "stringPredicate" use = "required">
              <simpleType>
                <restriction base = "NMTOKEN">
                  <enumeration value = "Contains"/>
                  <enumeration value = "NotContains"/>
                  <enumeration value = "StartsWith"/>
                  <enumeration value = "NotStartsWith"/>
                  <enumeration value = "Like"/>
                  <enumeration value = "Equal"/>
                  <enumeration value = "NotEqual"/>
                  <enumeration value = "EndsWith"/>
                  <enumeration value = "NotEndsWith"/>
                </restriction>
              </simpleType>
            </attribute>
          </extension>
        </simpleContent>
      </complexType>
    </element>
```

Examples

Simple BooleanClause: "Smoker" = True

```
<Clause>
  <SimpleClause leftArgument="Smoker">
    <BooleanClause booleanPredicate="True"/>
  </SimpleClause>
</Clause>

<BooleanClause param="sqlQuerySupported" operation="Equals”>
  <value>true</value>
</BooleanClause>
```

Simple StringClause: "Smoker" contains "mo"

```
<Clause>
   <SimpleClause leftArgument = "Smoker">
      <StringClause stringPredicate = "Contains">mo</StringClause>
   </SimpleClause>
<Clause>
```

Simple IntClause: "Age" >= 7

```
<Clause>
   <SimpleClause leftArgument="Age">
      <RationalClause logicalPredicate="GE">
         <IntClause>7</IntClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

Simple FloatClause: "Size" = 4.3

```
<Clause>
   <SimpleClause leftArgument="Size">
      <RationalClause logicalPredicate="Equal">
         <FloatClause>4.3</FloatClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

Compound with two Simples (("Smoker" = False)AND("Age" =< 45))

```
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
            <BooleanClause booleanPredicate="False"/>
         </SimpleClause>
      </Clause>
      <Clause>
         <SimpleClause leftArgument="Age">
            <RationalClause logicalPredicate="LE">
               <IntClause>45</IntClause>
            </RationalClause>
         </SimpleClause>
      </Clause>
   </CompoundClause>
</Clause>
```

Coumpound with one Simple and one Compound

(("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

```
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
```



```
          <BooleanClause booleanPredicate="False"/>
        </SimpleClause>
      </Clause>
      <Clause>
        <CompoundClause connectivePredicate="Or">
          <Clause>
            <SimpleClause leftArgument="Age">
              <RationalClause logicalPredicate="LE">
                <IntClause>45</IntClause>
              </RationalClause>
            </SimpleClause>
          </Clause>
          <Clause>
            <SimpleClause leftArgument="American">
              <BooleanClause booleanPredicate="True"/>
            </SimpleClause>
          </Clause>
        </CompoundClause>
      </Clause>
    </CompoundClause>
<Clause>
```

8.3 SQL Query Support

The Registry may optionally support an SQL based query capability that is designed for Registry clients that demand more advanced query capability. The optional SQLQuery element in the AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query language.

The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database Language SQL [SQL], extended to include `<sql invoked routines>` (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined in template form in Appendix D.3. The syntax of the Registry query language is defined by the BNF grammar in D.1.

Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use relational databases in a Registry implementation.

8.3.1 SQL Query Syntax Binding To [ebRIM]

SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as described in the following sections.

Class Binding

A subset of the class names defined in [ebRIM] map to table names that may be queried by an SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an SQL query.

The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3 is as follows:

- Classes that have concrete instances are mapped to relational tables. In addition entity classes (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- The intermediate classes in the inheritance hierarchy, namely RegistryObject and RegistryEntry, map to relational views.
- The names of relational tables and views are the same as the corresponding [ebRIM] class name. However, the name binding is case insensitive.
- Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in Appendix D.3 where the column definitions are based on a subset of attributes defined for that class in [ebRIM]. The attributes that map to columns include the inherited attributes for the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed which column definitions.

An SQLQuery against a table not defined in Appendix D.3 may raise an error condition: InvalidQueryException.

The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn definitions.

Primitive Attributes Binding

Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same way as column names in SQL. Again the exact attribute names are defined in the class definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is therefore valid for a query to contain attribute names that do not exactly match the case defined in [ebRIM].

Reference Attribute Binding

A few of the [ebRIM] class attributes are of type ObjectRef and are a reference to an instance of a class defined by [ebRIM]. For example, the sourceObject attribute of the Association class returns a reference to an instance of a RegistryObject.

In such cases the reference maps to the `id` attribute for the referenced object. The name of the resulting column is the same as the attribute name in [ebRIM] as defined by 0. The data type for the column is VARCHAR(64) as defined in Appendix D.3.

When a reference attribute value holds a null reference, it maps to a null value in the SQL binding and may be tested with the <null specification> (“IS [NOT] NULL” syntax) as defined by [SQL].

Reference attribute binding is a special case of a primitive attribute mapping.

Complex Attribute Binding

A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

The SQL query schema does not map complex attributes as columns in the table for the class for which the attribute is defined. Instead the complex attributes are mapped to columns in the table for the domain class that represents the data type for the complex attribute (e.g. TelephoneNumber). A column links the row in the domain table to the row in the parent table (e.g. User). An additional column named 'attribute_name' identifies the attribute name in the parent class, in case there are multiple attributes with the same complex attribute type.

This mapping also easily allows for attributes that are a collection of a complex type. For example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier and an attribute_name.

Binding of Methods Returning Collections

Several of the [ebRIM] classes define methods in addition to attributes, where these methods return collections of references to instances of classes defined by [ebRIM]. For example, the getPackages method of the RegistryObject class returns a Collection of references to instances of Packages that the object is a member of.

Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in Appendix D.3 such that these stored procedures return a collection of `id` attribute values. The returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

These stored procedures may be used as the right-hand-side of an SQL IN clause to test for membership of an object in such collections of references.

8.3.2 Semantic Constraints On Query Syntax

This section defines simplifying constraints on the query syntax that cannot be expressed in the BNF for the query syntax. These constraints *must* be applied in the semantic analysis of the query.

1. Class names and attribute names must be processed in a case insensitive manner.
2. The syntax used for stored procedure invocation must be consistent with the syntax of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
3. For this version of the specification, the SQL select column list consists of exactly one column, and must always be `t.id`, where t is a table reference in the FROM clause.

8.3.3 SQL Query Results

The result of an SQL query resolves to a collection of objects within the registry. It never resolves to partial attributes. The objects related to the result set may be returned as an ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the returnType attribute of the responseOption parameter specified by the client on the AdHocQueryRequest. The entire result set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 8.1.

8.3.4 Simple Metadata Based Queries

The simplest form of an SQL query is based upon metadata attributes specified for a single class

within [ebRIM]. This section gives some examples of simple metadata based queries.

For example, to retrieve the collection of ExtrinsicObjects whose name contains the word 'Acme' and that have a version greater than 1.3, the following query must be submitted:

```
SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
       eo.id = nm.parent AND
       eo.majorVersion >= 1 AND
       (eo.majorVersion >= 2 OR  eo.minorVersion > 3);
```

Note that the query syntax allows for conjugation of simpler predicates into more complex queries as shown in the simple example above.

8.3.5 RegistryObject Queries

The schema for the SQL query defines a view called RegistryObject that allows doing a polymorphic query against all RegistryObject instances regardless of their actual concrete type or table name.

The following example is similar to the example in Section 8.3.4 except that it is applied against all RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id for all qualifying RegistryObject instances whose name contains the word 'Acme' and whose description contains the word "bicycle".

```
SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND
       d.value LIKE '%bicycle%' AND
       ro.id = nm.parent AND ro.id = d.parent;
```

8.3.6 RegistryEntry Queries

The schema for the SQL query defines a view called RegistryEntry that allows doing a polymorphic query against all RegistryEntry instances regardless of their actual concrete type or table name.

The following example is the same as the example in Section 8.3.4 except that it is applied against all RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a version greater than 1.3.

```
SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
       re.id = nm.parent AND
       re.majorVersion >= 1 AND
       (re.majorVersion >= 2 OR  re.minorVersion > 3);
```

8.3.7 Classification Queries

This section describes various classification related queries.

Identifying ClassificationNodes

ClassificationNodes are identified by their "id" attribute, as are all objects in [ebRIM]. However, they may also be identified by their a "path" attribute that specifies an XPATH expression [XPT] from a root classification node to the specified classification node in the XML document that would represent the ClassificationNode tree including the said ClassificationNode.

Retrieving ClassificationSchemes

The following query retrieves the collection of ClassificationSchemes :

```
SELECT scheme.id FROM ClassificationScheme scheme;
```

The above query returns all ClassificationSchemes. Note that the above query may also specify additional predicates (e.g. name, description etc.) if desired.

Retrieving Children of Specified ClassificationNode

The following query retrieves the children of a ClassificationNode given the “id” attribute of that node:

```
SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

The above query returns all ClassificationNodes that have the node specified by <id> as their parent attribute.

Retrieving Objects Classified By a ClassificationNode

The following query retrieves the collection of ExtrinsicObjects classified by specified ClassificationNodes:

```
SELECT id FROM ExtrinsicObject
WHERE
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = "/Geography/Asia/Japan'))
  AND
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = '/Industry/Automotive'))
```

The above query retrieves the collection of ExtrinsicObjects that are classified by the Automotive Industry and the Japan Geography. Note that according to the semantics defined for GetClassifiedObjectsRequest, the query will also contain any objects that are classified by descendents of the specified ClassificationNodes.

Retrieving Classifications That Classify an Object

The following query retrieves the collection of Classifications that classify a specified Object:

```
SELECT id FROM Classification c
       WHERE c.classifiedObject = <id>;
```

8.3.8 Association Queries

This section describes various Association related queries.

Retrieving All Association With Specified Object As Its Source

The following query retrieves the collection of Associations that have the specified Object as its source:


```
SELECT id FROM Association WHERE sourceObject = <id>
```

Retrieving All Association With Specified Object As Its Target

The following query retrieves the collection of Associations that have the specified Object as its target:

```
SELECT id FROM Association WHERE targetObject = <id>
```

Retrieving Associated Objects Based On Association Attributes

The following query retrieves the collection of Associations that have specified Association attributes:

Select Associations that have the specified name.

```
SELECT id FROM Association WHERE name = <name>
```

Select Associations that have the specified association type, where association type is a string containing the corresponding field name described in [ebRIM].

```
SELECT id FROM Association WHERE
       associationType = <associationType>
```

Complex Association Queries

The various forms of Association queries may be combined into complex predicates. The following query selects Associations that have a specific sourceObject, targetObject and associationType:

```
SELECT id FROM Association WHERE
       sourceObject = <id1> AND
       targetObject = <id2> AND
       associationType = <associationType>;
```

8.3.9 Package Queries

The following query retrieves all Packages that a specified RegistryObject belongs to:

```
SELECT id FROM Package WHERE id IN (RegistryObject_registryPackages(<id>));
```

Complex Package Queries

The following query retrieves all Packages that a specified object belongs to, that are not deprecated and where name contains "RosettaNet."

```
SELECT id FROM Package p, Name n WHERE
       p.id IN (RegistryObject_registryPackages(<id>)) AND
       nm.value LIKE '%RosettaNet%'  AND nm.parent = p.id AND
       p.status <> 'Deprecated'
```

8.3.10 ExternalLink Queries

The following query retrieves all ExternalLinks that a specified ExtrinsicObject is linked to:


```
SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
```

The following query retrieves all ExtrinsicObjects that are linked by a specified ExternalLink:

```
SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
```

Complex ExternalLink Queries

The following query retrieves all ExternalLinks that a specified ExtrinsicObject belongs to, that contain the word 'legal' in their description and have a URL for their externalURI.

```
SELECT id FROM ExternalLink WHERE
       id IN (RegistryObject_externalLinks(<id>)) AND
       description LIKE '%legal%' AND
       externalURI LIKE '%http://%'
```

8.3.11 Audit Trail Queries

The following query retrieves all the AuditableEvents for a specified RegistryObject:

```
SELECT id FROM AuditableEvent WHERE registryObject = <id>
```

8.3.12 Object Export Queries

The standard Ad hoc Query protocol may be used to export RegistryObjects from a registry.

Export Objects Owned By Specified User

The following query retieves all RegistryObjects for a specified User id:

```
SELECT * from RegistryObject ro, AuditableEvent ae, User u WHERE
       ae.user_ = <userId> AND
       ae.eventType = 'Created' AND
       ae.registryObject = ro.id
```

Export Objects Owned By Users Affiliated With Specified Organization

The following query retieves all RegistryObjects that are owned by any User that is affiliated with the Organization matching the specied name pattern:

```
SELECT * from RegistryObject ro, AuditableEvent ae, User u WHERE
    ae.user_ = u.id AND ae.eventType = 'Created' AND ae.registryObject = ro.id AND
    u.id IN (
        SELECT u1.id from User u1, Organization o, Name n WHERE
            n.value LIKE '%Sun%' AND u1.organization = o.id AND n.parent = o.id
    )
```

8.4 Content Retrieval

A client retrieves content via the Registry by sending the GetContentRequest to the QueryManager. The GetContentRequest specifies a list of ObjectRefs for Objects that need to be retrieved. The QueryManager returns the specified content by sending a GetContentResponse message to the RegistryClient interface of the client.

If there are no errors encountered, the GetContentResponse message includes the specified

content(s) as mime multipart attachment(s) within the message.

If there are errors encountered, the RegistryResponse payload includes the errors and there are no additional mime multipart attachments.

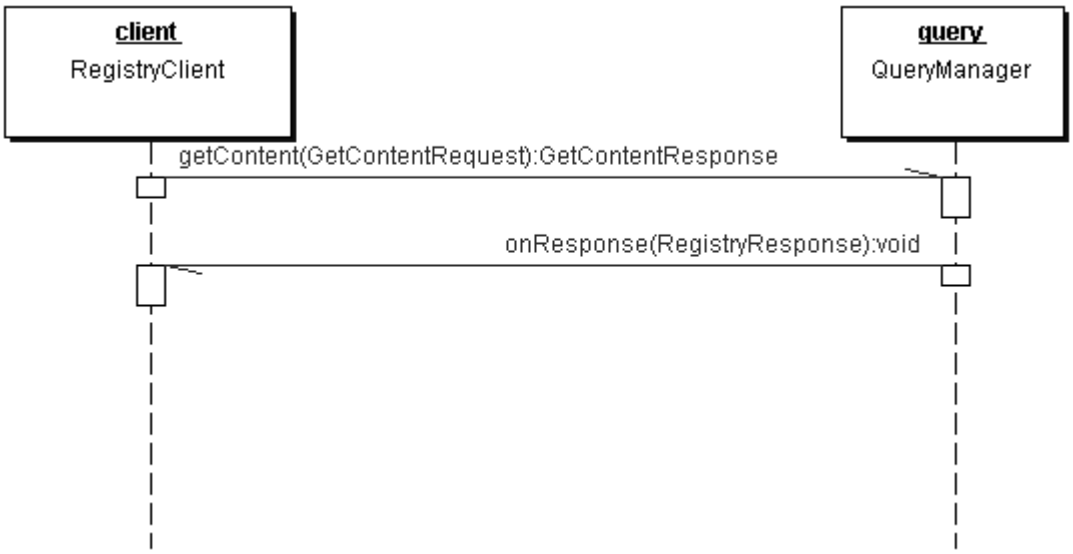


Figure 53: Content Retrieval Sequence Diagram

8.4.1 GetContentRequest

The GetContentRequest is used to retrieve repository item content from the registry.

Syntax:

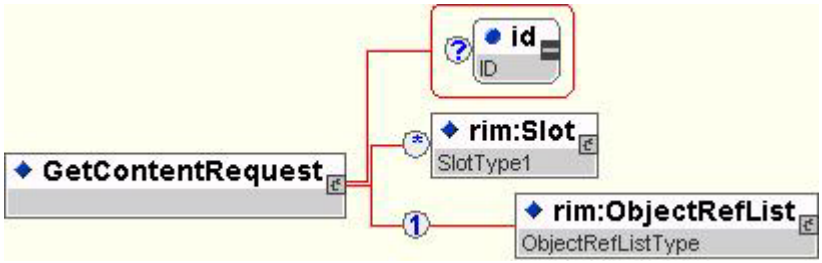


Figure 54: GetContentRequest Syntax

Parameters:

- *ObjectRefList:* This parameter specifies a collection of ObjectRef elements that specify references to the ExtrinsicObjects whose corresponding repository items are being retrieved.

Returns:

This request returns a GetContentResponse. See section 8.4.2 for details.

Exceptions:

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

- *ObjectNotFoundException:* signifies that one or more ObjectRef elements in the ObjectRefList did not match any objects in the registry.

8.4.2 GetContentResponse

The GetContentResponse is sent by the registry as a response to GetContentRequest.

Syntax:

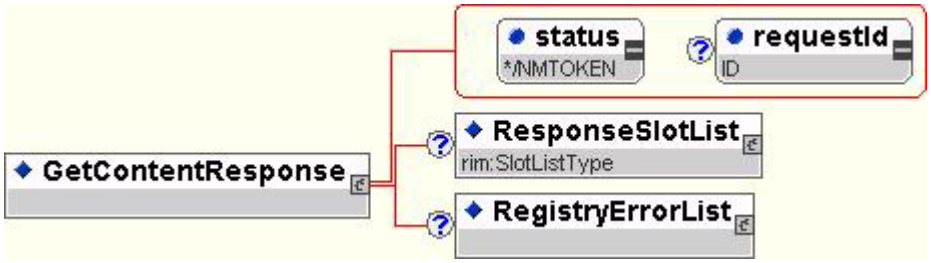


Figure 55: GetContentResponse Syntax

Parameters:

The GetContentResponse does not define any new parameters beyond those inherited by RegistryResponseType as defined in 6.9.2.

