



---

Creating A Single Global Electronic Market

1

# **OASIS/ebXML Registry Services Specification v2.5**

**–Committee Approved Specification**

## **OASIS/ebXML Registry Technical Committee**

June 2003

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

## 3 **1 Status of this Document**

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

6 Distribution of this document is unlimited.

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

### 8 ***This version:***

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

10

### 11 ***Latest Technical Committee Approved version:***

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

13

### 14 ***Latest OASIS Approved Standard:***

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

16

## 17 **2 OASIS/ebXML Registry Technical Committee**

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

20 Kathryn Breininger, Boeing  
21 Joseph M. Chiusano, Booz Allen Hamilton  
22 Suresh Damodaran, Sterling Commerce  
23 Sally Fuger, Individual Member  
24 Peter Kacandes, Adobe Systems Incorporated  
25 Michael Kass, NIST  
26 Paul Macias, LMI  
27 Matthew MacKenzie, Individual Member  
28 Monica Martin, Sun Microsystems  
29 Farrukh Najmi, Sun Microsystems  
30 Sanjay Patil, IONA  
31 Nikola Stojanovic, Individual Member  
32 Uday Subarayan, Sun Microsystems

### 33 Contributors

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

36 John Bekisz, Software AG, Inc.  
37 Lisa Carnahan, NIST  
38 Anne Fischer, Individual  
39 Len Gallagher, NIST  
40 Duane Nickull, XML Global  
41 John Silva, Philips Medical  
42 Sekhar Vajjhala, Sun Microsystems

43

44	<b>Table of Contents</b>	
45	<b>1 Status of this Document</b>	<b>3</b>
46	<b>2 OASIS/ebXML Registry Technical Committee</b>	<b>4</b>
47	<b>Table of Contents</b>	<b>5</b>
48	<b>Table of Figures</b>	<b>11</b>
49	<b>Table of Tables</b>	<b>13</b>
50	<b>3 Introduction</b>	<b>14</b>
51	3.1 Summary of Contents of Document	14
52	3.2 General Conventions	14
53	3.3 Audience	14
54	<b>4 Design Objectives</b>	<b>15</b>
55	4.1 Goals	15
56	4.2 Caveats and Assumptions	15
57	<b>5 System Overview</b>	<b>16</b>
58	5.1 What The ebXML Registry Does	16
59	5.2 How The ebXML Registry Works	16
60	5.2.1 Schema Documents Are Submitted	16
61	5.2.2 Business Process Documents Are Submitted	16
62	5.2.3 Seller's Collaboration Protocol Profile Is Submitted	16
63	5.2.4 Buyer Discovers The Seller	16
64	5.2.5 CPA Is Established	17
65	5.3 Registry Users	17
66	5.4 Where the Registry Services May Be Implemented	18
67	5.5 Implementation Conformance	18
68	5.5.1 Conformance as an ebXML Registry	18
69	5.5.2 Conformance as an ebXML Registry Client	19
70	<b>6 ebXML Registry Architecture</b>	<b>20</b>
71	6.1 Registry Service Described	20
72	6.2 Abstract Registry Service	21
73	6.2.1 LifeCycleManager Interface	21
74	6.2.2 QueryManager Interface	22
75	6.3 Concrete Registry Services	22
76	6.4 SOAP Binding	23
77	6.4.1 WSDL Terminology Primer	23
78	6.4.2 Concrete Binding for SOAP	24
79	6.5 ebXML Message Service Binding	24
80	6.5.1 Service and Action Elements	24
81	6.5.2 Synchronous and Asynchronous Responses	24
82	6.5.3 ebXML Registry Collaboration Profiles and Agreements	25
83	6.6 HTTP Binding	26
84	6.6.1 Standard URI Parameters	26
85	6.6.2 QueryManager HTTP Interface	26
86	6.6.3 LifeCycleManager HTTP Interface	28
87	6.6.4 Security Considerations	30
88	6.6.5 Exception Handling	30
89	6.7 Registry Clients	30

90	6.7.1	Registry Client Described .....	30
91	6.7.2	Registry Communication Bootstrapping .....	31
92	6.7.3	RegistryClient Interface .....	32
93	6.7.4	Registry Response.....	32
94	6.8	Interoperability Requirements.....	33
95	6.8.1	Client Interoperability .....	33
96	6.9	Registry Requests and Responses .....	33
97	6.9.1	RegistryRequestType .....	33
98	6.9.2	RegistryResponseType.....	34
99	6.9.3	RegistryResponse.....	35
100	6.9.4	RegistryErrorList .....	35
101	6.9.5	RegistryError .....	36
102	6.9.6	ErrorType .....	36
103	<b>7</b>	<b>Lifecycle Management Service .....</b>	<b>38</b>
104	7.1	Lifecycle of a RegistryObject .....	38
105	7.2	RegistryObject Attributes .....	38
106	7.3	The Submit Objects Protocol.....	39
107	7.3.1	SubmitObjectsRequest .....	39
108	7.3.2	RegistryResponse.....	40
109	7.3.3	Universally Unique ID Generation .....	40
110	7.3.4	ID Attribute And Object References.....	41
111	7.3.5	Audit Trail .....	41
112	7.3.6	Sample SubmitObjectsRequest.....	41
113	7.4	The Update Objects Protocol .....	45
114	7.4.1	UpdateObjectsRequest .....	45
115	7.4.2	Audit Trail .....	46
116	7.5	The Add Slots Protocol.....	46
117	7.5.1	AddSlotsRequest.....	47
118	7.6	The Remove Slots Protocol.....	48
119	7.6.1	RemoveSlotsRequest .....	48
120	7.7	The Approve Objects Protocol.....	49
121	7.7.1	ApproveObjectsRequest.....	50
122	7.7.2	Audit Trail .....	51
123	7.8	The Deprecate Objects Protocol.....	51
124	7.8.1	DeprecateObjectsRequest.....	51
125	7.8.2	Audit Trail .....	52
126	7.9	The Undeprecate Objects Protocol.....	52
127	7.9.1	UndeprecateObjectsRequest .....	53
128	7.9.2	Audit Trail .....	54
129	7.10	The Remove Objects Protocol .....	54
130	7.10.1	RemoveObjectsRequest .....	54
131	<b>8</b>	<b>Query Management Service.....</b>	<b>57</b>
132	8.1	Ad Hoc Query Request/Response .....	57
133	8.1.1	AdhocQueryRequest .....	58
134	8.1.2	AdhocQueryResponse.....	59
135	8.1.3	ReponseOption .....	59
136	8.1.4	Iterative Query Support.....	61
137	8.2	Filter Query Support.....	61

138	8.2.1	FilterQuery .....	63
139	8.2.2	RegistryObjectQuery .....	65
140	8.2.3	RegistryEntryQuery .....	77
141	8.2.4	AssociationQuery.....	80
142	8.2.5	AuditableEventQuery.....	83
143	8.2.6	ClassificationQuery.....	86
144	8.2.7	ClassificationNodeQuery .....	87
145	8.2.8	ClassificationSchemeQuery .....	92
146	8.2.9	RegistryPackageQuery .....	94
147	8.2.10	ExtrinsicObjectQuery.....	96
148	8.2.11	OrganizationQuery.....	98
149	8.2.12	ServiceQuery .....	101
150	8.2.13	FederationQuery .....	104
151	8.2.14	RegistryQuery.....	105
152	8.2.15	SubscriptionQuery .....	106
153	8.2.16	UserQuery.....	107
154	8.2.17	Registry Filters.....	109
155	8.2.18	XML Clause Constraint Representation .....	113
156	8.3	SQL Query Support.....	118
157	8.3.1	SQL Query Syntax Binding To [ebRIM].....	118
158	8.3.2	Semantic Constraints On Query Syntax.....	120
159	8.3.3	SQL Query Results .....	120
160	8.3.4	Simple Metadata Based Queries .....	120
161	8.3.5	RegistryObject Queries .....	121
162	8.3.6	RegistryEntry Queries .....	121
163	8.3.7	Classification Queries .....	121
164	8.3.8	Association Queries .....	122
165	8.3.9	Package Queries.....	123
166	8.3.10	ExternalLink Queries .....	123
167	8.3.11	Audit Trail Queries .....	124
168	8.3.12	Object Export Queries.....	124
169	8.4	Content Retrieval.....	124
170	8.4.1	GetContentRequest .....	125
171	8.4.2	GetContentResponse.....	126
172	8.4.3	Identification Of Content Payloads.....	126
173	8.4.4	GetContentResponse Message Structure .....	126
174	<b>9</b>	<b>Content Management Services .....</b>	<b>129</b>
175	9.1	Content Validation.....	129
176	9.1.1	Content Validation: Use Cases .....	129
177	9.2	Content Cataloging .....	130
178	9.2.1	Content-based Discovery: Use Cases .....	130
179	9.3	Abstract Content Management Service .....	131
180	9.3.1	Inline Invocation Model.....	131
181	9.3.2	De-coupled Invocation Model .....	132
182	9.4	Content Management Service Protocol .....	133
183	9.4.1	ContentManagementServiceRequestType .....	133
184	9.4.2	ContentManagementServiceResponseType .....	135
185	9.5	Publishing / Configuration of a Content Management Service .....	135