8.4.3 Identification Of Content Payloads

Since the GetContentResponse message may include several repository items as additional payloads, it is necessary to have a way to identify each mime multipart attachment in the message. To facilitate this identification, the Registry must do the following:

- Use the "id" attribute of the ExtrinsicObject instance as the value of the Content-ID header parameter for the mime multipart that contains the corresponding repository item for the ExtrinsicObject.
- In case of [ebMS] transport, use the "id" attribute of the ExtrinsicObject instance in the Reference element for that object in the Manifest element of the ebXMLHeader.

8.4.4 GetContentResponse Message Structure

The following message fragment illustrates the structure of the GetContentResponse Message that is returning a Collection of Collaboration Protocol Profiles as a result of a GetContentRequest that specified the "id" attributes for the requested objects. Note that the boundary parameter in the Content-Type headers in the example below are meant to be illustrative not prescriptive.

```
Content-type: multipart/related; boundary="MIME_boundary"; type="text/xml";

--MIME_boundary
Content-ID: <GetContentRequest@example.com>
```



```
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:eb= 'http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
   <SOAP-ENV:Header>

      <!--ebMS header goes here if using ebMS-->
      ...

   <ds:Signature ...>
      <!--signature over soap envelope-->
      ...
   </ds:Signature>

   </SOAP-ENV:Header>

   <SOAP-ENV:Body>

      <!--ebMS manifest goes here if using ebMS-->
      ...

      <?xml version="1.0" encoding="UTF-8"?>

      <GetContentResponse>
         <ObjectRefList>
            <ObjectRef id="urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff"/>
            <ObjectRef id="urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f"/>
         </ObjectRefList>
      </GetContentResponse>

   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--MIME_boundary

Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
Content-Type: Multipart/Related; boundary=payload1_boundary; type=text/xml
Content-Description: Optionally describe payload1 here

--payload1_boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: signature:urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff

<ds:Signature ...>
   ... Signature for payload1
</ds:Signature>

--payload1_boundary
Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<tp:CollaborationProtocolProfile ...>
   .....
</tp:CollaborationProtocolProfile>
--payload1_boundary--

--MIME_boundary

Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
Content-Type: Multipart/Related; boundary=payload2_boundary; type=text/xml
Content-Description: Optionally describe payload2 here

--payload2_boundary
Content-Type: text/xml; charset=UTF-8
Content-ID: signature:urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f

<ds:Signature ...>
   ... Signature for payload2
</ds:Signature>
--payload2_boundary
Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
```



```
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<tp:CollaborationProtocolProfile ...>
   .....
</tp:CollaborationProtocolProfile>

--payload2_boundary--


--MIME_boundary--
```

9 Content Management Services

This chapter describes the Content Management services of the ebXML Registry. Examples of Content Management Services include, but are not limited to, content validation and content cataloging. Content Management Services result in improved quality and integrity of registry content and metadata as well as improved ability for clients to discover that content and metadata.

The Content Management Services facility of the registry is based upon a pluggable architecture that allows clients to publish and discover new Content Management Services as Service objects that conform to a normative web service interface specified in this chapter. Clients may define a Content Management Services that is specialized for managing a specific type of content.

The Content Management Services facility as a whole is an optional normative feature of ebXML Registries compliant to version 3 or later of this specification. Note however that some aspects of the Content Management Services facility are required normative features of ebXML Registries.

9.1 Content Validation

The Content Validation feature provides the ability to enforce validation rules upon submitted content and metadata in a content specific manner.

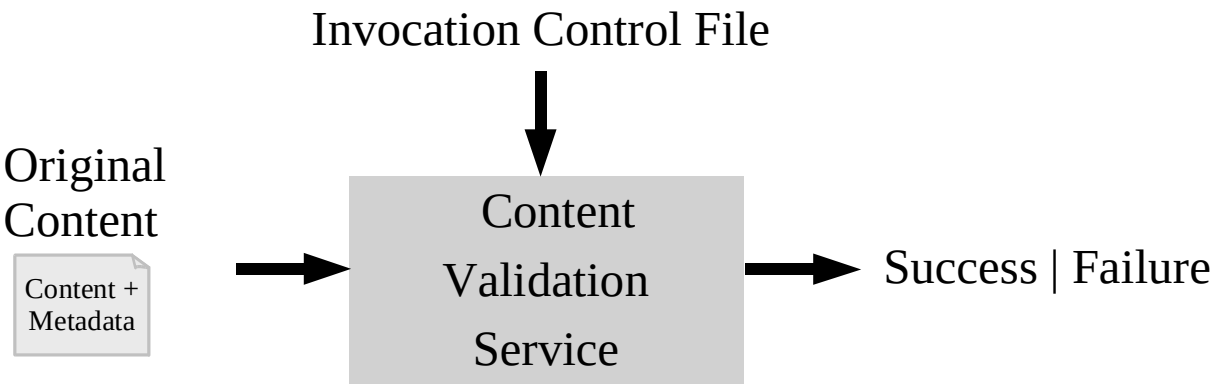


Figure 56: Content Validation Service

A registry uses one or more Content Validation Services to automatically validate the RegistryObjects and repository items when they are submitted to the registry. A registry must reject a submission request in its entirety if it contains invalid data. In such cases a ValidationException must be returned to the client.

Content Validation feature improves the quality of data in the registry.

9.1.1 Content Validation: Use Cases

The following use cases illustrates use cases of the Content Validation feature:

Validation of HL7 Conformance Profiles

The Healthcare Standards organization HL7 uses content validation to enforce consistency rules and semantic checks whenever an HL7 member submits an HL7 Conformance Profile. HL7 is also planning to use the feature to improve the quality of other types of HL7 artifacts.

Validation of Business Processes

Content validation may be used to enforce consistency rules and semantic checks whenever a Business Process is submitted to the registry. This feature may be used by organizations such as UN/CEFACT, OAG, and RosettaNet.

Validation of UBL Business Documents

Content validation may be used by the UBL technical committee to enforce consistency rules and semantic checks whenever a UBL business document is submitted to the registry.

9.2 Content Cataloging

The Content Cataloging feature provides the ability to selectively convert submitted RegistryObject and repository items into metadata defined by [ebRIM], in a content specific manner.

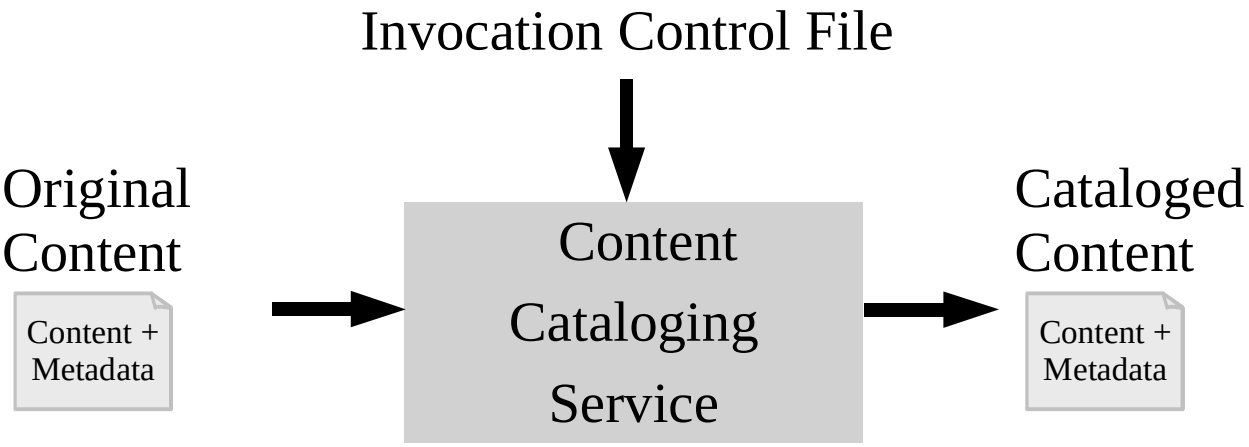


Figure 57: Content Cataloging Service

A registry uses one or more Content Cataloging Services to automatically catalog RegistryObjects and repository items. Cataloging creates and/or updates RegistryObject metadata such as ExtrinsicObject or Classification instances. The cataloged metadata enables clients to discover the repository item based upon content from the repository item, using standard query capabilities of the registry. This is referred to as *Content-based Discovery*. The main benefit of the Content Cataloging feature is to enable Content-based Discovery.

9.2.1 Content-based Discovery: Use Cases

There are many scenarios where content-based discovery is necessary.

Find All CPPs Where Role is "Buyer"

A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to find CPPs for other companies where the Role element of the CPP is that of "Buyer".

Find All XML Schema's That Use Specified Namespace

A client may wish to discover all XML Schema documents in the registry that use an XML namespace containing the word "oasis".

Find All WSDL Descriptions with a SOAP Binding

An ebXML registry client is attempting to discover all repository items that are WSDL descriptions that have a SOAP binding defined. Note that SOAP binding related information is content within the WSDL document and not metadata.

9.3 Abstract Content Management Service

This section describes in abstract terms how the registry supports pluggable, user-defined Content Management Services. A Content Management Service is invoked in response to content being submitted to the registry via the standard Submit/UpdateObjectsRequest method. The Service invocation is on a per request basis where one request may result in many invocations, one for each RegistryObject for which a Content Management Service is configured within the registry.

The registry may perform such invocation in one of two ways.

- *Inline Invocation Model*: Content Management Service may be invoked inline with the processing of the Submit/UpdateObjectsRequest and prior to committing the content. This is referred to as Inline Invocation Model.
- *De-coupled Invocation Model*: Content Management Service may be invoked de-coupled from the processing of the Submit/UpdateObjectsRequest and some time after committing the content. This is referred to as De-coupled Invocation Model.

9.3.1 Inline Invocation Model

In an inline invocation model a registry must invoke a Content Management Service inline with Submit/UpdateObjectsRequest processing and prior to committing the Submit/UpdateObjectsRequest. All metadata and content from the original Submit/UpdateObjectsRequest request or from the Content Management Service invocation must be committed as an atomic transaction.

Figure 58 shows an abstract Content Management Service and how it is used by an ebXML Registry using an inline invocation model. The steps are as follows:

1. A client submits a Content Management Service S1 to an ebXML Registry. The client typically belongs to an organization responsible for defining a specific type of content. For example the client may belong to RosettaNet.org and submits a Content Validation Service for validating RosettaNet PIPs. The client uses the standard Submit/UpdateObjectsRequest interface to submit the Service. This is a one-time step to configure this Content Management Service in the registry.
2. Once the Content Management Service has been submitted, a potentially different client may submit content to the registry that is of the same object type for which the Content Management Service has been submitted. The client uses the standard Submit/UpdateObjectsRequest interface to submit the content.
3. The registry determines there is a Content Management Service S1 configured for the object type for the content submitted. It invokes S1 using a ContentManagementServiceRequest and passes it the content.

4. The Content Management Service S1 processes the content and sends back a ContentManagementServiceResponse.
5. The registry then commits the content to the registry if there are no errors encountered.
6. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest in step 2.

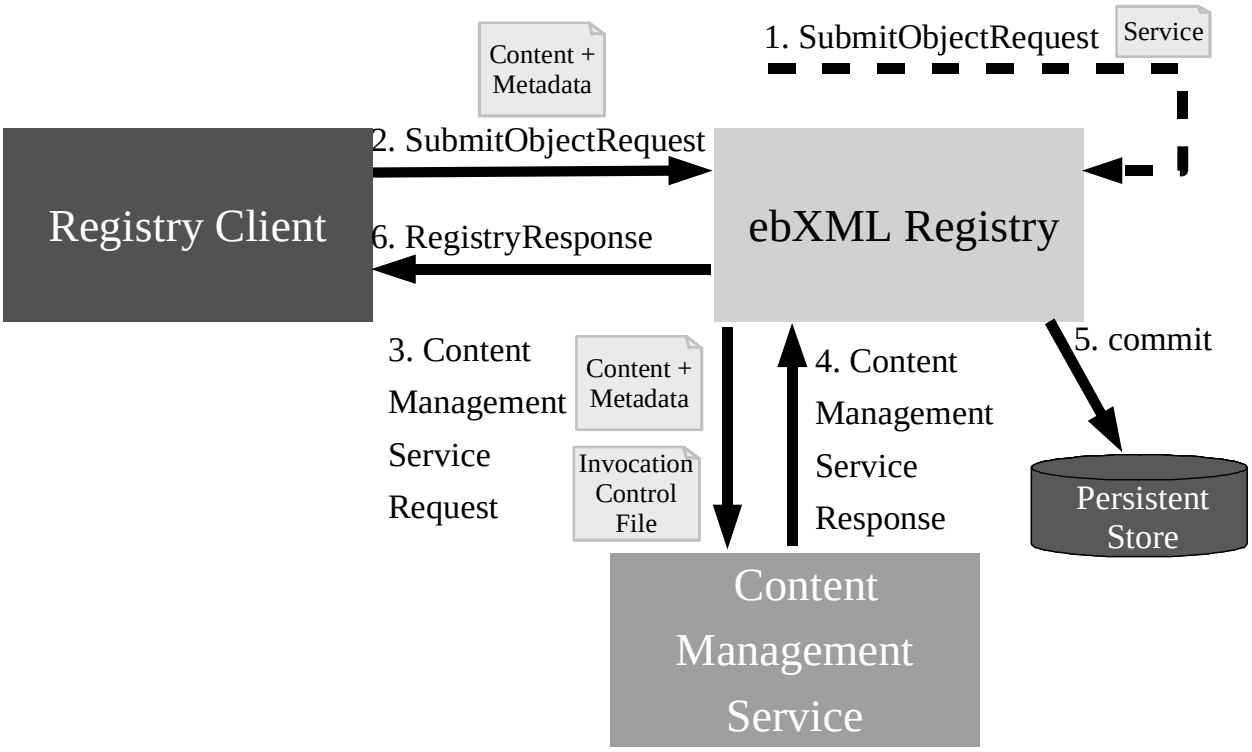


Figure 58: Content Management Service: Inline Invocation Model

9.3.2 De-coupled Invocation Model

In a de-coupled invocation model a registry must invoke a Content Management Service independent of or de-coupled from the Submit/UpdateObjectsRequest processing. Any errors encountered during Content Management Service invocation must not have any impact on the original Submit/UpdateObjectsRequest processing.

All metadata and content from the original Submit/UpdateObjectsRequest request must be committed as an atomic transaction that is decoupled from the metadata and content that may be generated by the Content Management Service invocation.

Figure 60 shows an abstract Content Management Service and how it is used by an ebXML Registry using a de-coupled invocation model. The steps are as follows:

1. Same as in inline invocation model (Content Management Service is submitted).
2. Same as in inline invocation model (client submits content using Submit/UpdateObjectsRequest).

3. The registry processes the Submit/UpdateObjectsRequest and commits it to persistent store.
4. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest in step 2.
5. The registry determines there is a Content Management Service S1 configured for the object type for the content submitted. It invokes S1 using a ContentManagementServiceRequest and passes it the content.
6. The Content Management Service S1 processes the content and sends back a ContentManagementServiceResponse.
7. If the ContentManagementServiceResponse includes any generated or modified content it is committed to the persistent store as separate transaction. If there are any errors encountered during de-coupled invocation of a Content Management Service then these errors are logged by the registry in a registry specific manner and must not be reported back to the client.

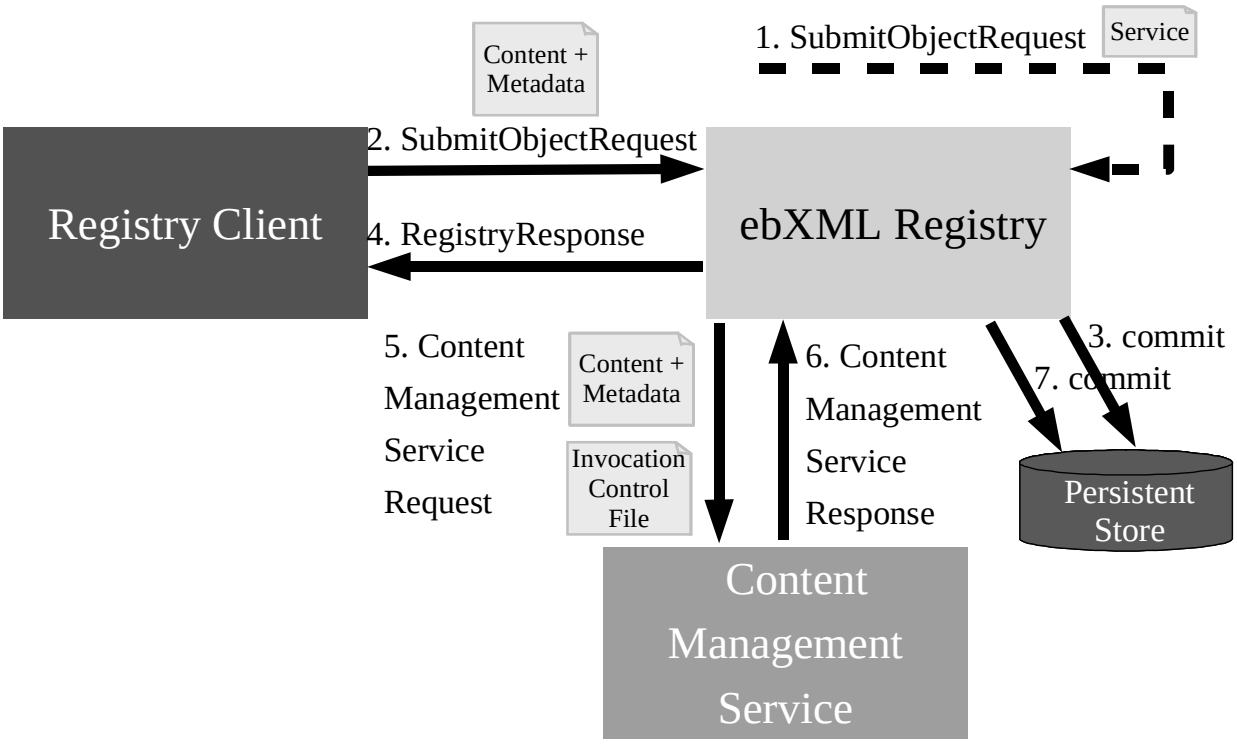


Figure 60: Content Management Service: De-coupled Invocation Model

9.4 Content Management Service Protocol

This section describe the abstract Content Management Service protocol that is the base-protocol for other concrete protocols such as Validate Content protocol and Catalog Content protocol. The concrete protocols will be defined later in this document.

9.4.1 ContentManagementServiceRequestType

The ContentManagementServiceRequestType must be the abstract base type for all requests sent from a registry to a Content Management Service.

Syntax:

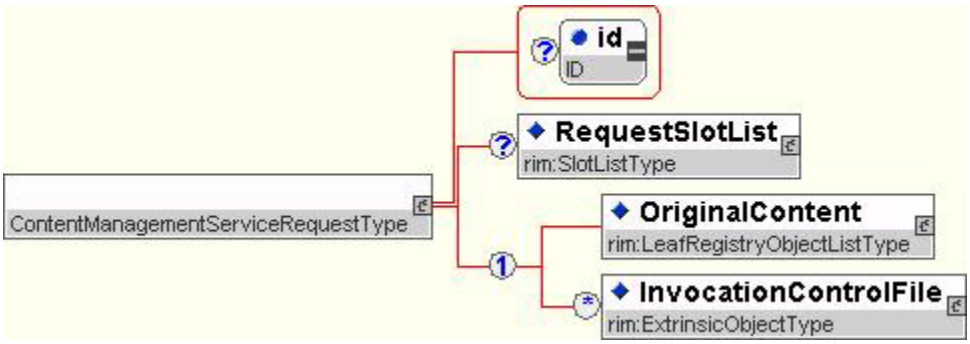


Figure 62: ContentManagementServiceRequestType Syntax

Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

- *InvocationControlFile*: This parameter specifies the ExtrinsicObject for a repository item that the caller wishes to specify as the Invocation Control File. This specification does not specify the format of this file. There must be a corresponding repository item as an attachment to this request. The corresponding repository item should follow the same rules as attachments in Submit/UpdateObjectsRequest.
- *OriginalContent:* This parameter specifies the RegistryObjects that will be processed by the content management service. In case of ExtrinsicObject instances within the OriginalContent there may be repository items present as attachments to the ContentManagementServiceRequest. This specification does not specify the format of such repository items. The repository items should follow the same rules as attachments in Submit/UpdateObjectsRequest.

Returns:

This request returns a ContentManagementServiceResponse. See section 9.4.2 for details.

Exceptions:

In addition to the exceptions returned by base request types, the following exceptions may be returned:

- *MissingRepositoryItemException:* signifies that the caller did not provide a repository item as an attachment to this request when the Service requires it.
- *InvocationControlFileException:* signifies that the InvocationControlFile(s) provided by the caller do not match the InvocationControlFile(s) expected by the Service.
- *UnsupportedContentException:* signifies that this Service does not support the content provided by the caller.

9.4.2 ContentManagementServiceResponseType

The ContentManagementServiceResponseType is sent by a Content Management Service as a response to a ContentManagementServiceRequestType. The ContentManagementServiceResponseType is the abstract base type for all responses sent to a registry from a Content Management Service. It extends the RegistryResponseType and does not define any new parameters.

Syntax:

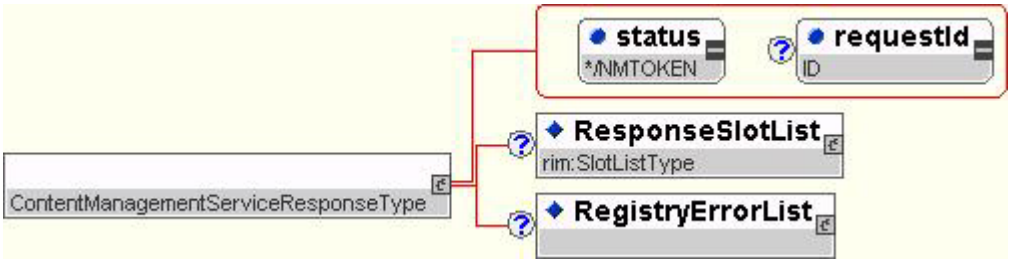


Figure 63: Content ContentManagementServiceResponseType Syntax

Parameters:

No new parameters are defined other than those inherited from RegistryResponseType.

9.5 Publishing / Configuration of a Content Management Service

Any publisher may publish an arbitrary Content Management Service to an ebXML Registry. The Content Management Service must be published using the standard LifeCycleManager interface.

The publisher must use the standard Submit/UpdateObjectsRequest to publish:

- A Service instance for the Content Management Service. In Figure 64 this is exemplified by the defaultXMLCatalogingService in the upper-left corner. The Service instance must have an Association with a ClassificationNode in the canonical ObjectType ClassificationScheme as defined by [ebRIM]. The Service must be the sourceObject while a ClassificationNode must be the targetObject. This association binds the Service to that specific ObjectType. The associationType for this Association instance must be "ContentManagementServiceFor". The Service must be classified by the canonical ContentManagementService ClassificationScheme as defined by [ebRIM]. For example it may be classified as a "ContentValidationService" or a "ContentCatalogingService".
- The Service instance may be classified by a ClassificationNode under the canonical InvocationModel ClassificationScheme as defined by [ebRIM], to determine whether it uses the Inline Invocation model or the De-coupled Invocation model.
- The Service instance may be classified by a ClassificationNode under the canonical ErrorHandlingModel ClassificationScheme as defined by [ebRIM], to determine whether the Service should fail on first error or simply log the error as warning and continue. See 9.6.4 section for details.

- A ServiceBinding instance contained within the Service instance that must provide the accessURI to the Cataloging Service.
- An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page describing:
  - The format of the supported content to be Cataloged
  - The format of the supported Invocation Control File

  Note that no SpecificationLink is required since this specification [ebRS] is implicit for Content Cataloging Services.
- One or more Invocation Control File(s) that must be an ExtrinsicObject and a repository item pair. The ExtrinsicObject for the Invocation Control File must have a required Association with associationType of "InvocationControlFileFor". This is exemplified by the cppCatalogingServiceXSLT and the oagBODCatalogingServiceXSLT objects in Figure 64 (left side of picture). The Invocation Control File must be the sourceObject while a ClassificationNode in the canonical ObjectType ClassificationScheme must be the targetObject.

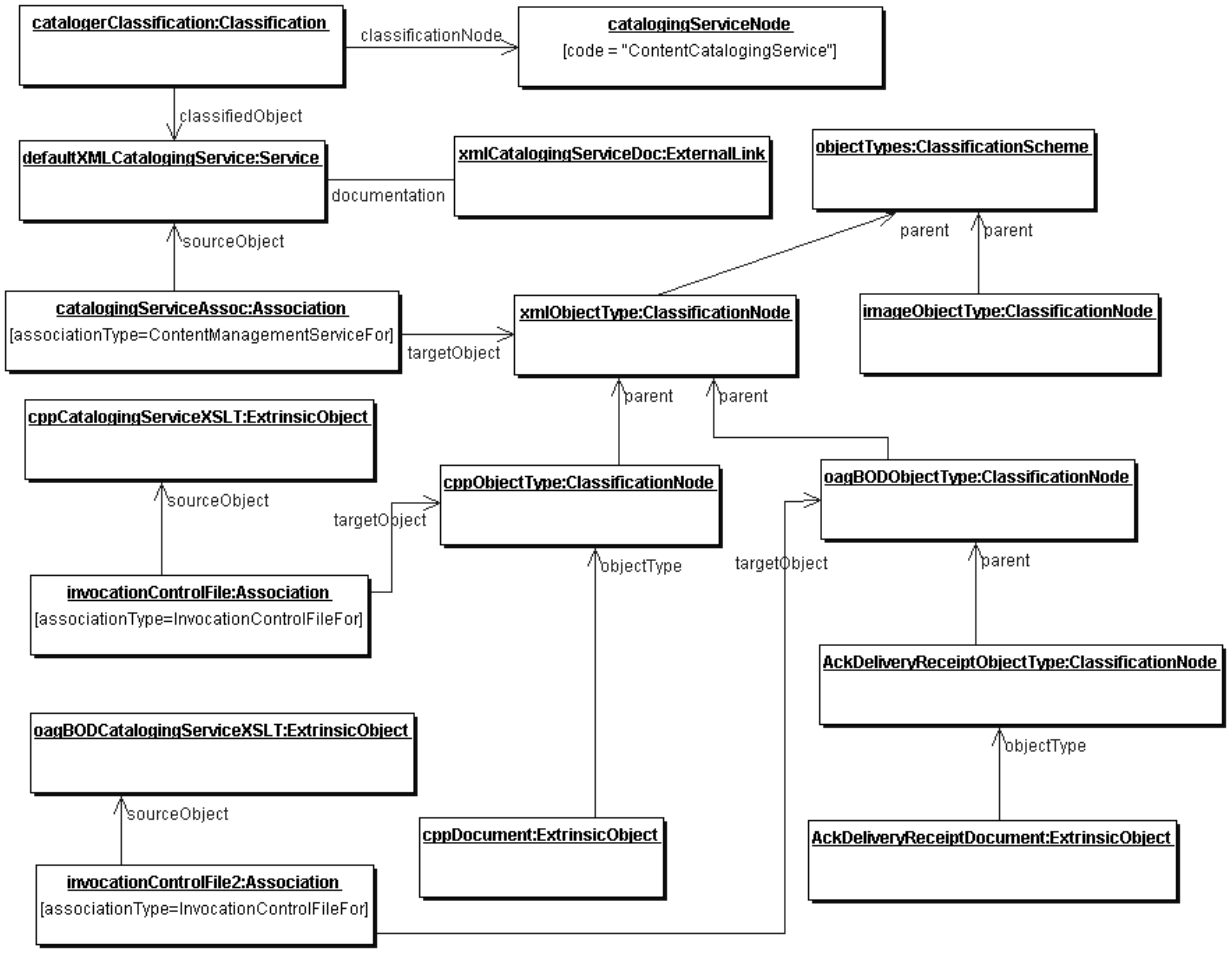


Figure 64: Cataloging Service Configuration

Figure 64 shows an example of the configuration of the default XML Cataloging Service associated with the objectType for XML content. This Cataloging Service may be used with any XML content that has its objectType attribute hold a reference to the xmlObjectType ClassificationNode or one of its descendents.

The figure also shows two different Invocation Control Files, cppCatalogingServiceXSLT and oagBODCatalogingServiceXSLT that may be used to catalog ebXML CPP and OAG Business Object Documents (BOD) respectively.

9.5.1 Multiple Content Management Services and Invocation Control Files

This specification allows clients to submit multiple Content Management Services of the same type (e.g. validation, cataloging) and multiple Invocation Control Files for the same objectType. Content Management Services of the same type of service for the same ObjectType are referred to as peer Content Management Services.

When there are multiple Content Management Services and Invocation Control Files for the same ObjectType there must be an unambiguous association between a Content Management Service and its Invocation Control File(s). This must be defined by an Association instance with Association type of "InvocationControlFileFor" where the ExtrinsicObject for each Invocation Control File is the sourceObject and the Service is the targetObject.

The order of invocation of peer Content Management Services is undefined and may be determined in a registry specific manner.

9.6 Invocation of a Content Management Service

This section describes how a registry invokes a Content Management Service.

9.6.1 Resolution Algorithm For Service and Invocation Control File

When a registry receives a submission of a RegistryObject, it must use the following algorithm to determine or resolve the Content Management Services and Invocation Control Files to be used for dynamic content management for the RegistryObject:

1. Get the objectType attribute of the RegistryObject.
2. Query to see if the ClassificationNode referenced by the objectType is the targetObject of an Association with associationType of *ContentManagementServiceFor*. If desired Association is not found for this ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired Association is found or until the parent is the ClassificationScheme. If desired Association(s) is found then repeat following steps for each such Association instance.
3. Check if the sourceObject of the desired Association is a Service instance. If not, log an InvalidConfigurationException. If it is a Service instance, then use this Service as the Content Management service for the RegistryObject.

4. Query to see if the objectType ClassificationNode is the targetObject of one or more Association with associationType of *InvocationControlFileFor*. If desired Association is not found for this ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired Association is found or until the parent is the ClassificationScheme.
5. If desired Association(s) are found then check if the sourceObject of the desired Association is an ExtrinsicObject instance. If not, log an InvalidConfigurationException. If sourceObject is an ExtrinsicObject instance, then use its repository item as an Invocation Control File. If there are multiple InvocationControlFiles then all of them must be provided when invoking the Service.

The above algorithm allows for objectType hierarchy to be used to configure Content Management Services and Invocation Control Files with varying degrees of specificity or specialization with respect to the type of content.

9.6.2 Audit Trail and Cataloged Content

The Cataloged Content generated as a result of the invocation of a Content Management Service has an audit trail consistent with RegistryObject instances that are submitted by Registry Clients. However, since a Registry Client does not submit Cataloged Content, the user attribute of the AuditableEvent instances for such Cataloged Content references the Service object for the Content Management Service that generated the Cataloged Content. This allows an efficient way to distinguish Cataloged Content from content submitted by Registry Clients.

9.6.3 Referential Integrity

A registry must maintain referential integrity between the RegistryObjects and repository items invocation of a Content Management Service.

9.6.4 Error Handling

If the Content Management Service is classified by the “FailOnError” ClassificationNode under canonical ErrorHandlingModel ClassificationScheme as defined by [ebRIM], then the registry must stop further processing of the Submit/UpdateObjectsRequest and return status of “Failure” upon first error returned by a Content Management Service Invocation.

If the Content Management Service is classified by the “LogErrorAndContinue” ClassificationNode under ErrorHandlingModel then the registry must continue to process the Submit/UpdateObjectsRequest and not let any Content Management Service invocation error to affect the storing of the RegistryObjects and repository items that were submitted. Such errors should be logged as Warnings within the RegistryResponse returned to the client. In this case a registry must return a normal response with status = “Success” if the submitted content and metadata is stored successfully even when there are errors encountered during dynamic invocation of one or more Content Management Service.

9.7 Validate Content Protocol

The interface of a Content Validation Service must implement a single method called validateContent. The validateContent method accepts a ValidateContentRequest as parameter and returns a ValidateContentResponse as its response if there are no errors.

The OriginalContent element within a ValidateContentRequest must contain exactly one RegistryObject that needs to be cataloged. The resulting ValidateContentResponse contains the status attribute that communicates whether the RegistryObject (and its content) are valid or not. The Validate Content protocol does not specify the implementation details of any specific Content Validation Service.

Figure 66: Validate Content Protocol

9.7.1 ValidateContentRequest

The ValidateContentRequest is used to pass content to a Content Validation Service so that it can validate the specified RegistryObject and any associated content. The RegistryObject typically is an ExternalLink (in case of external content) or an ExtrinsicObject. The ValidateContentRequest extends the base type ContentManagementServiceRequestType.

Syntax:

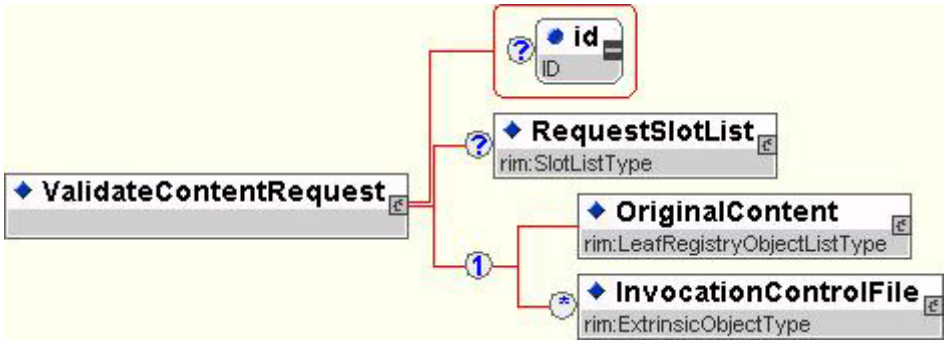


Figure 67: ValidateContentRequest Syntax

Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

- *InvocationControlFile*: Inherited from base type. This parameter may not be

present. If present its format is defined by the Content Validation Service.

- *OriginalContent:* Inherited from base type. This parameter must contain exactly one RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content. This specification does not specify the format of the content. If it is an ExtrinsicObject then there may be a corresponding repository item as an attachment to this request that is the content. The corresponding repository item should follow the same rules as attachments in Submit/UpdateObjectsRequest.

Returns:

This request returns a ValidateContentResponse. See section 9.7.2 for details.

Exceptions:

In addition to the exceptions returned by base request types, the following exceptions may be returned:

- *InavlidContentException:* signifies that the specified content was found to be invalid. The exception should include enough detail for the client to be able to determine how to make the content valid.