186	9.5.1	Multiple Content Management Services and Invocation Control Files	137
187	9.6	Invocation of a Content Management Service .....	137
188	9.6.1	Resolution Algorithm For Service and Invocation Control File .....	137
189	9.6.2	Audit Trail and Cataloged Content .....	138
190	9.6.3	Referential Integrity .....	138
191	9.6.4	Error Handling .....	138
192	9.7	Validate Content Protocol.....	138
193	9.7.1	ValidateContentRequest.....	139
194	9.7.2	ValidateContentResponse .....	140
195	9.8	Catalog Content Protocol.....	141
196	9.8.1	CatalogContentRequest.....	141
197	9.8.2	CatalogContentResponse.....	142
198	9.9	Illustrative Example: Default XML Cataloging Service .....	143
199	9.10	Default XML Content Cataloging Service .....	144
200	9.10.1	Publishing of Default XML Content Cataloging Service .....	145
201	<b>10</b>	<b>Event Notification Service .....</b>	<b>146</b>
202	10.1	Use Cases .....	146
203	10.1.1	CPP Has Changed .....	146
204	10.1.2	New Service is Offered .....	146
205	10.1.3	Monitor Download of Content .....	146
206	10.1.4	Monitor Price Changes.....	146
207	10.1.5	Keep Replicas Consistent With Source Object .....	146
208	10.2	Registry Events .....	147
209	10.3	Subscribing to Events .....	147
210	10.3.1	Event Selection .....	148
211	10.3.2	Notification Action .....	148
212	10.3.3	Subscription Authorization.....	148
213	10.3.4	Subscription Quotas .....	148
214	10.3.5	Subscription Expiration.....	148
215	10.3.6	Subscription Rejection .....	149
216	10.4	Unsubscribing from Events.....	149
217	10.5	Notification of Events.....	149
218	10.6	Retrieval of Events .....	150
219	10.6.1	GetNotificationsRequest .....	150
220	10.6.2	NotificationType .....	150
221	10.6.3	ObjectRefsNotification.....	151
222	10.6.4	ObjectsNotification .....	151
223	10.7	Purging of Events .....	152
224	<b>11</b>	<b>Cooperating Registries Support.....</b>	<b>153</b>
225	11.1	Cooperating Registries Use Cases.....	153
226	11.1.1	Inter-registry Object References .....	153
227	11.1.2	Federated Queries .....	153
228	11.1.3	Local Caching of Data from Another Registry .....	153
229	11.1.4	Object Relocation.....	154
230	11.2	Registry Federations .....	154
231	11.2.1	Federation Metadata.....	154
232	11.2.2	Local Vs. Federated Queries .....	155
233	11.2.3	Federated Lifecycle Management Operations .....	156

234	11.2.4 Federations and Local Caching of Remote Data .....	156
235	11.2.5 Caching of Federation Metadata.....	156
236	11.2.6 Time Synchronization Between Registry Peers .....	156
237	11.2.7 Federations and Security .....	157
238	11.2.8 Federation Lifecycle Management Protocols .....	157
239	11.3 Object Replication .....	158
240	11.3.1 Use Cases for Object Replication .....	158
241	11.3.2 Queries And Replicas.....	159
242	11.3.3 Lifecycle Operations And Replicas .....	159
243	11.3.4 Object Replication and Federated Registries .....	159
244	11.3.5 Creating a Local Replica .....	159
245	11.3.6 Transactional Replication.....	159
246	11.3.7 Keeping Replicas Current .....	160
247	11.3.8 Write Operations on Local Replica.....	160
248	11.3.9 Tracking Location of a Replica .....	160
249	11.3.10 Remote Object References to a Replica .....	160
250	11.3.11 Removing a Local Replica .....	160
251	11.4 Object Relocation Protocol .....	160
252	11.4.1 RelocateObjectsRequest.....	163
253	11.4.2 AcceptObjectsRequest .....	164
254	11.4.3 Object Relocation and Remote ObjectRefs .....	164
255	11.4.4 Notification of Object Relocation To ownerAtDestination .....	165
256	11.4.5 Notification of Object Commit To sourceRegistry.....	165
257	11.4.6 Object Relocation and Timeouts.....	166
258	<b>12 Registry Security .....</b>	<b>167</b>
259	12.1 Security Concerns.....	167
260	12.2 Integrity of Registry Content .....	167
261	12.2.1 Message Payload Signature.....	167
262	12.2.2 Payload Signature Requirements .....	168
263	12.3 Authentication .....	169
264	12.3.1 Message Header Signature .....	170
265	12.4 Key Distribution and KeyInfo Element.....	171
266	12.5 Confidentiality.....	172
267	12.5.1 On-the-wire Message Confidentiality .....	172
268	12.5.2 Confidentiality of Registry Content.....	172
269	12.6 Access Control and Authorization .....	172
270	12.6.1 Actors / Role Mapping .....	172
271	<b>Appendix A Web Service Architecture .....</b>	<b>174</b>
272	A.1 Registry Service Abstract Specification .....	174
273	A.2 Registry Service SOAP Binding .....	174
274	<b>Appendix B ebXML Registry Schema Definitions.....</b>	<b>175</b>
275	B.1 RIM Schema.....	175
276	B.2 Registry Services Interface Base Schema.....	175
277	B.3 QueryManager Service Schema .....	175
278	B.4 LifecycleManager Service Schema .....	175
279	B.5 Content Management Service Schema .....	175
280	B.6 Examples of Instance Documents .....	175

281	<b>Appendix C</b>	<b>Interpretation of UML Diagrams.....</b>	<b>176</b>
282	C.1	UML Class Diagram.....	176
283	C.2	UML Sequence Diagram .....	176
284	<b>Appendix D</b>	<b>SQL Query .....</b>	<b>177</b>
285	D.1	SQL Query Syntax Specification .....	177
286	D.2	Non-Normative BNF for Query Syntax Grammar.....	177
287	D.3	Relational Schema For SQL Queries.....	178
288	<b>Appendix E</b>	<b>Security Implementation Guideline .....</b>	<b>179</b>
289	E.1	Security Concerns.....	179
290	E.2	Authentication .....	180
291	E.3	Authorization.....	180
292	E.4	Registry Bootstrap .....	180
293	E.5	Content Submission – Client Responsibility .....	180
294	E.6	Content Submission – Registry Responsibility.....	180
295	E.7	Content Remove/Deprecate – Client Responsibility .....	181
296	E.8	Content Remove/Deprecate – Registry Responsibility .....	181
297	E.9	Using ds:KeyInfo Field.....	181
298	<b>Appendix F</b>	<b>Native Language Support (NLS) .....</b>	<b>183</b>
299	F.1	Definitions.....	183
300	F.1.1	Coded Character Set (CCS): .....	183
301	F.1.2	Character Encoding Scheme (CES):.....	183
302	F.1.3	Character Set (charset):.....	183
303	F.2	NLS And Request / Response Messages .....	183
304	F.3	NLS And Storing of RegistryObject .....	183
305	F.3.1	Character Set of <i>LocalizedString</i> .....	184
306	F.3.2	Language Information of <i>LocalizedString</i> .....	184
307	F.4	NLS And Storing of Repository Items .....	184
308	F.4.1	Character Set of Repository Items .....	184
309	F.4.2	Language information of repository item .....	184
310	<b>13</b>	<b>References .....</b>	<b>185</b>
311	<b>14</b>	<b>Disclaimer .....</b>	<b>187</b>
312	<b>15</b>	<b>Contact Information.....</b>	<b>188</b>
313	<b>16</b>	<b>Copyright Statement .....</b>	<b>189</b>
314			

## 315 Table of Figures

316	Figure 1: Actor Relationships .....	18
317	Figure 2: ebXML Registry Service Architecture .....	20
318	Figure 3: The Abstract ebXML Registry Service .....	21
319	Figure 4: A Concrete ebXML Registry Service.....	23
320	Figure 5: Registry Architecture Supports Flexible Topologies .....	31
321	Figure 6: RegistryRequestType Syntax.....	33
322	Figure 7: RegistryResponseType Syntax.....	34
323	Figure 8: RegistryErrorList Syntax .....	35
324	Figure 9: RegistryError Syntax .....	36
325	Figure 10: Lifecycle of a RegistryObject .....	38
326	Figure 11: Submit Objects Sequence Diagram .....	39
327	Figure 12: SubmitObjectsRequest Syntax .....	39
328	Figure 13: Update Objects Sequence Diagram .....	45
329	Figure 14: UpdateObjectsRequest Syntax .....	45
330	Figure 15: Add Slots Sequence Diagram.....	47
331	Figure 16: AddSlotsRequest Syntax.....	47
332	Figure 17: Remove Slots Sequence Diagram .....	48
333	Figure 18: RemoveSlotsRequest Syntax .....	49
334	Figure 19: Approve Objects Sequence Diagram.....	50
335	Figure 20: ApproveObjectsRequest Syntax.....	50
336	Figure 21: Deprecate Objects Sequence Diagram.....	51
337	Figure 22: DeprecateObjectsRequest Syntax.....	52
338	Figure 23: Undeprecate Objects Sequence Diagram.....	53
339	Figure 24: UndeprecateObjectsRequest Syntax.....	53
340	Figure 25: Remove Objects Sequence Diagram .....	54
341	Figure 26: RemoveObjectsRequest Syntax .....	55
342	Figure 27: Submit Ad Hoc Query Sequence Diagram .....	57
343	Figure 28: AdhocQueryRequest Syntax .....	58
344	Figure 29: AdhocQueryResponse Syntax.....	59
345	Figure 30: ResponseOption Syntax .....	60
346	Figure 31: Example ebRIM Binding .....	62
347	Figure 32: ebRIM Binding for RegistryObjectQuery .....	65
348	Figure 33: ebRIM Binding for RegistryEntryQuery .....	78
349	Figure 34: ebRIM Binding for AssociationQuery.....	81
350	Figure 35: ebRIM Binding for AuditableEventQuery.....	83
351	Figure 36: ebRIM Binding for ClassificationQuery.....	86
352	Figure 37: ebRIM Binding for ClassificationNodeQuery .....	88
353	Figure 38: ebRIM Binding for ClassificationSchemeQuery .....	93
354	Figure 39: ebRIM Binding for RegistryPackageQuery.....	94
355	Figure 40: ebRIM Binding for ExtrinsicObjectQuery.....	96

356	Figure 41: ebRIM Binding for OrganizationQuery.....	98
357	Figure 42: ebRIM Binding for ServiceQuery .....	102
358	Figure 43: ebRIM Binding for FederationQuery .....	104
359	Figure 44: ebRIM Binding for RegistryQuery.....	105
360	Figure 45: ebRIM Binding for SubscriptionQuery .....	106
361	Figure 46: ebRIM Binding for OrganizationQuery Add PersonName under User and line up	
362	107	
363	Figure 47: The Clause Structure.....	113
364	Figure 48: Content Retrieval Sequence Diagram.....	125
365	Figure 49: GetContentRequest Syntax .....	125
366	Figure 50: GetContentResponse Syntax.....	126
367	Figure 51: Content Validation Service .....	129
368	Figure 52: Content Cataloging Service .....	130
369	Figure 53: Content Management Service: Inline Invocation Model.....	132
370	Figure 54: Content Management Service: De-coupled Invocation Model.....	133
371	Figure 55: ContentManagementServiceRequestType Syntax .....	134
372	Figure 56: Content ContentManagementServiceResponseType Syntax.....	135
373	Figure 57: Cataloging Service Configuration .....	136
374	Figure 58: Validate Content Protocol.....	139
375	Figure 59: ValidateContentRequest Syntax.....	139
376	Figure 60: ValidateContentResponse Syntax .....	140
377	Figure 61: Catalog Content Protocol.....	141
378	Figure 62: CatalogContentRequest Syntax.....	142
379	Figure 63: CatalogContentResponse Syntax .....	143
380	Figure 64: Example of CPP cataloging using Default XML Cataloging Service.....	144
381	Figure 65: GetNotificationsRequest Syntax .....	150
382	Figure 66: NotificationType Syntax.....	151
383	Figure 67: ObjectRefsNotification Syntax .....	151
384	Figure 68: ObjectsNotification Syntax .....	152
385	Figure 69: Inter-registry Object References .....	153
386	Figure 70: Registry Federations .....	154
387	Figure 71: Federation Metadata Example.....	155
388	Figure 72: Object Replication .....	158
389	Figure 73: Object Relocation .....	161
390	Figure 74: Relocate Objects Protocol.....	162
391	Figure 75: RelocateObjectsRequest XML Schema.....	163
392	Figure 76: AcceptObjectsRequest Syntax .....	164
393		

**394 Table of Tables**

395	Table 1: Registry Users .....	17
396	Table 2: LifeCycle Manager Summary .....	21
397	Table 3: Query Manager .....	22
398	Table 4: Standard URI Parameters .....	26
399	Table 5: QueryManager HTTP Interface .....	27
400	Table 6: LifeCycleManager HTTP Interface .....	29
401	Table 7: RegistryClient Summary .....	32
402	Table 8: Path Filter Expressions for Use Cases .....	91
403	Table 9: Default Actor to Role Mappings .....	173
404		

## 405 3 Introduction

### 406 3.1 Summary of Contents of Document

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

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

### 412 3.2 General Conventions

413 The following conventions are used throughout this document:

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

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

419 The term “*ExtrinsicObject*” is used to refer to an object that provides metadata about a *repository*  
420 *item*.

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

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

### 426 3.3 Audience

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

- 428 • Implementers of ebXML Registry Services
- 429 • Implementers of ebXML Registry Clients

#### 430 Related Documents

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

- 432 a) *ebXML Registry Information Model* [ebRIM]
- 433 b) *ebXML Message Service Specification* [ebMS]
- 434 c) *ebXML Business Process Specification Schema* [ebBPSS]
- 435 d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

## 436 **4 Design Objectives**

### 437 **4.1 Goals**

438 The goals of this version of the specification are to:

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

### 443 **4.2 Caveats and Assumptions**

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

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

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

## 454 **5 System Overview**

### 455 **5.1 What The ebXML Registry Does**

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

### 460 **5.2 How The ebXML Registry Works**

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

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

#### 470 **5.2.1 Schema Documents Are Submitted**

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

#### 474 **5.2.2 Business Process Documents Are Submitted**

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

#### 478 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

#### 483 **5.2.4 Buyer Discovers The Seller**

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

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

### 489 5.2.5 CPA Is Established

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

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

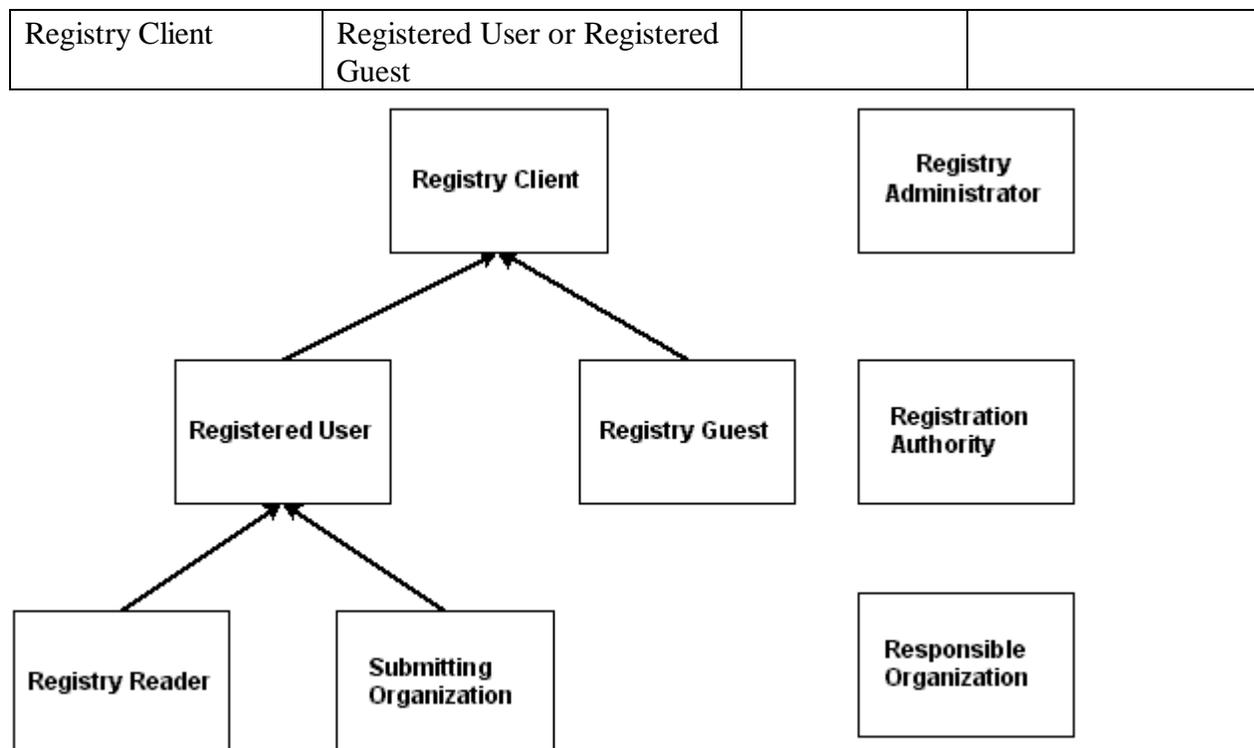
### 496 5.3 Registry Users

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

500

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 <i>read</i> access		
Responsible Organization	Creates Registry Objects	Responsible Organization (RO)	RO MAY have the same identity as SO



501  
502

Figure 1: Actor Relationships

503 Note:

504 In the current version of the specification the following are true.

505 A Submitting Organization and a Responsible Organization are the same.

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

507 A Registry Administrator and Registration Authority are the same.

## 508 5.4 Where the Registry Services May Be Implemented

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

## 511 5.5 Implementation Conformance

512 An implementation is a *conforming* ebXML Registry if the implementation meets the conditions  
513 in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the  
514 implementation meets the conditions in Section 5.5.2. An implementation is a conforming  
515 ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to  
516 the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming  
517 ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and  
518 Registry Client.