9.7.2 ValidateContentResponse

The ValidateContentResponse is sent by the Content Validation Service as a response to a ValidateContentRequest.

Syntax:

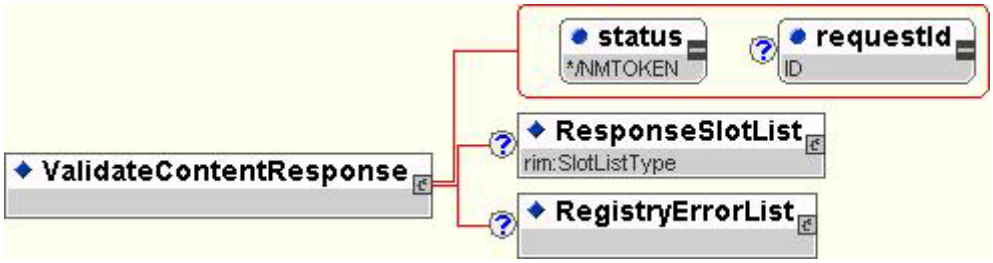


Figure 68: ValidateContentResponse Syntax

Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

- *status*: Inherited attribute. This enumerated value is used to indicate the status of the request. Values for status are as follows:

  - Success - This status specifies that the content specified in the ValidateContentRequest was valid.

- Failure - This status specifies that the request failed. If the error returned is an InvalidContentException then the content specified in the ValidateContentRequest was invalid. If there was some other failure encountered during the processing of the request then a different error may be returned.

9.8 Catalog Content Protocol

The interface of the Content Cataloging Service must implement a single method called catalogContent. The catalogContent method accepts a CatalogContentRequest as parameter and returns a CatalogContentResponse as its response if there are no errors.

The CatalogContentRequest may contain repository items that need to be cataloged. The resulting CatalogContentResponse contains the metadata and possibly content that gets generated or updated by the Content Cataloging Service as a result of cataloging the specified repository items.

The Catalog Content protocol does not specify the implementation details of any specific Content Cataloging Service.

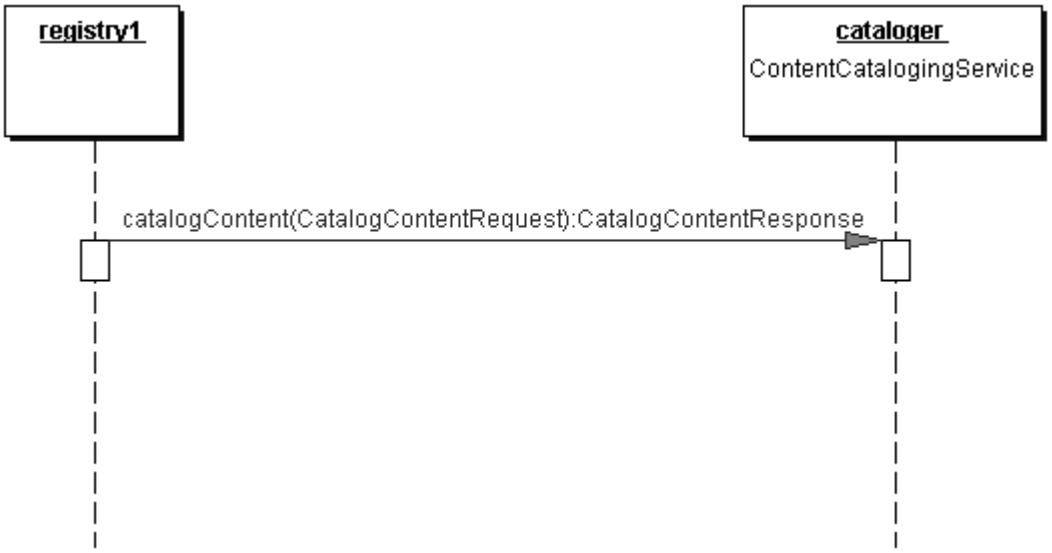


Figure 69: Catalog Content Protocol

9.8.1 CatalogContentRequest

The CatalogContentRequest is used to pass content to a Content Cataloging Service so that it can create catalog metadata for the specified RegistryObject and any associated content. The RegistryObject typically is an ExternalLink (in case of external content) or an ExtrinsicObject. The CatalogContentRequest extends the base type ContentManagementServiceRequestType.

Syntax:

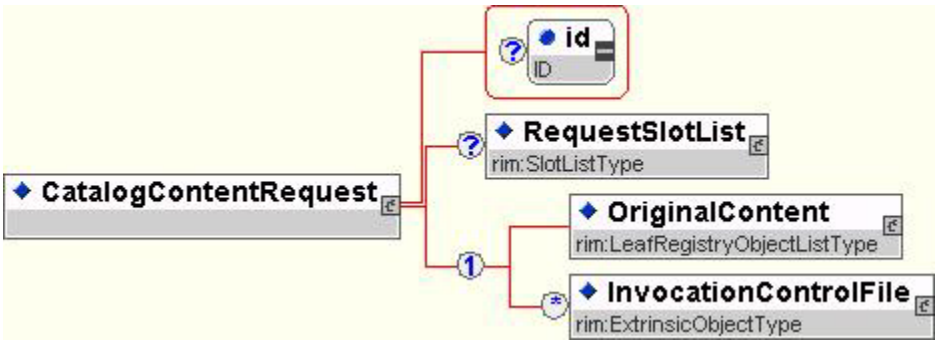


Figure 70: CatalogContentRequest Syntax

Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

- *InvocationControlFile*: Inherited from base type. If present its format is defined by the Content Cataloging Service.
- *OriginalContent:* Inherited from base type. This parameter must contain exactly one RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content. This specification does not specify the format of the content. If it is an ExtrinsicObject then there may be a corresponding repository item as an attachment to this request that is the content. The corresponding repository item should follow the same rules as attachments in Submit/UpdateObjectsRequest.

Returns:

This request returns a CatalogContentResponse. See section 9.8.2 for details.

Exceptions:

In addition to the exceptions returned by base request types, the following exceptions may be returned:

- *CatalogingException:* signifies that an exception was encountered in the Cataloging algorithm for the service.

9.8.2 CatalogContentResponse

The CatalogContentResponse is sent by the Content Cataloging Service as a response to a CatalogContentRequest.

Syntax:

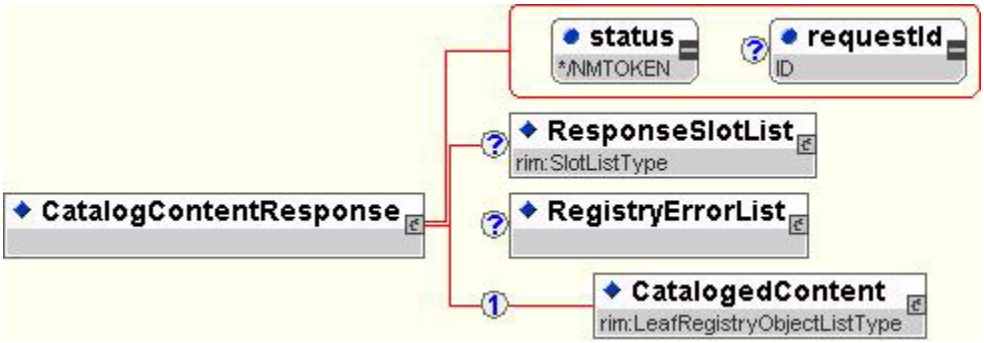


Figure 71: CatalogContentResponse Syntax

Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

- *CatalogedContent:* This parameter specifies a collection of RegistryObject instances that were created or updated as a result of dynamic content cataloging by a content cataloging service. The Content Cataloging Service may add metadata such as Classifications, ExternalIdentifiers, name, description etc. to the CatalogedContent element. There *may* be an accompanying repository item as an attachment to this response message if the original repository item was modified by the request.

9.9 Illustrative Example: Default XML Cataloging Service

Figure 72 shows a UML instance diagram to illustrate how a Content Cataloging Service is used. This Content Cataloging Service is the normative Default XML Cataloging Service described in section 9.10.

- In the center we see a Content Cataloging Service name defaultXMLCataloger Service.
- On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP being input as Original Content to the defaultXMLCataloging Service.
- On top we see an XSLT style sheet repository item and its ExtrinsicObject that is configured as an Invocation Control File for the defaultXMLCataloger Service.
- On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for the CPP. We also see a Classification roleClassification, which classifies the CPP by the Role element within the CPP. These are the Cataloged Content generated as a result of the Cataloging Service cataloging the CPP.

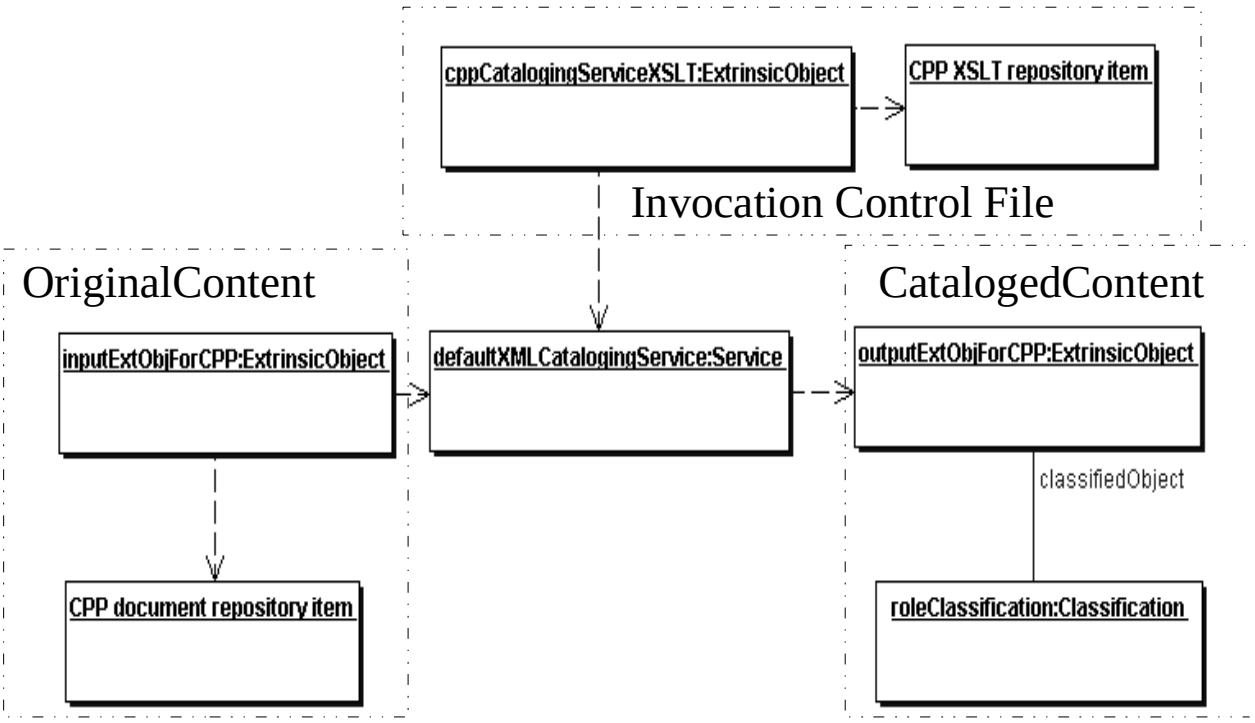


Figure 72: Example of CPP cataloging using Default XML Cataloging Service

9.10 Default XML Content Cataloging Service

An ebXML Registry must provide the default XML Content Cataloging Service natively as a built-in service with the following constraints:

- There is exactly one Service instance for the Default XML Content Cataloging Service
- The Service is an XSLT engine
- The Service may be invoked with exactly one Invocation Control File
- The Original Content for the Service must be XML document(s)
- The Cataloged Content for the Service must be XML document(s)
- The Invocation Control File must be an XSLT style sheet
- Each invocation of the Service may be with different Invocation Control File (XSLT style sheet) depending upon the objectType of the RegistryObject being cataloged. Each objectType should have its own unique XSLT style sheet. For example, ebXML CPP documents should have a specialized ebXML CPP Invocation Control XSLT style sheet.
- The Service must have at least one input XML document that is a RegistryObject. Typically this is an ExtrinsicObject or an ExternalLink.
- The Service may have at most one additional input XML document that is the content represented by the RegistryObject (e.g. a CPP document or an HL7 Conformance Profile). The optional second input must be referenced within the XSLT Style sheet by a using the “document” function with the document name specified by variable “repositoryItem” as in “document($repositoryItem)”. A registry must define the variable “repositoryItem” when invoking the default XML Cataloging Service.

- The default XML Content Cataloging Service must apply the XSLT style sheet to the input XML instance document(s) in an XSLT transformation to generate the Cataloged Output.

The Default XML Content Cataloging Service is a required normative feature of an ebXML Registry.

9.10.1 Publishing of Default XML Content Cataloging Service

An ebXML Registry must provide the default XML Content Cataloging Service natively as a built-in service. This built-in service must be published to the registry as part of the intrinsic bootstrapping of required data within the registry.

10 Event Notification Service

This chapter defines the Event Notification feature of the OASIS ebXML Registry. The Event Notification feature is an optional but normative feature of the ebXML Registry.

Event Notification feature allows OASIS ebXML Registries to notify its users and / or other registries about events of interest. It allows users to stay informed about registry events without being forced to periodically poll the registry. It also allows a registry to propagate internal changes to other registries whose content might be affected by those changes.

ebXML registries support content-based Notification where interested parties express their interest in form of a query. This is different from subject–based (sometimes referred to as topic-based) notification, where information is categorized by subjects and interested parties express their interests in those predefined subjects.

10.1 Use Cases

The following use cases illustrate different ways in which ebXML registries notify users or other registries.

10.1.1 CPP Has Changed

A user wishes to know when the CPP [ebCPP] of her partner is updated or superceded by another CPP. When that happens she may wish to create a CPA [ebCPP] based upon the new CPP.

10.1.2 New Service is Offered

A user wishes to know when a new Plumbing service is offered in her town and be notified every 10 days. When that happens, she might try to learn more about that service and compare it with her current Plumbing service provider's offering.

10.1.3 Monitor Download of Content

User wishes to know whenever her CPP [ebCPP] is downloaded in order to evaluate on an ongoing basis the success of her recent advertising campaign. She might also want to analyze who the interested parties are.

10.1.4 Monitor Price Changes

User wishes to know when the price of a product that she is interested in buying drops below a certain amount. If she buys it she would also like to be notified when the product has been shipped to her.

10.1.5 Keep Replicas Consistent With Source Object

In order to improve performance and availability of accessing some registry objects, a local registry may make replicas of certain objects that are hosted by another registry. The registry would like to be notified when the source object for a replica is updated so that it can synchronize the replica with the latest state of the source object.

10.2 Registry Events

Activities within a registry result in meaningful events. Typically, registry events are generated when a registry processes client requests. In addition, certain registry events may be caused by administrative actions performed by a registry operator. [ebRIM] defines the AuditableEvent class, instances of which represent registry events. When such an event occurs, an AuditableEvent instance is generated by the registry.

10.3 Subscribing to Events

A User may create a subscription with a registry if she wishes to receive notification for a specific type of event. A User creates a subscription by submitting a Subscription instance to a registry using the SubmitObjectsRequest. If a Subscription is submitted to a registry that does not support event notification then the registry must return an UnsupportedCapabilityException.

The listing below shows a sample Subscription that uses a pre-defined SQL query as its selector, that will result in an email notification to the user whenever a Service is created that is classified as a “Plumbing” service and located in “A Little Town”.

The SQL query within the selector in plain English says the following:

```
Find all Services that are Created AND classified by ClassificationNode
where ClassificationNode's Path ends with string "Plumbing", AND classified
by ClassificationNode where ClassificationNode's Code contains string "A
Little Town".
```

```
<?xml version="1.0" encoding="UTF-8"?>

<tns:Subscription xmlns:tns="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5"
xmlns:query="urn:oasis:names:tc:ebxml-regrep:query:xsd:2.5"
xmlns:rim="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5"
xmlns:rs="urn:oasis:names:tc:ebxml-regrep:rs:xsd:2.5"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5 ../../../ebxmlrr-
spec/misc/schema/v3/rim.xsd urn:oasis:names:tc:ebxml-regrep:query:xsd:2.5
../../../ebxmlrr-spec/misc/schema/v3/query.xsd  " id="e3373a7b-4958-4e55-8820-
d03a191fb76a" notificationInterval="P10D" selector="urn:uuid:8ea2c0f5-b14a-4f46-
88b2-f69e35405d86">

<!-- The selector is a reference to a query object that has the following query defined
SELECT * FROM Service s, AuditableEvent e, AffectectedObject ao,
 Classification c1, Classification c2
 ClassificationNode cn1, ClassificationNode cn2 WHERE
  e.eventType = 'Created' AND ao.id = s.id AND ao.parent=e.id AND
  c1.classifiedObject = s.id AND c1.classificationNode = cn1.id AND
    cn1.path LIKE '%Plumbing' AND
  c2.classifiedObject = s.id AND c2.classificationNode = cn2.id AND
    cn2.path LIKE '%A Little Town%'
-->
```



```
    </Selector>
    <Action xsi:type="tns:NotifyActionType" notificationOption="Objects"
endPoint="mailto:someone@littletown.us"/>
</tns:Subscription>
```

10.3.1 Event Selection

In order for a User to only be notified of specific events of interest, she must specify a reference to a stored AdHocQuery via the selector attribute within the Subscription instance. The query determines whether an event qualifies for that Subscription or not. The query syntax is the standard ad hoc query syntax described in chapter 8.