### 519 5.5.1 Conformance as an ebXML Registry

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

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

### 526 **5.5.2 Conformance as an ebXML Registry Client**

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

- 529 1. Supports the ebXML CPA and bootstrapping process.  
530 2. Supports the syntax and the semantics of the Registry Client Interfaces.  
531 3. Supports the defined ebXML Error Message DTD.  
532 4. Supports the defined ebXML Registry Schema (Appendix B).  
533

## 534 **6 ebXML Registry Architecture**

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

538  
539

**Figure 2: ebXML Registry Service Architecture**

### 540 **6.1 Registry Service Described**

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

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

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

## 551 6.2 Abstract Registry Service

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

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

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

559 The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide  
 560 two key functional interfaces called **QueryManager**<sup>1</sup> (QM) and **LifeCycleManager**<sup>2</sup>  
 561 (LM).



562  
563 **Figure 3: The Abstract ebXML Registry Service**

564 0 provides hyperlinks to the abstract service definition in the Web Service Description Language  
 565 (WSDL) syntax.

### 566 6.2.1 LifeCycleManager Interface

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

572 **Table 2: LifeCycle Manager Summary**

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

<sup>1</sup> Known as ObjectQueryManager in V1.0

<sup>2</sup> Known as ObjectManager in V1.0

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

## 573 6.2.2 QueryManager Interface

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

577 **Table 3: Query Manager**

<b>Method Summary of QueryManager</b>	
GetContentResponse	<u>getContent</u> ( <u>GetContentRequest</u> req) Submit an ad hoc query request. This method is being deprecated and may go away in version 4.
GetNotificationsResponse	<u>getNotifications</u> ( <u>GetNotificationsRequest</u> req) Submit a request to get event notifications.
AdhocQueryResponse	<u>submitAdhocQuery</u> ( <u>AdhocQueryRequest</u> req) Submit an ad hoc query request.
RegistryObject	<u>getRegistryObject</u> ( <u>String</u> id) Submit a request to get the RegistryObject that matches the specified id.
RepositoryItem	<u>getRepositoryItem</u> ( <u>String</u> 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.

578

## 579 6.3 Concrete Registry Services

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

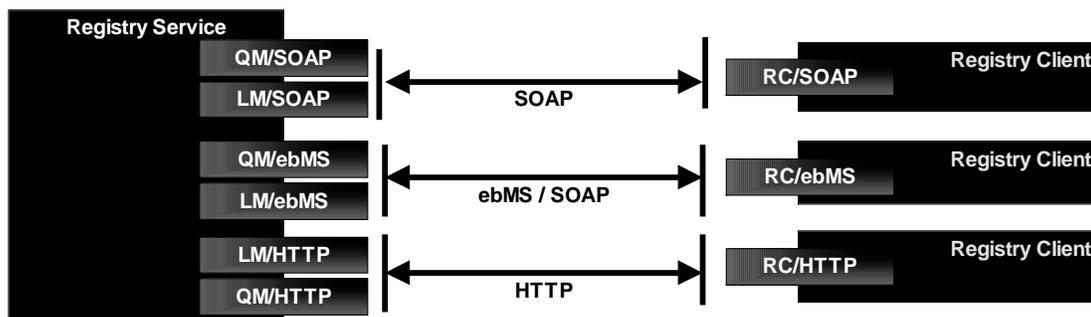
- 582 • Implementations of the interfaces defined by the abstract registry service.
- 583 • Bindings of these concrete interfaces to specific communication protocols.

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

- 585 • A SOAP binding using the HTTP protocol
- 586 • An ebXML Messaging Service (ebMS) binding
- 587 • An HTTP binding

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

591



592

593

Figure 5: A Concrete ebXML Registry Service

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

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

## 601 6.4 SOAP Binding

### 602 6.4.1 WSDL Terminology Primer

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

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

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

## 615 6.4.2 Concrete Binding for SOAP

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

- 618 • A single service element with name “RegistryService” defines the concrete SOAP binding  
619 for the registry service.
- 620 • The service element includes two port definitions, where each port corresponds with one of  
621 the interfaces defined for the abstract registry service. Each port includes an HTTP URL for  
622 accessing that port.
- 623 • Each port definition also references a binding element, one for each interface defined in the  
624 WSDL for the abstract registry service.

```
625  
626 <service name = "RegistryService">  
627   <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">  
628     <soap:address location = "http://your_URL_to_your_QueryManager"/>  
629   </port>  
630  
631   <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">  
632     <soap:address location = "http://your_URL_to_your_QueryManager"/>  
633   </port>  
634 </service>  
635
```

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

## 637 6.5 ebXML Message Service Binding

### 638 6.5.1 Service and Action Elements

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

- 641 • The value of the Service element in the MessageHeader is an ebXML Registry Service  
642 interface name (e.g., “LifeCycleManager”). The type attribute of the Service element should  
643 have a value of “ebXMLRegistry”.
- 644 • The value of the Action element in the MessageHeader is an ebXML Registry Service  
645 method name (e.g., “submitObjects”).

```
646  
647 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>  
648 <eb:Action>submitObjects</eb:Action>  
649
```

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

### 653 6.5.2 Synchronous and Asynchronous Responses

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

### 655 **Asynchronous response**

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

- 659 • MessageHeader
- 660 • RegistryResponse element including:
  - 661 ○ status attribute with value **Unavailable**

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

- 666 • MessageHeader
- 667 • RegistryResponse element including:
  - 668 ○ Status attribute (Success, Failure)
  - 669 ○ Optional RegistryErrorList

### 670 **Synchronous response**

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

- 673 • MessageHeader
- 674 • RegistryResponse element including:
  - 675 ○ Status attribute (Success, Failure)
  - 676 ○ Optional RegistryErrorList

## 677 **6.5.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

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

## 689 6.6 HTTP Binding

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

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

### 697 6.6.1 Standard URI Parameters

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

700

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

701

Table 4: Standard URI Parameters

702

### 703 6.6.2 QueryManager HTTP Interface

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

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

708

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**

709

710

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

### 713 **Sample getRegistryObject Request**

714 The following example shows a getRegistryObject request.

715

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

718

### 719 **Sample getRegistryObject Response**

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

722

```
723 HTTP/1.1 200 OK
724 Content-Type: text/xml
725 Content-Length: 555
726
727 <?xml version="1.0"?>
728 <ExtrinsicObject id = "urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6"
729   objectType="urn:uuid:32bbb291-0291-486d-a80d-cdd6cd625c57">
730   <Name>
731     <LocalizedString value = "Sample Object"/>
732   </Name>
733 </ExtrinsicObject>
```

734

### 735 **Sample getRepositoryItem Request**

736 The following example shows a getRepositoryItem request.

737

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

740

**741 Sample getRepositoryItem Response**

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

```
744  
745 HTTP/1.1 200 OK  
746 Content-Type: text/xml  
747 Content-Length: 555  
748  
749 <?xml version="1.0"?>  
750 < CollaborationProtocolProfile>  
751 ...  
752 </CollaborationProtocolProfile>
```

753

**754 Sample submitAdhocQueryRequest Request**

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

```
757  
758 POST /http?interface=QueryManager&method=submitAdhocQueryRequest HTTP/1.1  
759 User-Agent: Foo-ebXML/1.0  
760 Host: www.registryserver.com  
761 Content-Type: text/xml  
762 Content-Length: 555  
763  
764 <?xml version="1.0"?>  
765 <AdhocQueryRequest>  
766 ...  
767 </AdhocQueryRequest>
```

768

**769 Sample submitAdhocQueryRequest Response**

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

```
772  
773 HTTP/1.1 200 OK  
774 Content-Type: text/xml  
775 Content-Length: 555  
776  
777 <?xml version="1.0"?>  
778 <AdhocQueryResponse>  
779 ...  
780 </AdhocQueryResponse>
```

781

**782 6.6.3 LifeCycleManager HTTP Interface**

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

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

787

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

788 **Table 6: LifeCycleManager HTTP Interface**

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

791 **Sample submitObjects Request**

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

```
794
795 POST /http?interface=LifeCycleManager&method=submitObjects HTTP/1.1
796 User-Agent: Foo-ebXML/1.0
797 Host: www.registryserver.com
798 Content-Type: text/xml
799 Content-Length: 555
800
801 <?xml version="1.0"?>
802 <SubmitObjectsRequest>
803 ...
804 </SubmitObjectRequest>
```

805

806 **Sample submitObjects Response**

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

```
809
810 HTTP/1.1 200 OK
811 Content-Type: text/xml
812 Content-Length: 555
813
814 <?xml version="1.0"?>
815 <RegistryResponse>
816 ...
817 </RegistryResponse>
```