10.3.2 Notification Action

When creating a Subscription, a User may also specify Actions within the subscription that specify what the registry must do when an event matching the Subscription (subscription event) transpires.

A user may omit specifying an Action within a Subscription if does not wish to be notified by the registry. A user may periodically poll the registry and pull the pending Notifications.

[ebRIM] defines two standard ways that a NotifyAction may be used:

- Email NotifyAction that allows delivery of event notifications via email to a human user or to an email end point for a software component or agent.
- Service NotifyAction that allows delivery of event notifications via a programmatic interface by invocating a specified listener web service.

For each event that transpires in the registry, if the registry supports event notification, it must check all registered and active Subscriptions and see if any Subscriptions match the event. If a match is found then the registry performs the Notification Actions required for the Subscription.

10.3.3 Subscription Authorization

A registry may use registry specific policies to decide which User is authorized to create a subscription and to what events. A Registry must return an AuthorizationException in the event that an Unauthorized User submits a Subscription to a registry.

10.3.4 Subscription Quotas

A registry may use registry specific policies to decide an upper limit on the number of Subscriptions a User is allowed to create. A Registry must return a QuotaExceededException in the event that an Authorized User submits more Subscriptions than allowed by their registry specific quota.

10.3.5 Subscription Expiration

Each subscription defines a startDate and and endDate attribute which determines the period within which a Subscription is active. Outside the bounds of the active period, a Subsription may exist in an expired state within the registry. A registry may remove an expired Subscription at any time. In such cases the identity of a RegistryOperator User must be used for the request in order to have sufficient authorization to remove a User's Subscription.

A Registry must not consider expired Subscriptions when delivering notifications for an event to its Subscriptions. An expired Subscription may be renewed by submitting a new Subscription.

10.3.6 Subscription Rejection

A Registry may reject a Subscription if it is too costly to support. For instance a Subscription that wishes to be notified of any change in any object may be too costly for most registries. A Registry MUST return a SubscriptionTooCostlyException in the event that an Authorized User submits a Subscription that is too costly for the registry to process.

10.4 Unsubscribing from Events

A User may terminate a Subscription with a registry if she no longer wishes to be notified of events related to that Subscription. A User terminates a Subscription by deleting the corresponding Subscription object using the RemoveObjectsRequest to the registry. Removal of a Subscription object follows the same rules as removal of any other object.

10.5 Notification of Events

A registry performs the Actions for a Subscription in order to actually deliver the events. However, regardless of the specific delivery action, the registry must communicate the Subscription events. The Subscription events are delivered within a Notification instance as described by [ebRIM]. In case of Service NotifyAction, the Notification is delivered to a handler service conformant to the RegistryClient interface described in section 6.7.3. In case of an Email NotifyAction the notification is delivered an email address.

The listing below shows a sample Notification matching the subscription example in section 10.3:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:Notification
    xmlns:tns="urn:oasis:names:tc:ebxml-regrep:event:xsd:2.5"
    xmlns:query="urn:oasis:names:tc:ebxml-regrep:query:xsd:2.5"
    xmlns:rim="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5"
    xmlns:rs="urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