818

## 819 **6.6.4 Security Considerations**

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

## 826 **6.6.5 Exception Handling**

827 Exception handling is consistent with exception handling in other registry interface bindings.  
828 Errors must be reported in a RegistryErrorList, and sent back to the client on the same  
829 connection as the request.  
830 When errors occur, the HTTP status code and message should correspond to the error(s) being  
831 reported in the RegistryErrorList. For example, if the RegistryErrorList reports that an  
832 object wasn't found, therefore cannot be returned, an appropriate error code should be 404, with a  
833 message of "ObjectNotFoundException". A detailed list of HTTP status codes can be found in  
834 [RFC2616]. The mapping between registry exceptions and HTTP status codes is currently  
835 unspecified.  
836

## 837 **6.7 Registry Clients**

### 838 **6.7.1 Registry Client Described**

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

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

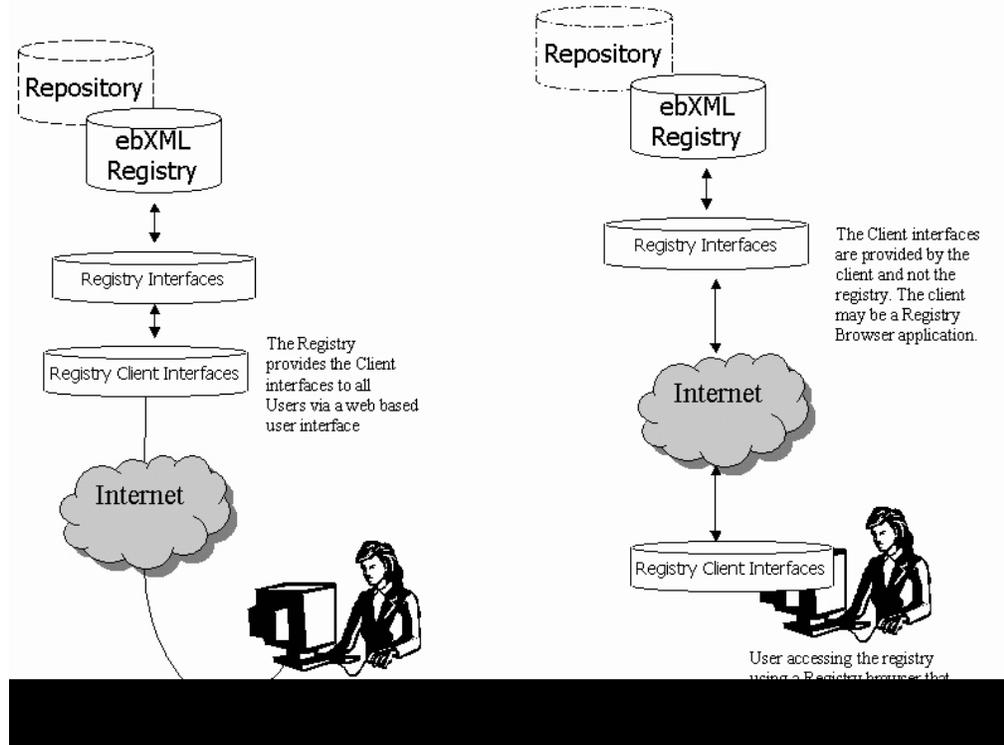


Figure 7: Registry Architecture Supports Flexible Topologies

854  
855

856 **6.7.2 Registry Communication Bootstrapping**

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

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

873

874 **Communication Bootstrapping for SOAP Binding**

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

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

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

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

#### 884 **Communication Bootstrapping for ebXML Message Service Binding**

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

#### 895 **Communication Bootstrapping for HTTP Binding**

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

### 898 **6.7.3 RegistryClient Interface**

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

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

905 **Table 7: RegistryClient Summary**

#### **Method Summary of RegistryClient**

void	<b>onResponse</b> ( <u>RegistryResponse</u> resp) Notifies client of the response sent by registry to previously submitted request.
------	--

### 906 **6.7.4 Registry Response**

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

## 909 6.8 Interoperability Requirements

### 910 6.8.1 Client Interoperability

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

## 914 6.9 Registry Requests and Responses

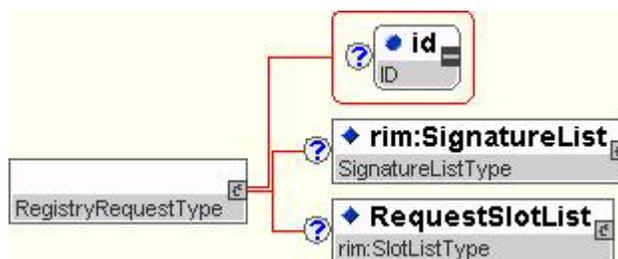
915 This section describes the generic aspects that are common to all requests/responses  
 916 sent/received by registry clients/registry to the registry/registry clients.

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

### 924 6.9.1 RegistryRequestType

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

926 **Syntax:**



927  
 928 **Figure 9: RegistryRequestType Syntax**

929 **Parameters:**

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

939

940 **Returns:**

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

943 **Exceptions:**

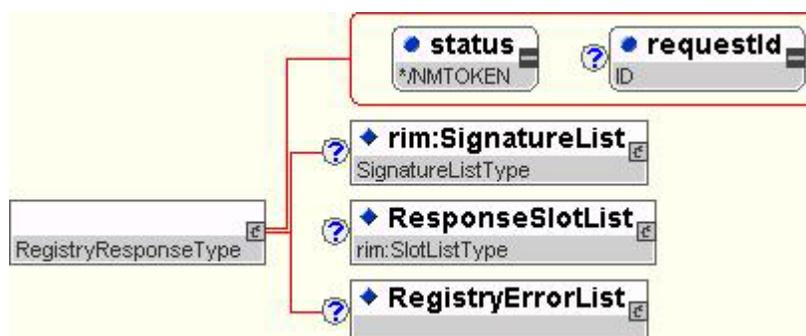
944 The following exceptions are common to all requests:

- 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.

- 970
- 971
- 972
- 973
- 974
- 975
- 976
- 977
- 978
- 979
- 980
- 981
- 982
- 983
- 984
- 985
- 986
- *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.

### 987 6.9.3 RegistryResponse

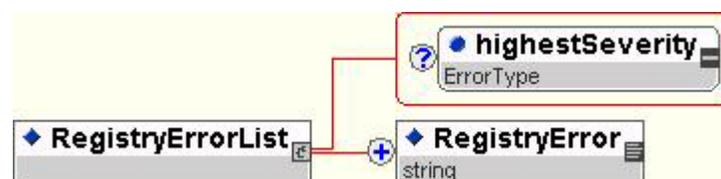
988 RegistryResponse is an element whose base type is RegistryResponseType. It adds no additional  
 989 elements or attributes beyond those described in RegistryResponseType. RegistryResponse is  
 990 used by many requests as their response.

### 991 6.9.4 RegistryErrorList

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

994

#### 995 Syntax:



996  
 997

Figure 11: RegistryErrorList Syntax

#### 998 Parameters:

- 999
- 1000
- 1001
- 1002
- 1003
- *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

1004 registry processes a request. RegistryError is defined in 6.9.5.

1005

## 1006 6.9.5 RegistryError

1007 A RegistryError specifies an error or warning message that is encountered while the registry  
1008 processes a request.

1009

1010 **Syntax:**



1011

1012

Figure 12: RegistryError Syntax

1013 **Parameters:**

- 1014     ▪ *codeContext*: This parameter specifies a string that indicates contextual text that  
1015       provides additional detail to the errorCode. For example, if the errorCode is  
1016       InvalidRequestException the codeContext may provide the reason why the  
1017       request was invalid.
- 1018     ▪ *errorCode*: This parameter specifies a string that indicates the error that was  
1019       encountered. Implementations must set this parameter to the Exception or Error as  
1020       defined by this specification (e.g. InvalidRequestException).
- 1021     ▪ *location*: This parameter specifies a string that indicated where in the code the  
1022       error occurred. Implementations should show the stack trace and/or, code module  
1023       and line number information where the error was encountered in code.
- 1024     ▪ *severity*: This parameter specifies an enumerated value of type ErrorType which  
1025       indicates the severity of error that was encountered. ErrorType is described in  
1026       6.9.6.

1027

## 1028 6.9.6 ErrorType

1029 The ErrorType type defines a set of enumerated values that indicate the different type of errors  
1030 that a registry may encounter while processing a request. The possible values are Warning and  
1031 Error.

### 1032 Warning

1033 A Warning is a non-fatal error encountered by the registry while processing a request. A registry  
1034 must return a status of Success in the RegistryResponse for a request that only encountered  
1035 Warnings during its processing and encountered no Errors.

### 1036 Error

1037 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

## 1042 7 Lifecycle Management Service

1043 This section defines the Lifecycle Management service of the Registry. The Lifecycle  
 1044 Management Service is a sub-service of the Registry service. It provides the functionality  
 1045 required by RegistryClients to manage the lifecycle of repository items (e.g. XML documents  
 1046 required for ebXML business processes). The Lifecycle Management Service can be used with  
 1047 all types of repository items as well as the metadata objects specified in [ebRIM] such as  
 1048 Classification and Association.