status="Success" subscription="e3373a7b-4958-4e55-8820-d03a191fb76a"
xsi:type="tns:ObjectsNotificationType">
       <rim:LeafRegistryObjectList>
                                    <rim:Service id="f3373a7b-4958-4e55-8820-d03a191fb76a">
                                      <rim:Name>
                                             <rim:LocalizedString value="A Little Town
Plumbing"/>
                                      </rim:Name>
                                      <rim:Classification id="a3373a7b-4958-4e55-8820-
d03a191fb76a" classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
                                      <rim:Classification id="b3373a7b-4958-4e55-8820-
d03a191fb76a" classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
                                    </rim:Service>
       </rim:LeafRegistryObjectList>
</tns:Notification>
```

[ebRIM] defines an extensible description of Notifications, making it possible to allow for registry or application specific Notifications. It defines several normative types of Notifications.

A client may specify the type of Notification they wish to receive using the notificationOption attribute of the Action within the Subscription. The registry may override this notificationOption based upon registry specific operational policies.

10.6 Retrieval of Events

The registry provides asynchronous PUSH style delivery of Notifications via notify Actions as described earlier. However, a client may also use a PULL style to retrieve any pending events for their Subscriptions. Pulling of events is done using the Get Notifications protocol.

10.6.1 GetNotificationsRequest

The GetNotificationsRequest is used by a client to retrieve any pending events for their Subscriptions.

Syntax:

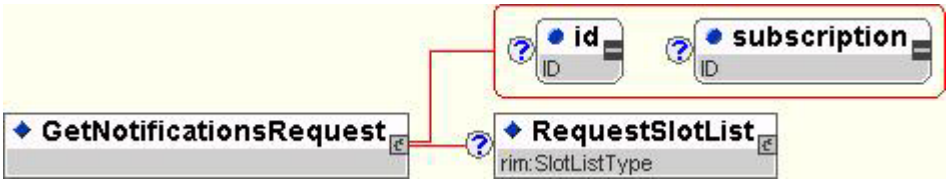


Figure 74: GetNotificationsRequest Syntax

Parameters:

- *subscription:* This parameter specifies the id to a Subscription object which the client wishes to get Notifications.

Returns:

This request returns a NotificationType. See section 0 for details.

Exceptions:

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

- *ObjectNotFoundException:* signifies that the specified Subscription was not found in the registry.

10.6.2 NotificationType

NotificationsType is the simplest form of notification. It is the base type for all types of Notifications.

Syntax:

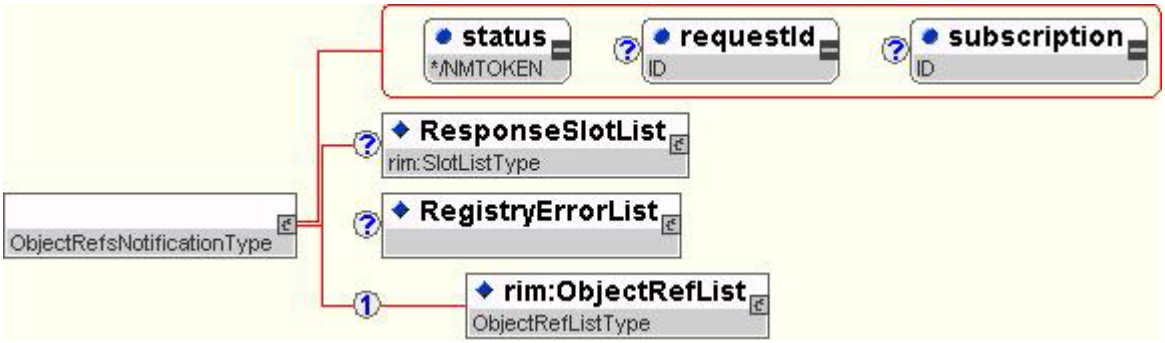


Figure 75: NotificationType Syntax

Parameters:

- *subscription:* This parameter specifies the id to a Subscription object for which this is a Notification.

10.6.3 ObjectRefsNotification

ObjectRefsNotification is a concrete type of Notification that may be sent by the registry as a response to GetNotificationsRequest. It extends NotificationType.

Syntax:

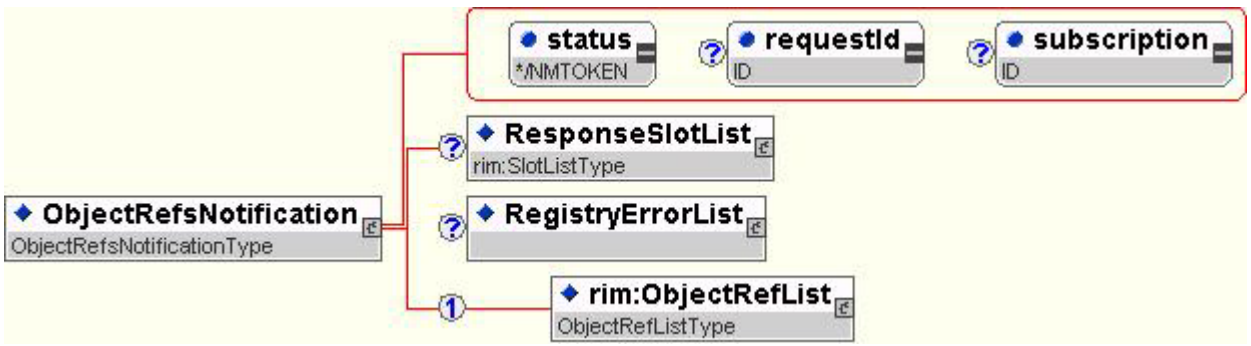


Figure 76: ObjectRefsNotification Syntax

Parameters:

- *ObjectRefList:* This parameter specifies a Collection of ObjectRef instances where each ObjectRef is to a RegistryObject that matches the Subscription for which this is a Notification. The client must retrieve the actual RegistryObjects separately using the ObjectRefs.

10.6.4 ObjectsNotification

ObjectsNotification is a concrete type of Notification that may be sent by the registry as a response to GetNotificationsRequest. It extends NotificationType.

Syntax:

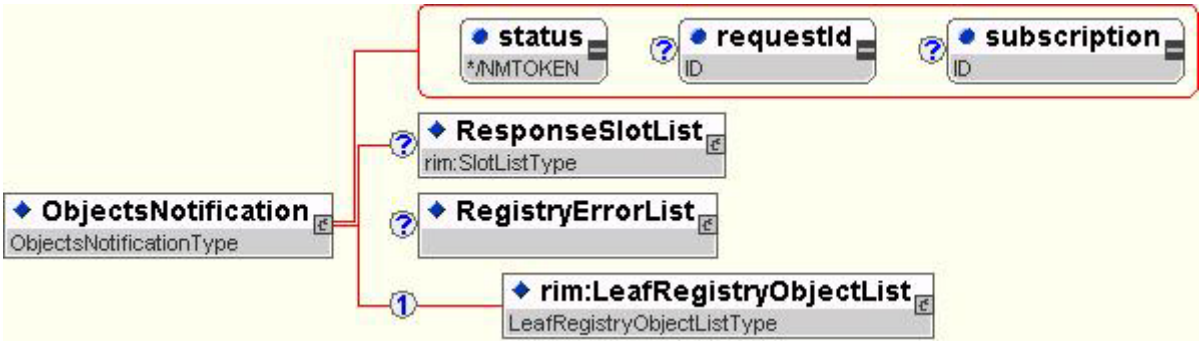


Figure 77: ObjectsNotification Syntax

Parameters:

- *LeafRegistryObjectList:* This parameter specifies a Collection of RegistryObject instances where each RegistryObject is one tha was matches the Subscription for which this is a Notification.

10.7 Purging of Events

A registry may periodically purge AuditableEvents in order to manage its resources. It is up to the registry when such purging occurs. It is up to the registry to determine when undelivered events are purged.

11 Cooperating Registries Support

This chapter describes the capabilities and protocols that enable multiple ebXML registries to cooperate with each other to meet advanced use cases.

11.1 Cooperating Registries Use Cases

The following is a list of use cases that illustrate different ways that ebXML registries cooperate with each other.

11.1.1 Inter-registry Object References

A Submitting Organization wishes to submit a RegistryObject to a registry such that the submitted object references a RegistryObject in another registry.

An example might be where a RegistryObject in one registry is associated with a RegistryObject in another registry.

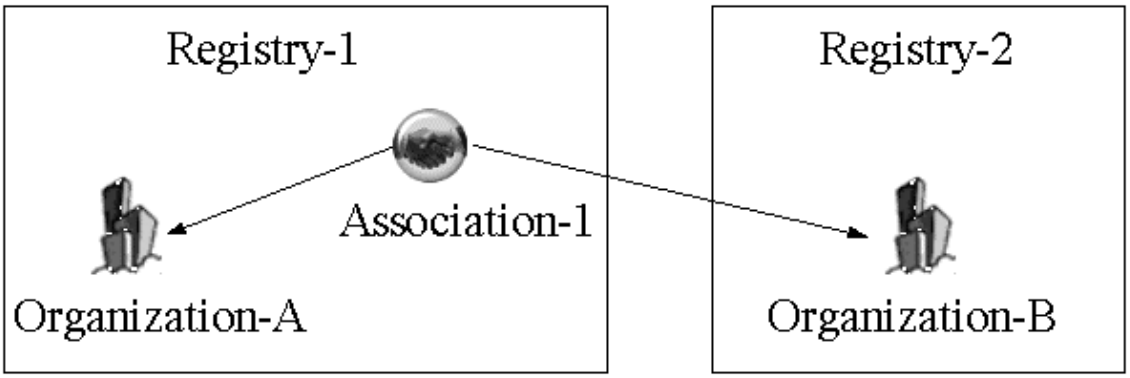


Figure 78: Inter-registry Object References

11.1.2 Federated Queries

A client wishes to issue a single query against multiple registries and get back a single response that contains results based on all the data contained in all the registries. From the client's perspective it is issuing its query against a single logical registry that has the union of all data within all the physical registries.

11.1.3 Local Caching of Data from Another Registry

A destination registry wishes to cache some or all the data of another source registry that is willing to share its data. The shared dataset is copied from the source registry to the destination registry and is visible to queries on the destination registry even when the source registry is not available.

Local caching of data may be desirable in order to improve performance and availability of accessing that object.

An example might be where a RegistryObject in one registry is associated with a RegistryObject in another registry, and the first registry caches the second RegistryObject locally.

11.1.4 Object Relocation

A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from the registry where it was submitted to another registry.

11.2 Registry Federations

A registry federation is a group of registries that have voluntarily agreed to form a loosely coupled union. Such a federation may be based on common business interests and specialties that the registries may share. Registry federations appear as a single logical registry, to registry clients.

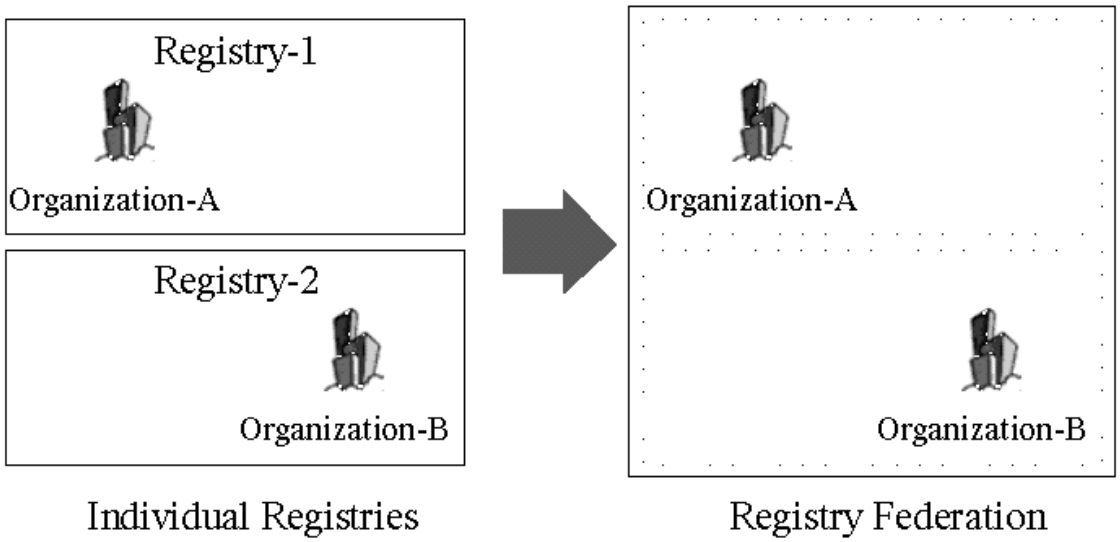


Figure 79: Registry Federations

Registry federations are based on a peer-to-peer (P2P) model where all participating registries are equal. Each participating registry is called a *registry peer*. There is no distinction between the registry operator that created a federation and those registry operators that joined that Federation later.

Any registry operator may form a registry federation at any time. When a federation is created it must have exactly one registry peer which is the registry operated by the registry operator that created the federation.

Any registry may choose to voluntarily join or leave a federation at any time.

11.2.1 Federation Metadata

The Registry Information model defines the Registry and Federation classes, instances of these classes and the associations between these instances describe a federation and its members. Such instance data is referred to as Federation Metadata. The Registry and Federation classes are described in detail in [ebRIM].

The Federation information model is summarized here as follows:

- A Federation instance represents a registry federation.
- A Registry instance represents a registry that is a member of the Federation.

- An Association instance with associationType of *HasFederationMember* represents membership of the registry in the federation. This Association links the Registry instance and the Federation instance.

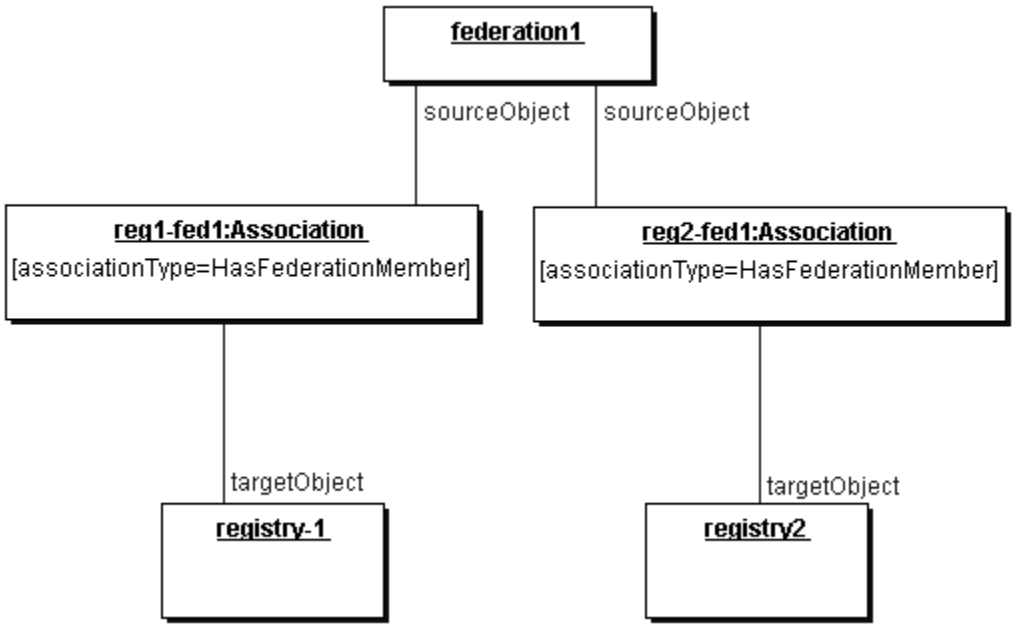


Figure 80: Federation Metadata Example

11.2.2 Local Vs. Federated Queries

A federation appears to registry clients as a single unified logical registry. An AdhocQueryRequest sent by a client to a federation member may be local or federated. A new boolean attribute named *federated* is added to AdhocQueryRequest to indicate whether the query is federated or not.

Local Queries

When the federated attribute of AdhocQueryRequest has the value of *false* then the query is a local query. In the absence of a *federated* attribute the default value of *federated* attribute is *false*. A local AdhocQueryRequest is only processed by the registry that receives the request. A local AdhocQueryRequest does not operate on data that belongs to other registries.

Federated Queries

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

A federated query to any federation member must be routed by that member to all other federation member registries as parallel-distributed queries. A federated query operates on data that belongs to all members of the federation.

When a client submits a federated query to a registry that is not a member of a federation, the registry must treat it as a local query.

Membership in Multiple Federations

A registry may be a member of multiple federations. In such cases if the *federated* attribute of AdhocQueryRequest has the value of *true* then the registry must route the federated query to *all* federations that it is a member of.

Alternatively, the client may specify the id of a specific federation that the registry is a member of, as the value of the *federation* parameter. The type of the federation parameter is anyURI and identifies the "id" attribute of the desired Federation.

In such cases the registry must route the federated query to the specified federation only.

11.2.3 Federated Lifecycle Management Operations

Details on how to create and delete federations and how to join and leave a federation are described in 11.2.8.

All lifecycle operations must be performed on a RegistryObject within its home registry using the operations defined by the LifeCycleManager interface. Unlike query requests, lifecycle management requests do not support any federated capabilities.

11.2.4 Federations and Local Caching of Remote Data

A federation member is not required to maintain a local cache of replicas of RegistryObjects and repository items that belong to other members of the federation.

A registry may choose to locally cache some or all data from any other registry whether that registry is a federation member or not. Data caching is orthogonal to registry federation and is described in section 11.3.

Since by default there is minimal replication in the members of a federation, the federation architecture scales well with respect to memory and disk utilization at each registry.

Data replication is often necessary for performance, scalability and fault-tolerance reasons.

11.2.5 Caching of Federation Metadata

A special case for local caching is the caching of the Federation and Registry instances and related Associations that define a federation and its members. Such data is referred to as federation metadata. A federation member is required to locally cache the federation metadata, from the federation home for each federation that it is a member of. The reason for this requirement is consistent with a Peer-to-Peer (P2P) model and ensures fault –tolerance in case the Federation home registry is unavailable.

The federation member must keep the cached federation metadata synchronized with the master copy in the Federation home, within the time period specified by the replicationSyncLatency attribute of the Federation. Synchronization of cached Federation metadata may be done via synchronous polling or asynchronous event notification using the event notification feature of the registry.

11.2.6 Time Synchronization Between Registry Peers

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

11.2.7 Federations and Security

Federation lifecycle management operations abide by the same security rules as standard lifecycle management.

11.2.8 Federation Lifecycle Management Protocols

This section describes the various operations that manage the lifecycle of a federation and its membership. A key design objective is to allow federation lifecycle operations to be done using standard LifeCycleManager interface of the registry in a stylized manner.

Joining a Federation

The following rules govern how a registry joins a federation:

- Each registry must have exactly one Registry instance within that registry for which it is a home. The Registry instance is owned by the RegistryOperator and may be placed in the registry using any operator specific means. The Registry instance must never change its home registry.
- A registry may request to join an existing federation by submitting an instance of an Extramural Association that associates the Federation instance as sourceObject, to its Registry instance as targetObject, using an associationType of *HasFederationMember*. The home registry for the Association and the Federation objects must be the same.
- The owner of the Federation instance must confirm the Extramural Association in order for the registry to be accepted as a member of the federation.

Creating a Federation

The following rules govern how a federation is created:

- A Federation is created by submitting a Federation instance to a registry using SubmitObjectsRequest.
- The registry where the Federation is submitted is referred to as the federation home.
- The federation home may or may not be a member of that Federation.
- A federation home may contain multiple Federation instances.

Leaving a Federation

The following rules govern how a registry leaves a federation:

A registry may leave a federation at any time by removing its *HasFederationMember* Association instance that links it with the Federation instance. This is done using the standard RemoveObjectsRequest.

Dissolving a Federation

The following rules govern how a federation is dissolved:

- A federation is dissolved by sending a RemoveObjectsRequest to its home registry and removing its Federation instance.
- The removal of a Federation instance is controlled by the same Access Control Policies that govern any RegistryObject.

- The removal of a Federation instance is controlled by the same lifecycle management rules that govern any RegistryObject. Typically, this means that a federation may not be dissolved while it has federation members. It may however be deprecated at any time. Once a Federation is deprecated no new members can join it.

11.3 Object Replication

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

The registry that replicates a remote object locally is referred to as the destination registry for the replication. The registry that contains the remote object being replicated is referred to as the source registry for the replication.

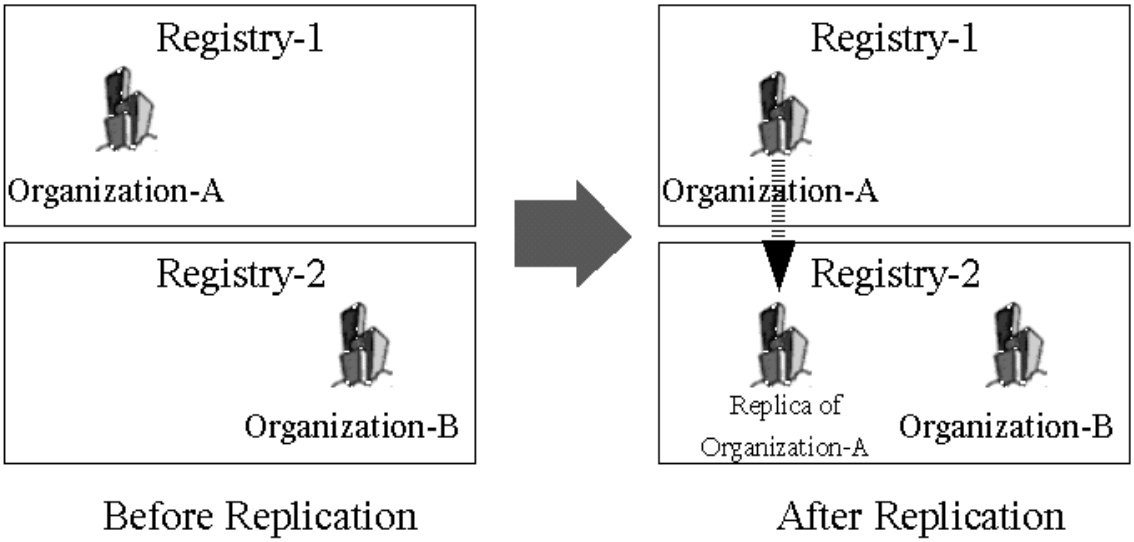


Figure 81: Object Replication

11.3.1 Use Cases for Object Replication

A registry may create a local replica of a remote object for a variety of reasons. A few sample use cases follow:

- Improve access time and fault tolerance via locally caching remote objects. For example, a registry may automatically create a local replica when a remote ObjectRef is submitted to the registry.
- Improve scalability by distributing access to hotly contested object, such as NAICS scheme, across multiple replicas.
- Enable cooperating registry features such as hierarchical registry topology and local caching of federation metadata.

11.3.2 Queries And Replicas

A registry must support client queries to consider a local replica of remote object as if it were a local object. Local replicas are considered within the extent of the data set of a registry as far as local queries are concerned.

When a client submits a local query that retrieves a remote object by its id attribute, if the registry contains a local replica of that object then the registry should return the state defined by the local replica.

11.3.3 Lifecycle Operations And Replicas

LifeCycle operations on an original object must be performed at the home registry for that object. LifeCycle operations on replicas of an original object only affect the replica and do not have any impact on the original object.

11.3.4 Object Replication and Federated Registries

Object replication capability is orthogonal to the registry federation capability. Objects may be replicated from any registry to any other registry without any requirement that the registries belong to the same federation.

11.3.5 Creating a Local Replica

Any Submitting Organization can create a replica by using the standard SubmitObjectsRequest. If a registry receives a SubmitObjectRequest which has an RegistryObjectList containing a remote ObjectRef, then it must create a replica for that remote ObjectRef.

In addition to Submitting Organizations, a registry itself may create a replica under specific situations in a registry specific manner.

Creating a local replica requires the destination registry to read the state of the remote object from the source registry and then create a local replica of the remote object.

A registry may use standard QueryManager interface to read the state of a remote object (whether it is an original or a replica). No new APIs are needed to read the state of a remote object. Since query functionality does not need prior registration, no prior registration or contract is needed for a registry to read the state of a remote object.

Once the state of the remote object has been read, a registry may use registry specific means to create a local replica of the remote object. Such registry specific means may include the use of the LifeCycleManager interface.

A replica of a RegistryObject may be distinguished from an original since a replica must have its home attribute point to the remote registry where the original for the replica resides.

11.3.6 Transactional Replication

Transactional replication enables a registry to replicate events in another registry in a transactionally consistent manner. This is typically the case when entire registries are replicated to another registry.

This specification defines a more loosely coupled replication model as an alternative to transactional replication for the following reasons:

- Transactional replication requires a tight coupling between registries participating in the replication
- Transactional replication is not a typical use case for registries
- Loosely coupled replication as defined by this specification typically suffices for most use cases
- Transaction replication is very complex and error prone

Registry implementations are not required to implement transactional replication.

11.3.7 Keeping Replicas Current

A registry must keep its replicas current within the latency specified by the value of the *replicationSyncLatency* attribute defined by the registry. This includes removal of the replica when its original is removed from its home registry.

Replicas may be kept current using the event notification feature of the registry or via periodic polling.

11.3.8 Write Operations on Local Replica

Local Replicas are read-only objects. Lifecycle management operations of RegistryObjects are not permitted on local replicas. All lifecycle management operation to RegistryObjects must be performed in the home registry for the object.

11.3.9 Tracking Location of a Replica

A local replica of a remote RegistryObject instance must have exactly one ObjectRef instance within the local registry. The home attribute of the ObjectRef associated with the replica tracks its home location. A RegistryObject must have exactly one home. The home for a RegistryObject may change via Object Relocation as described in section 11.4. It is optional for a registry to track location changes for replicas within it.

11.3.10 Remote Object References to a Replica

It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another RegistryObject. In such cases the home attribute of the ObjectRef contains the base URI to the home registry for the replica.

11.3.11 Removing a Local Replica

A Submitting Organization can remove a replica by using the RemoveObjectsRequest. If a registry receives a RemoveObjectsRequest that has an ObjectRefList containing a remote ObjectRef, then it must remove the local replica for that remote ObjectRef.

11.4 Object Relocation Protocol

Every RegistryObject has a home registry and a User within the home registry that is the publisher or owner of that object. Initially, the home registry is the where the object is originally submitted. Initially, the owner is the User that submitted the object.

A RegistryObject may be relocated from one home registry to another home registry using the Object Relocation protocol.

Within the Object Relocation protocol, the new home registry is referred to as the *destination* registry while the previous home registry is called the *source* registry.

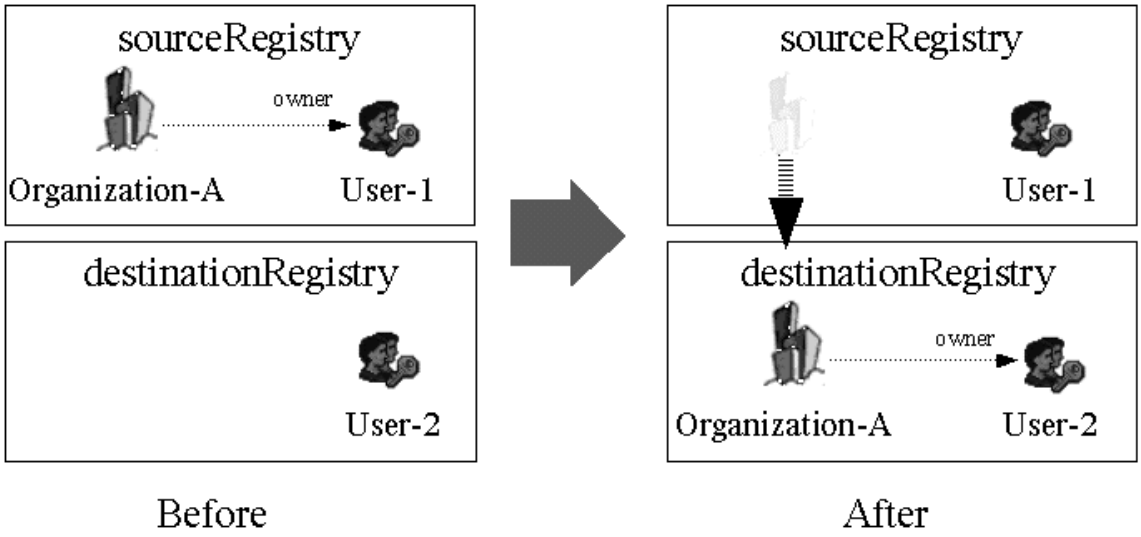


Figure 82: Object Relocation

The User at the source registry who owns the objects being relocated is referred to as the *ownerAtSource*. The User at the destination registry, who is the new owner of the objects, is referred to as the *ownerAtDestination*. While the ownerAtSource and the ownerAtDestination may often be the same identity, the Object Relocation protocol treats them as two distinct identities.

A special case usage of the Object Relocation protocol is to transfer ownership of RegistryObjects from one User to another within the same registry. In such cases the protocol is the same except for the fact that the source and destination registries are the same.

Following are some notable points regarding object relocation:

- Object relocation does not require that the source and destination registries be in the same federation or that either registry have a prior contract with the other.
- Object relocation must preserve object id. While the home registry for a RegistryObject may change due to object relocation, its id never changes.
- ObjectRelocation must preserve referential integrity of RegistryObjects. Relocated objects that have references to an object that did not get relocated must preserve their reference. Similarly objects that have references to a relocated object must also preserve their reference. Thus, relocating an object may result in making the value of a reference attribute go from being a local reference to being a remote reference or vice versa.
- AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque transactonId identifying the relocateObjects transaction.
- The requests defined by the Relocate Objects protocol must be sent to the source or destination registry only.

- When an object is relocated an AuditableEvent of type "Relocated" must be recorded by the sourceRegistry. Relocated events must have the source and destination registry's base URIs recorded as two Slots on the Relocated event. The names of these Slots are sourceRegistry and destinationRegistry respectively.

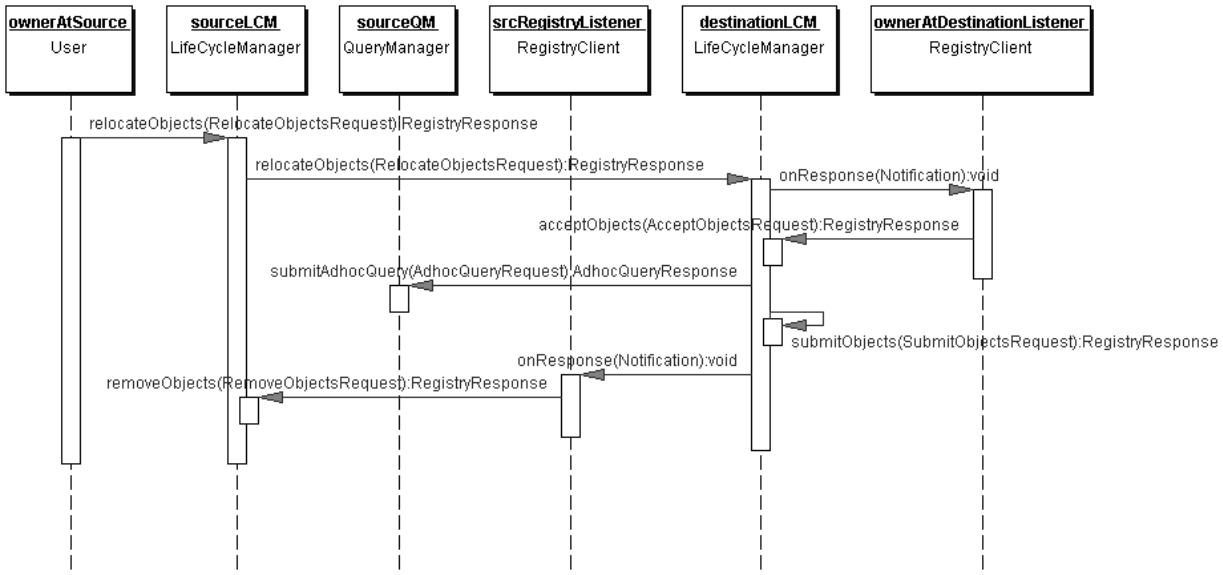


Figure 83: Relocate Objects Protocol

Figure 83 illustrates the Relocate Objects Protocol. The participants in the protocol are the ownerAtSource and ownerAtDestination User instances as well as the LifeCycleManager interfaces of the sourceRegistry and destinationRegistry.

The steps in the protocol are described next:

1. The protocol is initiated by the ownerAtSource sending a RelocateObjectsRequest message to the LifeCycleManager interface of the sourceRegistry. The sourceRegistry must make sure that the ownerAtSource is authorized to perform this request. The id of this RelocateObjectsRequest is used as the transaction identifier for this instance of the protocol. This RelocateObjectsRequest message must contain an ad hoc query that specifies the objects that are to be relocated.

2. Next, the sourceRegistry must relay the same RelocateObjectsRequest message to the LifeCycleManager interface of the destinationRegistry. This message enlists the detsinationRegistry to participate in relocation protocol. The destinationRegistry must store the request information until the protocol is completed or until a registry specific period after which the protocol times out.

3. The destinationRegistry must relay the RelocateObjectsRequest message to the ownerAtDestination. This notification may be done using the event notification feature of the registry as described in chapter 10. The notification may be done by invoking a listener Service for the ownerAtDestination or by sending an email to the ownerAtDestination. This concludes the first phase of the Object Relocation protocol.

4. The ownerAtDestination at a later time may send an AcceptObjectsRequest message to the destinationRegistry. This request must identify the object relocation transaction via the *correlationId*. The value of this attribute must be the id of the original RelocateObjectsRequest.

5. The destinationRegistry sends an AdhocQueryRequest message to the sourceRegistry. The source registry returns the objects being relocated as an AdhocQueryResponse. In the event of a large number of objects this may involve multiple AdhocQueryRequest/responses as described by the iterative query feature described in section 8.1.4.
6. The destinationRegistry submits the relocated data to itself assigning the identity of the ownerAtDestination as the owner. The relocated data may be submitted to the destination registry using any registry specific means or a SubmitObjectsRequest. However, the effect must be the same as if a SubmitObjectsRequest was used.
7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been safely committed using the Event Notification feature of the registry as described in chapter 10.
8. The sourceRegistry removes the relocated objects using any registry specific means and logging an AuditableEvent of type Relocated. This concludes the Object Relocation transaction.

11.4.1 RelocateObjectsRequest

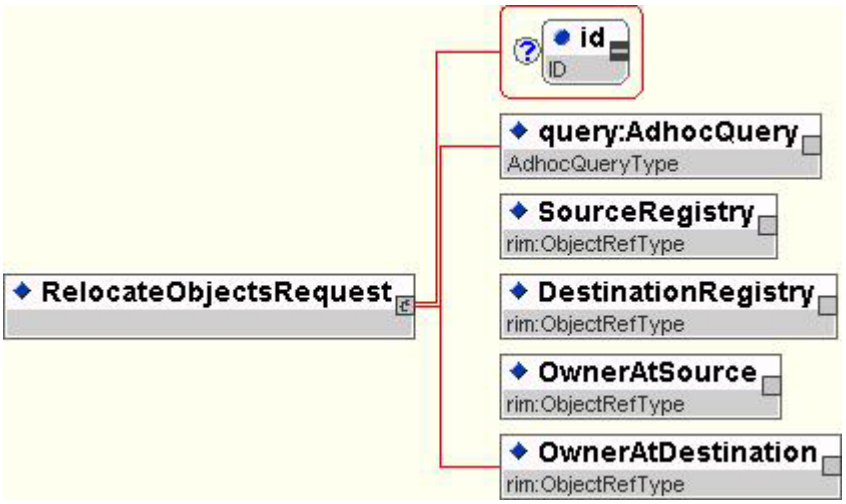


Figure 84: RelocateObjectsRequest XML Schema

Parameters:

- *id:* the attribute id provides the transaction identifier for this instance of the protocol.
- *AdhocQuery:* This element specifies an ad hoc query that selects the RegistryObjects that are being relocated.
- *SourceRegistry:* This element specifies the ObjectRef to the sourceRegistry Registry instance. The value of this attribute must be a local reference when the message is sent by the ownerAtSource to the sourceRegistry.
- *destinationRegistry:* This element specifies the ObjectRef to the destinationRegistry Registry instance.
- *ownerAtSource:* This element specifies the ObjectRef to the ownerAtSource User instance.

- *ownerAtDestination:* This element specifies the ObjectRef to the ownerAtDestination User instance.

Returns:

This request returns a RegistryResponse. See section 6.9.3 for details.

Exceptions:

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

- *ObjectNotFoundException:* signifies that the specified Registry or User was not found in the registry.

11.4.2 AcceptObjectsRequest

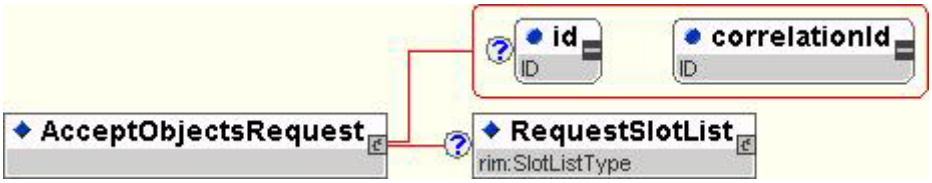


Figure 85: AcceptObjectsRequest Syntax

Parameters:

- *correlationId:* Provides the transaction identifier for this instance of the protocol.

Returns:

This request returns a RegistryResponse. See section 6.9.3 for details.

Exceptions:

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

- *InvalidRequestException:* signifies that the specified correlationId was not found to match an ongoing RelocateObjectsRequest in the registry.

11.4.3 Object Relocation and Remote ObjectRefs

The following scenario describes what typically happens when a person moves:

1. When a person moves from one house to another, other persons may have their old postal addresses.
2. When a person moves, they leave their new address as the forwarding address with the post office.
3. The post office forwards their mail for some time to their new address.
4. Eventually the forwarding request expires and the post office no longer forwards mail for that person.

5. During this forwarding interval the person notifies interested parties of their change of address.

The registry must support a similar model for relocation of RegistryObjects. The following steps describe the expected behavior when an object is relocated.

1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other RegistryObjects may have remote ObjectRefs to O1.
2. The registry R1 must leave an AuditableEvent of type Relocated that includes the home URI for the new registry R2.
3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it must forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the client. The object O1 must include the home URI to R2 within the optional home attribute of RegistryObject. Clients are advised to check the home attribute and update the home attribute of their local ObjectRef to match the new home URI value for the object.
4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no longer required to relay requests for O1 to R2 transparent to the client. Instead R1 must return an ObjectNotFoundException.
5. Clients that are interested in the relocation of O1 and being notified of its new address may choose to be notified by having a prior subscription using the event notification facility of the registry. For example a Registry that has a remote ObjectRefs to O1 may create a subscription on relocation events for O1. This however, is not required behavior.

11.4.4 Notification of Object Relocation To ownerAtDestination

This section describes how the destinationRegistry uses the event notification feature of the registry to notify the ownerAtDestination of a Relocated event.

The destinationRegistry must send a Notification with the following required characteristics:

- The notification must be an instance of a Notification element.
- The Notification instance must have at least one Slot.
- The Slot must have the name eventNotification.correlationId
- The Slot must have the correlationId for the Object Relocation transaction as the value of the Slot.

11.4.5 Notification of Object Commit To sourceRegistry

This section describes how the destinationRegistry uses the event notification feature of the registry to notify the sourceRegistry that it has completed committing the relocated objects.

The destinationRegistry must send a Notification with the following required characteristics:

- The notification must be an instance of a Notification element.
- The Notification instance must have at least one Slot.
- The Slot must have the name eventNotification.objectsCommitted
- The Slot must have the value of *true*.

11.4.6 Object Relocation and Timeouts

No timeouts are specified for the Object Relocation protocol. Registry implementations may cleanup incomplete Object Relocation transactions in a registry specific manner as an administrative task using registry specific policies.

12 Registry Security

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [ebRIM]. Security glossary terms can be referenced from RFC 2828.

12.1 Security Concerns

In the current version of this specification, we address data integrity and source integrity (item 1 in Appendix E.1). We have used a minimalist approach to address the access control concern as in item 2 of Appendix E.1. By default, any Registered User identified by a User instance as defined by [ebRIM] can publish content and anyone can view published content. In addition to this default behaviour, the Registry Information Model [ebRIM] is designed to support more sophisticated security policies in future versions of this specification.

12.2 Integrity of Registry Content

It is assumed that most registries do not have the resources to validate the veracity of the content submitted to them. The mechanisms described in this section can be used to ensure that any tampering with the content can be detected. Furthermore, these mechanisms support unambiguous identification of a Registered User as the submitter for any registry content. The Registered User has to sign the contents before submission – otherwise the content will be rejected.

12.2.1 Message Payload Signature

The integrity of the Registry content requires that all submitted content be signed by the Registered User that submits the content. The signature on the submitted content ensures that:

- Any tampering of the content can be detected.
- The content's veracity can be ascertained by its association with a specific Registered User and its indirect association with the Organization that the Registered User may be affiliated with.

This section specifies the requirements for generation, packaging and validation of payload signatures. A payload signature is packaged with the payload. Therefore the requirements apply regardless of whether the Registry Client and the Registration Authority communicate over standard SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML Messaging Service does not specify the generation, validation and packaging of payload signatures. The specification of payload signatures is left up to the application and therefore defined by this specification for ebXML Registry client applications. The requirements on the payload signatures augment the [ebMS] specification.

Use Case

This Use Case illustrates the use of header and payload signatures (we discuss header signatures later).

- RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the content along with the payload signature to the Registry.

- RC2 (Registry Client 2) retrieves RC1's content from the Registry.
- RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves the content, the response from the Registration Authority to RC2 contains the following:
  - Payload containing the content that has been published by RC1.
  - RC1's payload signature (represented by a ds:Signature element) over RC1's published content.
  - The public key for validating RC1's payload signature in ds:Signature element (using the KeyInfo element as specified in [XMLDSIG]) so RC2 can obtain the public key for signature (e.g. retrieve a certificate containing the public key for RC1).
  - A ds:Signature element containing the header signature. Note that the Registration Authority (not RC1) generates this signature.

12.2.2 Payload Signature Requirements

Payload Signature Packaging Requirements

A payload signature is represented by a ds:Signature element. The payload signature must be packaged with the payload as specified here. This packaging assumes that the payload is always signed.

- The payload and its signature must be enclosed in a MIME multipart message with a Content-Type of multipart/related.
- The first body part must contain the XML signature as specified in Section 0, "Payload Signature Generation Requirements".
- The second body part must be the content.

The packaging of the payload signature with two payloads is as shown in the example in Section 8.4.4.

Payload Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in this section. Note: the "ds" name space reference is to http://www.w3.org/2000/09/xmldsig#

- ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a client may use any of these attributes. However, signing using the following Algorithm attribute: http://www.w3.org/2000/09/xmldsig#dsa-sha1 will allow interoperability with all XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this algorithm.

The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The following Canonicalization algorithm (specified in [XMLDSIG]) must be supported
http://www.w3.org/TR/2001/REC-xml-c14n-20010315

- One ds:Reference element to reference each of the payloads that needs to be signed must be created. The ds:Reference element:
  - Must identify the payload to be signed using the URI attribute of the ds:Reference element.
  - Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support the following digest algorithm:

http://www.w3.org/2000/09/xmldsig#sha1

- Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

The ds:SignatureValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject to the requirements stated in Section 12.4, "KeyDistrbution and KeyInfo element".

Message Payload Signature Validation

The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

Payload Signature Example

The following example shows the format of the payload signature:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
     <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1"/>
     <ds:CanonicalizationMethod>
          Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
      </ds:CanonicalizationMethod>
       <ds:Reference URI=#Payload1>
            <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmldsig#sha1">
             <ds:DigestValue> ... </ds:DigestValue>
       </ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>
```