1049 The minimum-security policy for an ebXML registry is to accept content from any client if a  
 1050 certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the  
 1051 content.

### 1052 7.1 Lifecycle of a RegistryObject

1053 The main purpose of the LifeCycleManagement service is to manage the lifecycle of  
 1054 RegistryObjects. Figure 13 shows the typical lifecycle of a RegistryObject.

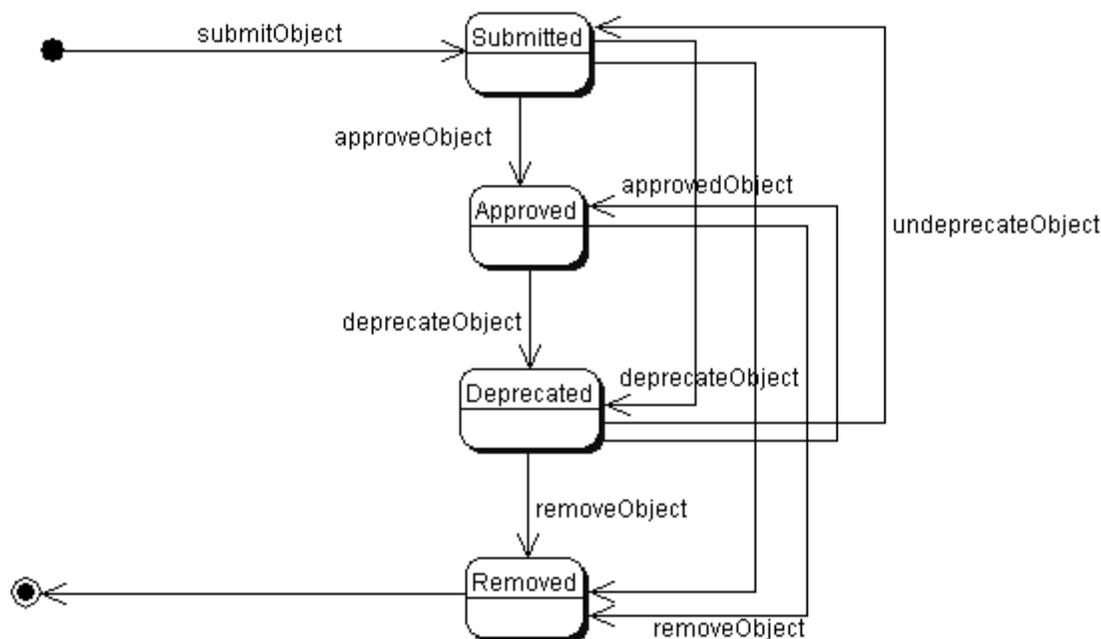


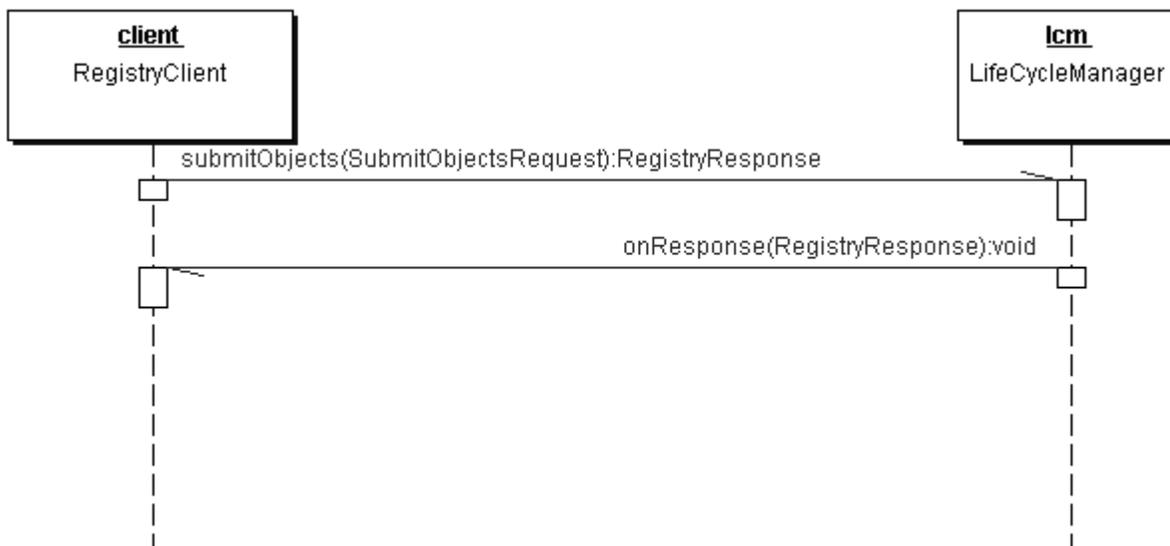
Figure 13: Lifecycle of a RegistryObject

### 1057 7.2 RegistryObject Attributes

1058 A repository item is associated with a set of standard metadata defined as attributes of the  
 1059 RegistryObject class and its sub-classes as described in [ebRIM]. These attributes reside outside  
 1060 of the actual repository item and catalog descriptive information about the repository item. XML  
 1061 elements called ExtrinsicObject and other elements (See Appendix B.1 for details) encapsulate  
 1062 all object metadata attributes defined in [ebRIM] as XML attributes.

1063 **7.3 The Submit Objects Protocol**

1064 This section describes the protocol of the Registry Service that allows a RegistryClient to submit  
 1065 one or more RegistryObjects and/or repository items using the LifeCycleManager on behalf of a  
 1066 Submitting Organization. It is expressed in UML notation as described in Appendix C.



1067  
1068

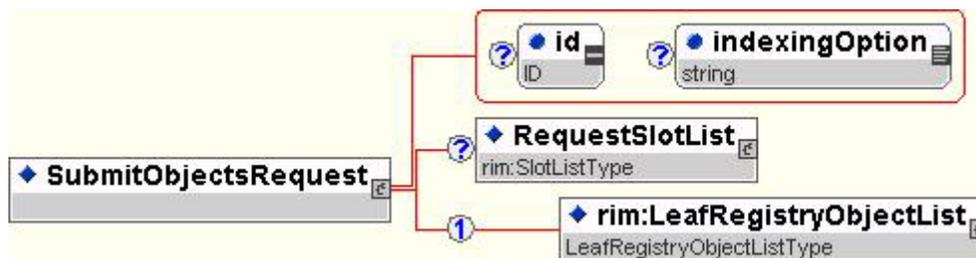
**Figure 15: Submit Objects Sequence Diagram**

1069 For details on the schema for the Business documents shown in this process refer to Appendix B.

1070 **7.3.1 SubmitObjectsRequest**

1071 The SubmitObjectsRequest is used by a client to submit RegistryObjects and/or repository items  
 1072 to the registry.

1073 **Syntax:**



1074  
1075

**Figure 16: SubmitObjectsRequest Syntax**

1076 **Parameters:**

1077

- 1078 ▪ *LeafRegistryObjectsList*: This parameter specifies a collection of RegistryObject  
 1079 instances that are being submitted to the registry. The RegistryObjects in the list  
 1080 may be brand new objects being submitted to the registry or they may be current  
 1081 objects already existing in the registry. In case of existing objects the registry

1082 must treat them in the same manner as UpdateObjectsRequest and simply update  
1083 the existing objects.

1084

1085 **Returns:**

1086 This request returns a RegistryResponse. See section 7.3.2 for details.

1087 **Exceptions:**

1088 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1089       ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an  
1090       operation for which she was not authorized.
- 1091       ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object  
1092       within the request that was not found.
- 1093       ▪ *ObjectExistsException*: Indicates that the requestor tried to submit an object using  
1094       an id that matched the id of an existing object in the registry.
- 1095       ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an  
1096       operation which was semantically invalid.
- 1097       ▪ *UnsupportedCapabilityException*: Indicates that the requestor attempted to  
1098       submit some content that is not supported by the registry.
- 1099       ▪ *QuotaExceededException*: Indicates that the requestor attempted to submit more  
1100       content than the quota allowed for them by the registry.

1101

## 1102 **7.3.2 RegistryResponse**

1103 The RegistryResponse is sent by the registry as a response to several different requests. It is a  
1104 simple response that can signal the status of a request and any errors or exceptions that may have  
1105 occurred during the processing of that request. The details of RegistryResponse are described by  
1106 the RegistryResponseType in 6.9.2.

## 1107 **7.3.3 Universally Unique ID Generation**

1108 As specified by [ebRIM], all objects in the registry have a unique id contained within the value  
1109 of the “id” attribute of a RegistryObject instance. The id must be a Universally Unique Identifier  
1110 (UUID) and must conform to the format of a URN that specifies a DCE 128 bit UUID as  
1111 specified in [UUID].

1112       (e.g. urn:uuid:a2345678-1234-1234-123456789012)

1113 The registry usually generates this id. The client may optionally supply the id attribute for  
1114 submitted objects. If the client supplies the id and it conforms to the format of a URN that  
1115 specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the id  
1116 for the object. In this case, the registry must honour a client-supplied id and use it as the id  
1117 attribute of the object in the registry. If the id is not unique within the registry, the registry must  
1118 return ObjectExistsException.

1119 If the client does not supply an id for a submitted object then the registry must generate a  
 1120 universally unique id . Whether the client generates the id or whether the registry generates it, it  
 1121 must be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

### 1122 7.3.4 ID Attribute And Object References

1123 The id attribute of an object may be used by other objects to reference the first object. Such  
 1124 references are common both within the SubmitObjectsRequest as well as within the registry.  
 1125 Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the  
 1126 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the  
 1127 SubmitObjectsRequest that needs to be referred to within the request document may be assigned  
 1128 an id by the submitter so that it can be referenced within the request. The submitter may give the  
 1129 object a proper UUID URN, in which case the id is permanently assigned to the object within the  
 1130 registry. Alternatively, the submitter may assign an arbitrary id (not a proper UUID URN) as  
 1131 long as the id is a unique anyURI value within the request document. In this case the id serves as  
 1132 a linkage mechanism within the request document but must be ignored by the registry and  
 1133 replaced with a registry generated UUID upon submission.

1134 When an object in a SubmitObjectsRequest needs to reference an object that is already in the  
 1135 registry, the request must contain an ObjectRef whose id attribute is the id of the object in the  
 1136 registry. This id is by definition a proper UUID URN. An ObjectRef may be viewed as a proxy  
 1137 within the request for an object that is in the registry.

### 1138 7.3.5 Audit Trail

1139 The RS must create AuditableEvent objects with eventType *Created* for each RegistryObject  
 1140 created via a SubmitObjectsRequest.

### 1141 7.3.6 Sample SubmitObjectsRequest

1142 The following example shows several different use cases in a single SubmitObjectsRequest. It  
 1143 does not show the complete SOAP or [ebMS] Message with the message header and additional  
 1144 payloads in the message for the repository items.

1145 A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects  
 1146 that are being submitted. It may also contain any number of ObjectRefs to link objects being  
 1147 submitted to objects already within the registry.

```

1148 <?xml version = "1.0" encoding = "UTF-8"?>
1149 <SubmitObjectsRequest
1150   xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
1151   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
1152   xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
1153 spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
1154 file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
1155   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
1156   xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
1157 >
1158   <rim:LeafRegistryObjectList>
1159
1160     <!--
1161     The following 3 objects package specified ExtrinsicObject in specified
1162     RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
1163     being submitted
1164     -->
1165
1166   </rim:LeafRegistryObjectList>
1167
  
```

```

1168 <rim:RegistryPackage id = "acmePackage1" >
1169   <rim:Name>
1170     <rim:LocalizedString value = "RegistryPackage #1"/>
1171   </rim:Name>
1172   <rim:Description>
1173     <rim:LocalizedString value = "ACME's package #1"/>
1174   </rim:Description>
1175 </rim:RegistryPackage>
1176
1177 <rim:ExtrinsicObject id = "acmeCPP1" >
1178   <rim:Name>
1179     <rim:LocalizedString value = "Widget Profile" />
1180   </rim:Name>
1181   <rim:Description>
1182     <rim:LocalizedString value = "ACME's profile for selling widgets" />
1183   </rim:Description>
1184 </rim:ExtrinsicObject>
1185
1186 <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
1187 = "acmePackage1" targetObject = "acmeCPP1" />
1188
1189 <!--
1190   The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
1191   Where the RegistryPackage is being submitted and the ExtrinsicObject is
1192   already in registry
1193   -->
1194
1195 <rim:RegistryPackage id = "acmePackage2" >
1196   <rim:Name>
1197     <rim:LocalizedString value = "RegistryPackage #2"/>
1198   </rim:Name>
1199   <rim:Description>
1200     <rim:LocalizedString value = "ACME's package #2"/>
1201   </rim:Description>
1202 </rim:RegistryPackage>
1203
1204 <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
1205
1206 <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
1207 sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
1208
1209 <!--
1210   The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
1211   where the RegistryPackage and the ExtrinsicObject are already in registry
1212   -->
1213
1214 <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
1215 <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
1216
1217 <!-- id is unspecified implying that registry must create a uuid for this object -->
1218
1219 <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
1220 123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
1221
1222 <!--
1223   The following 3 objects externally link specified ExtrinsicObject using
1224   specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
1225   are being submitted
1226   -->
1227
1228 <rim:ExternalLink id = "acmeLink1" externalURI="http://www.acme.com">
1229   <rim:Name>
1230     <rim:LocalizedString value = "Link #1"/>
1231   </rim:Name>
1232   <rim:Description>
1233     <rim:LocalizedString value = "ACME's Link #1"/>
1234   </rim:Description>
1235 </rim:ExternalLink>
1236
1237 <rim:ExtrinsicObject id = "acmeCPP2" >
1238   <rim:Name>
1239     <rim:LocalizedString value = "Sprockets Profile" />
1240   </rim:Name>

```

```

1241     <rim:Description>
1242       <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
1243     </rim:Description>
1244   </rim:ExtrinsicObject>
1245
1246   <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
1247   sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
1248
1249   <!--
1250   The following 2 objects externally link specified ExtrinsicObject using specified
1251   ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
1252   is already in registry. Note that the targetObject points to an ObjectRef in a
1253   previous line
1254   -->
1255
1256   <rim:ExternalLink id = "acmeLink2" externalURI="http://www.acme2.com">
1257     <rim:Name>
1258       <rim:LocalizedString value = "Link #2"/>
1259     </rim:Name>
1260     <rim:Description>
1261       <rim:LocalizedString value = "ACME's Link #2"/>
1262     </rim:Description>
1263   </rim:ExternalLink>
1264
1265   <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
1266   "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
1267   123456789012"/>
1268
1269   <!--
1270   The following 3 objects externally identify specified ExtrinsicObject using specified
1271   ExternalIdentifier, where the ExternalIdentifier is being submitted and the
1272   ExtrinsicObject is already in registry. Note that the targetObject points to an
1273   ObjectRef in a previous line
1274   -->
1275
1276   <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
1277     <rim:Name>
1278       <rim:LocalizedString value = "DUNS"/>
1279     </rim:Name>
1280
1281     <rim:Description>
1282       <rim:LocalizedString value = "This is the DUNS scheme"/>
1283     </rim:Description>
1284   </rim:ClassificationScheme>
1285
1286   <rim:ExternalIdentifier id = "acmeDUNSID" identificationScheme="DUNS-id" value =
1287   "13456789012">
1288     <rim:Name>
1289       <rim:LocalizedString value = "DUNS" />
1290     </rim:Name>
1291     <rim:Description>
1292       <rim:LocalizedString value = "DUNS ID for ACME"/>
1293     </rim:Description>
1294   </rim:ExternalIdentifier>
1295
1296   <rim:Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc" associationType =
1297   "ExternallyIdentifies" sourceObject = "acmeDUNSID" targetObject = "urn:uuid:a2345678-1234-1234-
1298   123456789012"/>
1299
1300   <!--
1301   The following show submission of a brand new classification scheme in its entirety
1302   -->
1303   <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
1304     <rim:Name>
1305       <rim:LocalizedString value = "Geography"/>
1306     </rim:Name>
1307
1308     <rim:Description>
1309       <rim:LocalizedString value = "This is a sample Geography scheme"/>
1310     </rim:Description>
1311
1312     <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
1313   "NorthAmerica" >