12.3 Authentication

The Registry must be able to authenticate the identity of the User associated with client requests and also the owner of each repository item.

The User can be identified by verifying the message header signature with the certificate of the User. The certificate may be in the message itself or provided to the registry through means unspecified in this specification. If not provided in the message, this specification does not specify how the Registry correlates a specific message with a certificate. Each payload must also be signed to ensure the integrity of that payload and to determine its owner. Authentication is also required in order to identify the actions a User is authorized to perform with respect to specific RegistryObject rosources in the Registry.

The Registry must perform authentication on a per message basis. From a security point of view, all messages are independent and there is no concept of a session encompassing multiple messages or conversations. Session support may be added as an optimization feature in future versions of this specification.

It is important to note that the message header signature can only guarantee data integrity. It does not guarantee safety from "replay" attacks. True support for authentication requires timestamps or nonce (nonrecurring series of numbers to identify each message) that are signed.

12.3.1 Message Header Signature

Message headers are signed to provide data integrity while the message is in transit. Note that the signature within the message header also signs the digests of the payloads.

Header Signature Requirements

Message headers may be signed and are referred to as a header signature. When a Registered User sends a request, the Registration Authority may use a pre-established contract or a default policy to determine whether the response should contain a header signature. When a Registry Guest sends a request, the Registration Authority may use a default policy to determine whether the response contains a header signature.

This section specifies the requirements for generation, packaging and validation of a header signature. These requirements apply when the Registry Client and Registration Authority communicate using standard SOAP with Attachments. When ebXML MS is used for communication, then the message handler (i.e. [ebMS]) specifies the generation, packaging and validation of XML signatures in the SOAP header. Therefore the header signature requirements do not apply when the ebXML MS is used for communication. However, payload signature generation requirements (0) do apply whether standard SOAP with Attachments or ebXML MS is used for communication.

Packaging Requirements

A header signature is represented by a ds:Signature element. The ds:Signature element generated must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature element in the SOAP header field is shown in Section 8.4.4.

Header Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in this section. A ds:Signature element contains:

- ds:SignedInfo
- ds:SignatureValue
- ds:KeyInfo

The ds:SignedInfo element must be generated as follows:

1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute, a client must be capable of signing using only the following Algorithm attribute: http://www.w3.org/2000/09/xmldsig#dsa-sha1. All XMLDSIG implementations conforming to the [XMLDSIG] specification support this algorithm.
2. The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:

   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation. This signs the entire ds:Reference element and:
   - Must include the ds:Transform
     http://www.w3.org/2000/09/xmldsig#enveloped-signature

This ensures that the signature (which is embedded in the <SOAP-ENV:Header> element) is not included in the signature calculation.

- Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the ds:Reference element. The URI attribute is optional in the [XMLDSIG] specification. The URI attribute must be an empty string ("").
- Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support the digest algorithm: http://www.w3.org/2000/09/xmldsig#sha1
- Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

The ds:SignatureValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in Section 12.4, "KeyDistrbution and KeyInfo element".

Header Signature Validation Requirements

The ds:Signature element for the ebXML message header must be validated by the recipient as specified by [XMLDSIG].

Header Signature Example

The following example shows the format of a header signature:

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>
        <ds:CanonicalizationMethod>
            Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
        </ds:CanonicalizationMethod>
        <ds:Reference URI= "">
            <ds:Transform>
                http://www.w3.org/2000/09/xmldsig#enveloped-signature
            </ds:Transform>
            <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
            <ds:DigestValue> ... </ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>
```

12.4 Key Distribution and KeyInfo Element

To validate a signature, the recipient of the signature needs the public key corresponding to the signer's private key. The participants may use the KeyInfo field of ds:Signature, or distribute the public keys in an implementation specific manner. In this section we consider the case when the public key is sent in the KeyInfo field. The following use cases need to be addressed:

- Registration Authority needs the public key of the Registry Client to validate the signature

- Registry Client needs the public key of the Registration Authority to validate the Registry's signature.
- Registry Client RC1 needs the public key of Registry Client RC2 to validate the content signed by RC1.
- [XMLDSIG] provides an optional *ds:KeyInfo* element that can be used to pass information to the recipient for retrieving the public key. This field together with the procedures outlined in this section is used to securely pass the public key to a recipient. If the KeyInfo field is present, it must contain a X509 Certificate as specified in [XMLDSIG].

The following assumptions are also made:

1. A Certificate is associated both with the Registration Authority and a Registry Client.
2. A Registry Client registers its certificate with the Registration Authority. The mechanism used for this is not specified here.
3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local key store. The mechanism is not specified here.

Appendix F.8 contains a few scenarios on the use of KeyInfo field.

12.5 Confidentiality

12.5.1 On-the-wire Message Confidentiality

It is suggested but not required that message payloads exchanged between Registry Clients and the Registry be encrypted during transmission. This specification does not specify how payload encryption is to be done.

12.5.2 Confidentiality of Registry Content

In the current version of this specification, there are no provisions for confidentiality of Registry content. All content submitted to the Registry may be discovered and read by any client. This implies that the Registry and the client need to have an a priori agreement regarding encryption algorithm, key exchange agreements, etc.

12.6 Access Control and Authorization

The Registry must provide an access control and authorization mechanism based on the Access Control Information Model defined in [ebRIM]. This model defines a default access control policy that must be supported by the registry. In addition it also defines a binding to [XACML] that allows fine-grained access control policies to be defined.

12.6.1 Actors / Role Mapping

The following table shows the mapping of actors listed in Section 5.3 and their default roles.

Actor	Role
Registered User	ContentOwner
Registry Administrator Registration Authority	RegistryAdministrator
Registry Guest	RegistryGuest
Registry Reader	RegistryGuest

5964 **Table 9: Default Actor to Role Mappings**

5965
5966

Appendix A Web Service Architecture

A.1 Registry Service Abstract Specification

The normative definition of the Abstract Registry Service in WSDL is defined at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/services/Registry.wsdl

A.2 Registry Service SOAP Binding

The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/services/RegistrySOAPBinding.wsdl

Appendix B ebXML Registry Schema Definitions

B.1 RIM Schema

The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rim.xsd

B.2 Registry Services Interface Base Schema

The normative XML Schema definition that defines the XML requests and responses supported by the registry service interfaces in this document can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rs.xsd

B.3 QueryManager Service Schema

The normative XML Schema definition for the XML syntax for the QueryManager service interface can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/query.xsd

B.4 LifecycleManager Service Schema

The normative XML Schema definition for the XML syntax for the LifecycleManager service interface can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/lcm.xsd

B.5 Content Management Service Schema

The normative XML Schema definition for the XML syntax for the Content Management Services interface can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/cms.xsd

B.6 Examples of Instance Documents

A growing number of non-normative XML instance documents that conform to the normative Schema definitions described earlier may be found at the following location on the web:
http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/

Appendix C Interpretation of UML Diagrams

This section describes in *abstract terms* the conventions used to define ebXML business process description in UML.

C.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces required to implement an ebXML Registry Services and clients. The UML class diagram contains:

1. A collection of UML interfaces where each interface represents a Registry Service interface.
2. Tabular description of methods on each interface where each method represents an Action (as defined by [ebCPP]) within the Service Interface.
3. One or more parameters for each method. The type of each parameter represents the ebXML message type that is exchanged as part of the Action corresponding to the method. Multiple arguments imply multiple payload documents within the body of the corresponding ebXML message.

C.2 UML Sequence Diagram

A UML sequence diagram is used to specify the business protocol representing the interactions between the UML interfaces for a Registry specific ebXML business process. A UML sequence diagram provides the necessary information to determine the sequencing of messages and request to response association as well as request to error response association.

Each sequence diagram shows the sequence for a specific conversation protocol as method calls from the requestor to the responder. Method invocation may be synchronous or asynchronous based on the UML notation used on the arrowhead for the link. A half arrowhead represents asynchronous communication. A full arrowhead represents synchronous communication.

Each method invocation may be followed by a response method invocation from the responder to the requestor to indicate the ResponseName for the previous Request. Possible error response is indicated by a conditional response method invocation from the responder to the requestor. See Figure 15 on page 39 for an example.

Appendix D SQL Query

D.1 SQL Query Syntax Specification

This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax conforms to the <query specification> as well as the additional restrictions identified below:

1. A **<select list>** may contain at most one **<select sublist>.**
2. The **<select list>** must be is a single column whose data type is UUID, from the table in the **<from clause>.**
3. A **<derived column>** may not have an **<as clause>**.
4. **A <table expression>** does not contain the optional **<group by clause>** and **<having clause>** clauses.
5. A **<table reference>** can only consist of **<table name>** and **<correlation name>.**
6. A **<table reference>** does not have the optional AS between **<table name>** and **<correlation name>.**
7. Restricted use of sub-queries is allowed by the syntax as follows. The **<in predicate>** allows for the right hand side of the **<in predicate>** to be limited to a restricted **<query specification>** as defined above.
8. The SQL query syntax allows for the use of **<sql invoked routines>** invocation from [SQL/PSM] as the RHS of the **<in predicate>**.

D.2 Non-Normative BNF for Query Syntax Grammar

The following BNF exemplifies the grammar for the registry query syntax. It is provided here as an aid to implementers. Since this BNF is not based directly on [SQL] it is provided as non-normative syntax. For the normative syntax rules see Appendix D.1.

```
sqlQuery ::= SQLSelect ( <SEMICOLON> )? <EOF>
SQLSelect ::= <SELECT> SQLSelectCols <FROM> SQLTableList ( SQLWhere )? ( SQLOrderBy )?
SQLSelectCols ::= ( <ALL> | <DISTINCT> )* ( ( "*" | SQLLvalueTerm ) )
SQLTableList ::= SQLTableRef ( "," SQLTableRef )*
SQLTableRef ::= ( <ID> ( <ID> )? )
SQLWhere ::= <WHERE> SQLOrExpr
SQLOrExpr ::= SQLAndExpr ( <OR> SQLAndExpr )*
SQLAndExpr ::= SQLNotExpr ( <AND> SQLNotExpr )*
SQLNotExpr ::= ( <NOT> )? SQLCompareExpr
SQLCompareExpr ::= ( SQLIsClause | SQLSumExpr ( SQLCompareExprRight )? )
```

```
SQLCompareExprRight ::= ( SQLLikeClause | SQLInClause | SQLCompareOp SQLSumExpr )
SQLCompareOp ::= ( <EQUAL> | <NOTEQUAL> | <NOTEQUAL2> |
                   <GREATER> | <GREATEREQUAL> | <LESS> |
                   <LESSEQUAL> )
SQLInClause ::= ( <NOT> )? <IN> "(" SQLLValueListOrProcedureCall ")"
SQLLValueListOrProcedureCall ::= ( ProcedureCall | SQLLValueList )
ProcedureCall ::= <ID> "(" <STRING_LITERAL> ")"
SQLLValueList ::= SQLLValueElement ( "," SQLLValueElement )*
SQLLValueElement ::= ( <NULL> | SQLSumExpr | SQLSelect )
SQLIsClause ::= SQLColRef <IS> ( <NOT> )? <NULL>
SQLLikeClause ::= ( <NOT> )? <LIKE> SQLPattern
SQLPattern ::= ( <STRING_LITERAL> )
SQLColRef ::= SQLLvalue
SQLLvalue ::= ( SQLLvalueTerm )
SQLLvalueTerm ::= <ID> ( <DOT> <ID> )*
SQLSumExpr ::= SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
SQLProductExpr ::= SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr )*
SQLUnaryExpr ::= ( ( "+" | "-" ) )? SQLTerm
SQLTerm ::= ( "(" SQLOrExpr ")" | SQLColRef | SQLLiteral )
SQLLiteral ::= ( <STRING_LITERAL> | <INTEGER_LITERAL> |
               <FLOATING_POINT_LITERAL> )
SQLOrderBy ::= <ORDER> <BY> SQLOrderByList
SQLOrderByElem ::= SQLColRef ( SQLOrderDirection )?
SQLOrderByList ::= SQLOrderByElem ( "," SQLOrderByElem )*
SQLOrderDirection ::= ( <ASC> | <DESC> )
```

D.3 Relational Schema For SQL Queries

The normative Relational Schema definition for SQL queries can be found at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/sql/database.sql

The stored procedures that must be supported by the SQL query feature are defined at the following location on the web:
http://www.oasis-open.org/committees/regrep/documents/2.5/sql/storedProcedures.sql

Appendix E Security Implementation Guideline

This section provides a suggested blueprint for how security processing may be implemented in the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have different implementations as long as they support the default security roles and authorization rules described in this document.

E.1 Security Concerns

The security risks broadly stem from the following concerns. After a description of these concerns and potential solutions, we identify the concerns that we address in the current specification

1. Is the content of the registry (data) trustworthy?
    a) How to make sure "what is in the registry" is "what is put there" by the ContentOwner? This concern can be addressed by ensuring that the publisher is authenticated using digital signature (Source Integrity), message is not corrupted during transfer using digital signature (Data Integrity), and the data is not altered by unauthorized subjects based on access control policy (Authorization)
    b) How to protect data while in transmission?
    Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data Confidentiality that can be addressed by encrypting the data in transmission. How to protect against a replay attack?
    c) Is the content up to date? The versioning as well as any time stamp processing, when done securely will ensure the "latest content" is guaranteed to be the latest content.
    d) How to ensure only bona fide responsible organizations add contents to registry? Ensuring Source Integrity (as in 1a).
    e) How to ensure that bona fide publishers add contents to registry only at authorized locations? (System Integrity)
    f) What if the publishers deny modifying certain content after-the-fact? To prevent this (Nonrepudiation) audit trails may be kept which contain signed message digests.
    g) What if the reader denies getting information from the registry?
2. How to provide selective access to registry content? The broad answer is, by using an access control policy – applies to (a), (b), and (c) directly.
    a) How does a ContentOwner restrict access to the content to only specific registry readers?
    b) How can a ContentOwner allow some "partners" (fellow publishers) to modify content?
    c) How to provide selective access to partners the registry usage data?
    d) How to prevent accidental access to data by unauthorized users? Especially with hardware or software failure of the registry security components? The solution to this problem is by having System Integrity.
    e) Data confidentiality of RegistryObject

3. How do we make "who can see what" policy itself visible to limited parties, even excluding the administrator (self & confidential maintenance of access control policy). By making sure there is an access control policy for accessing the policies themselves.
4. How to transfer credentials? The broad solution is to use credentials assertion (such as being worked on in Security Assertions Markup Language (SAML)). Currently, Registry does not support the notion of a session. Therefore, some of these concerns are not relevant to the current specification.
    a) How to transfer credentials (authorization/authentication) to federated registries?
    b) How do aggregators get credentials (authorization/authentication) transferred to them?
    c) How to store credentials through a session?

E.2 Authentication

1. As soon as a message is received, a User object is created.
2. If the message is signed, it is verified (including the validity of the certificate) and the DN of the certificate becomes the identity of the User.
3. If the message is not signed, a User instance is created with the role RegistryGuest. This step is suggested for symmetry and to decouple the rest of the processing.
4. The message is then processed for the Action and the objects it will act on.

E.3 Authorization

For every RegistryObject resource, the Policy Decision Point as defind by [XACML] within the registry will process the AccessControlPolicy object associated with the RegistryObject resource to verify that the requestor subject (e.g. User) is permitted to perform the requested action (e.g. create, update, delete) on the specified resource.

E.4 Registry Bootstrap

When a Registry is newly created, a default User object should be created with the identity of the Registry Administrator's certificate DN with a role RegistryAdministrator. This way, any message signed by the RegistryAdministrator will get all the privileges.

When a Registry is newly created, an instance of AccessControlPolicy is created as the default AccessControlPolicy.

E.5 Content Submission – Client Responsibility

The Registered User must sign the contents before submission – otherwise the content will be rejected.

E.6 Content Submission – Registry Responsibility

1. As with any other request, the Registry Client will first be authenticated. In this case, the User object will get the DN from the certificate.
2. As per the request in the message, the RegistryObject will be created.

3. The RegistryObject is assigned the default AccessControlPolicy.
4. If the Registry Client is not previously registered, the registry may either reject the request or accept it and implicitly register the Registry Client.

E.7 Content Remove/Deprecate – Client Responsibility

The Registry client must sign the header before submission, for authentication purposes; otherwise, the request will be rejected

E.8 Content Remove/Deprecate – Registry Responsibility

1. As with any other request, the Registry Client will first be authenticated. In this case, the User object will get the DN from the certificate.
2. As per the request in the message (remove or deprecate), the appropriate method in the RegistryObject class will be accessed.
3. The access controller performs the authorization by iterating through the Permission objects associated with this object via the default AccessControlPolicy.
4. If authorization succeeds then the action will be permitted. Otherwise an error response is sent back with a suitable AuthorizationException error message.

E.9 Using ds:KeyInfo Field

Two typical usage scenarios for ds:KeyInfo are described below.

Scenario 1

1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature.
5. Registration Authority validates the header signature using the public key from the certificate.
6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the certificate from the KeyInfo field of the payload signature. Note that this step is not an essential one if the onus of validation is that of the eventual user, another Registry Client, of the content.

Scenario 2

1. RC1 signs the payload and SOAP envelope using its private key and publishes to the Registry.
2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.

3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature. This step is required in addition to step 2 because when RC2 retrieves content, it should see RC1's signature with the payload.
4. RC2 retrieves content from the Registry.
5. Registration Authority signs the SOAP envelope using its private key. Registration Authority sends RC1's content and the RC1's signature (signed by RC1).
6. Registration Authority need not send its certificate in the KeyInfo field sinceRC2 is assumed to have obtained the Registration Authority's certificate in an implementation-specific manner and installed it in its local key store.
7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the Registration Authority's signature.
8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates the signature on the payload.

Appendix F Native Language Support (NLS)

F.1 Definitions

Although this section discusses only character set and language, the following terms have to be defined clearly.

F.1.1 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.

F.1.2 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.

F.1.3 Character Set (charset):

- charset is a set of rules for mapping from a sequence of octets to a sequence of characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- A list of registered character sets can be found at [IANA].

F.2 NLS And Request / Response Messages

For the accurate processing of data in both registry client and registry services, it is essential to know which character set is used. Although the body part of the transaction may contain the charset in xml encoding declaration, registry client and registry services shall specify charset parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a text/xml entity is received with the charset parameter omitted, MIME processors and XML processors MUST use the default charset value of "us-ascii". For example:

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

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

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

F.3 NLS And Storing of RegistryObject

This section provides NLS guidelines on how a registry should store RegistryObject instances.

A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple locales. Thus there is no language or character set associated with a specific RegistryObject instance.

A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows. Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in

RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed as LocalizedString sub-elements within the XML element representing the I18N capable attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a specific locale. Each LocalizedString element has a charset and lang attribute as well as a value attribute of type string.

F.3.1 Character Set of *LocalizedString*

The character set used by a locale specific String (LocalizedString) is defined by the charset attribute. It is highly recommended to use UTF-8 or UTF-16 for maximum interoperability.

F.3.2 Language Information of *LocalizedString*

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

F.4 NLS And Storing of Repository Items

This section provides NLS guidelines on how a registry should store repository items.

While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is always associated with a single repository item. The repository item may be in a single locale or may be in multiple locales. This specification does not specify the repository item.

F.4.1 Character Set of Repository Items

The MIME *Content-Type* mime header for the mime multipart containing the repository item MAY contain a *charset* attribute that specifies the character set used by the repository item. For example:

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

It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset of a repository item must be preserved as it is originally specified in the transaction.

F.4.2 Language information of repository item

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

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

13 References

[Bra97] Keywords for use in RFCs to Indicate Requirement Levels.

[ebRIM] ebXML Registry Information Model version 2.5
http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebRIM.pdf

[ebRIM Schema] ebXML Registry Information Model Schema
http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rim.xsd

[ebBPSS] ebXML Business Process Specification Schema
http://www.ebxml.org/specs

[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
http://www.ebxml.org/specs/

[ebMS] ebXML Messaging Service Specification, Version 1.0
http://www.ebxml.org/specs/

[XPT] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[SQL] Structured Query Language (FIPS PUB 127-2)
http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
(SQL/PSM) [ISO/IEC 9075-4:1996]

[IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

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

[RFC 2119] IETF (Internet Engineering Task Force). RFC 2119

[RFC 2130] IETF (Internet Engineering Task Force). RFC 2130

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

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

[RFC2616] RFC 2616:
Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1* . 1999.
http://www.w3.org/Protocols/rfc2616/rfc2616.html

[RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
Internet Security Glossary, ed. R. Shirey. May 2000.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html

[RFC 3023] ETF (Internet Engineering Task Force). RFC 3023:
XML Media Types, ed. M. Murata. 2001.
ftp://ftp.isi.edu/in-notes/rfc3023.txt

[REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
http://www.w3.org/TR/REC-xml

6294 [UUID] DCE 128 bit Universal Unique Identifier
6295 http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
6296 http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

6297 [WSDL] W3C Note. Web Services Description Language (WSDL) 1.1
6298 http://www.w3.org/TR/wsdl

6299 [SOAP11] W3C Note. Simple Object Access Protocol, May 2000,
6300 http://www.w3.org/TR/SOAP

6301 [SOAPAt] W3C Note: SOAP with Attachments, Dec 2000,
6302 http://www.w3.org/TR/SOAP-attachments

6303 [XMLDSIG] XML-Signature Syntax and Processing,
6304 http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

14 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

15 Contact Information

Team Leader

Name:	Kathryn R. Breininger
Company:	The Boeing Company
Street:	P.O. Box 3707 MC 62-LC
City, State, Postal Code:	Seattle, WA 98124-2207
Country:	USA
Phone:	425-965-0182
Email:	kathryn.r.breininger@boeing.com

Editor

Name:	Sally Fuger
Company:	Automotive Industry Action Group
Street:	26200 Lahser Road, Suite 200
City, State, Postal Code:	Southfield, MI 48034
Country:	USA
Phone:	(248) 358-9744
Email:	sfuger@aiag.org

Technical Editor

Name:	Farrukh S. Najmi
Company:	Sun Microsystems
Street:	1 Network Dr., MS BUR02-302
City, State, Postal Code:	Burlington, MA, 01803-0902
Country:	USA
Phone:	(781) 442-9017
Email:	farrukh.najmi@sun.com

Technical Editor

Name:	Nikola Stojanovic
Company:	Metaspaces Consulting
Street:	101 Pineview Terrace
City, State, Postal Code:	Ithaca, NY, 14850
Country:	USA
Phone:	(607) 273-2224
Email:	nikola.stojanovic@acm.org

16 Copyright Statement

Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

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's procedures with respect to rights in OASIS specifications can be found at 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 specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.