```

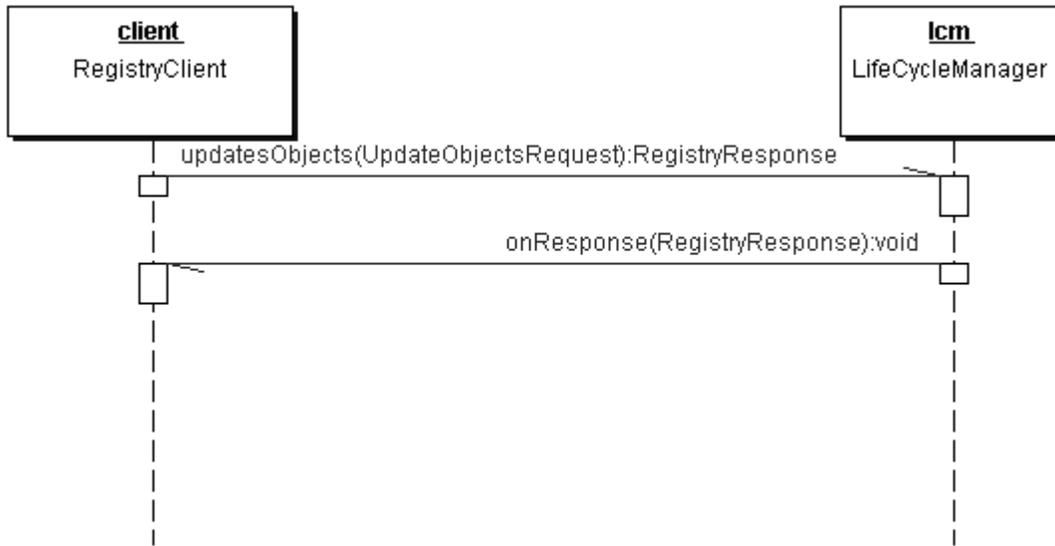
```

1314     <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
1315 "UnitedStates" />
1316     <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
1317     </rim:ClassificationNode>
1318
1319     <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
1320     <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
1321     <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
1322     </rim:ClassificationNode>
1323     </rim:ClassificationNode>
1324 </rim:ClassificationScheme>
1325
1326
1327 <!--
1328 The following show submission of a Automotive sub-tree of ClassificationNodes that
1329 gets added to an existing classification scheme named 'Industry'
1330 that is already in the registry
1331 -->
1332
1333 <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
1334 <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
1335 123456789012">
1336     <rim:Name>
1337     <rim:LocalizedString value = "Automotive" />
1338     </rim:Name>
1339     <rim:Description>
1340     <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
1341     </rim:Description>
1342 </rim:ClassificationNode>
1343
1344 <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
1345     <rim:Name>
1346     <rim:LocalizedString value = "Parts Supplier" />
1347     </rim:Name>
1348     <rim:Description>
1349     <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
1350     </rim:Description>
1351 </rim:ClassificationNode>
1352
1353 <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
1354     <rim:Name>
1355     <rim:LocalizedString value = "Engine Supplier" />
1356     </rim:Name>
1357     <rim:Description>
1358     <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
1359     </rim:Description>
1360 </rim:ClassificationNode>
1361
1362 <!--
1363 The following show submission of 2 Classifications of an object that is already in
1364 the registry using 2 ClassificationNodes. One ClassificationNode
1365 is being submitted in this request (Japan) while the other is already in the registry.
1366 -->
1367
1368 <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
1369 1234-123456789012" classificationNode = "Japan-id">
1370     <rim:Description>
1371     <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
1372     </rim:Description>
1373 </rim:Classification>
1374
1375 <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
1376 "urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
1377 123456789012">
1378     <rim:Description>
1379     <rim:LocalizedString value = "Classifies object using a node in the registry" />
1380     </rim:Description>
1381 </rim:Classification>
1382
1383 <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
1384 </rim:LeafRegistryObjectList>
1385 </SubmitObjectsRequest>
1386

```

1387 **7.4 The Update Objects Protocol**

1388 This section describes the protocol of the Registry Service that allows a Registry Client to update  
 1389 one or more existing RegistryObjects and/or repository items in the registry on behalf of a  
 1390 Submitting Organization. It is expressed in UML notation as described in Appendix C.



1391  
 1392

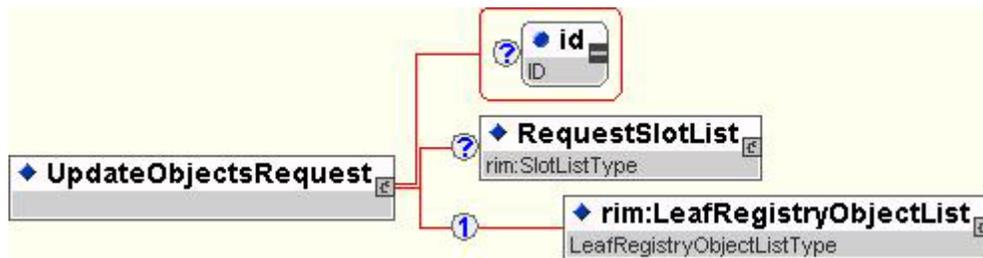
**Figure 17: Update Objects Sequence Diagram**

1393

1394 **7.4.1 UpdateObjectsRequest**

1395 The UpdateObjectsRequest is used by a client to update RegistryObjects and/or repository items  
 1396 that already exist within the registry.

1397 **Syntax:**



1398  
 1399

**Figure 18: UpdateObjectsRequest Syntax**

1400 **Parameters:**

1401

- 1402     ▪ *LeafRegistryObjectsList*: This parameter specifies a collection of RegistryObject  
 1403 instances that are being updated within the registry. All immediate RegistryObject  
 1404 children of the LeafRegistryObjectList must be a current RegistryObject already  
 1405 in the registry. RegistryObjects must include all required attributes, even those the

1406 user does not intend to change. A missing attribute is interpreted as a request to  
1407 set that attribute to NULL or in case it has a default value, the default value will  
1408 be assumed. If this collection contains an immediate child RegistryObject that  
1409 does not already exists in the registry, then the registry must return an  
1410 InvalidRequestException. If the user wishes to submit a mix of new and updated  
1411 objects then she should use a SubmitObjectsRequest.

1412

1413 **Returns:**

1414 This request returns a RegistryResponse. See section 7.3.2 for details.

1415 **Exceptions:**

1416 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1417     ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an  
1418     operation for which she was not authorized.
- 1419     ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object  
1420     within the request that was not found.
- 1421     ▪ *ObjectExistsException*: Indicates that the requestor tried to submit an object using  
1422     an id that matched the id of an existing object in the registry.
- 1423     ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an  
1424     operation which was semantically invalid.
- 1425     ▪ *UnsupportedCapabilityException*: Indicates that the requestor attempted to  
1426     submit some content that is not supported by the registry.
- 1427     ▪ *QuotaExceededException*: Indicates that the requestor attempted to submit more  
1428     content than the quota allowed for them by the registry.

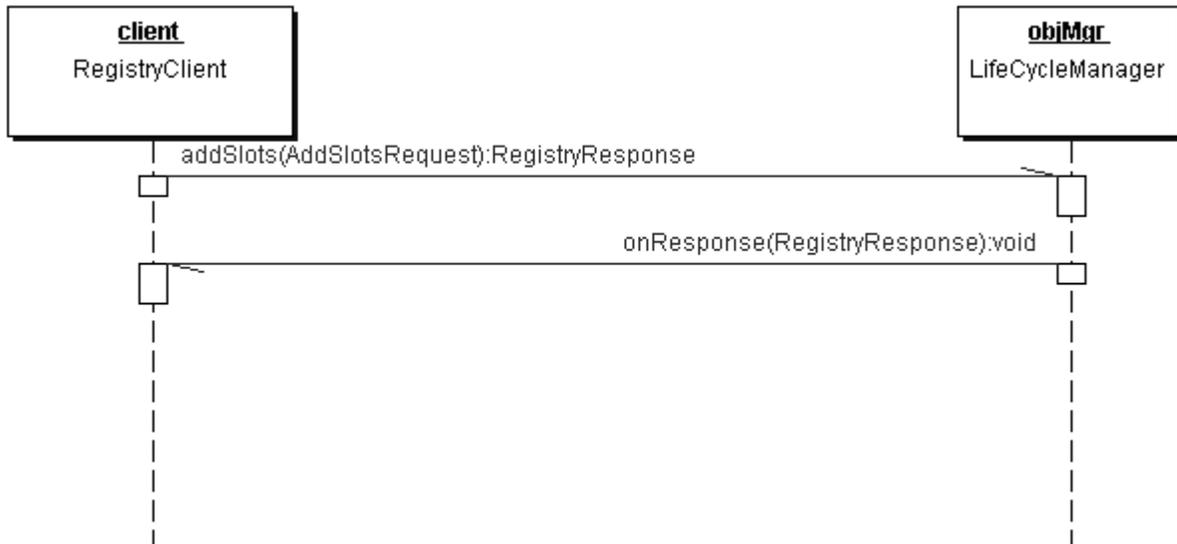
1429

1430 **7.4.2 Audit Trail**

1431 The RS must create AuditableEvents object with eventType *Updated* for each RegistryObject  
1432 updated via an UpdateObjectsRequest.

1433 **7.5 The Add Slots Protocol**

1434 This section describes the protocol of the Registry Service that allows a client to add slots to a  
1435 previously submitted RegistryObject using the LifecycleManager. Slots provide a dynamic  
1436 mechanism for extending RegistryObjects as defined by [ebRIM].



1437  
1438

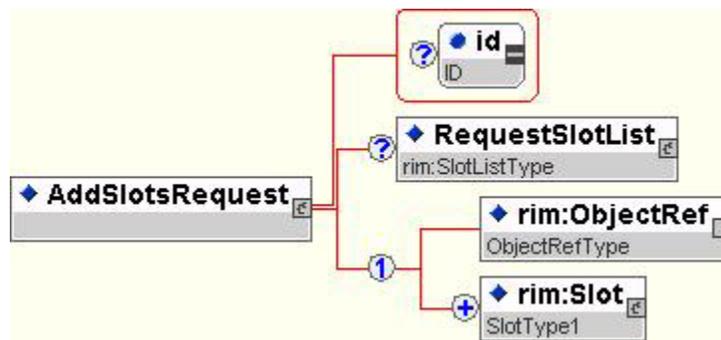
Figure 19: Add Slots Sequence Diagram

1439

1440 **7.5.1 AddSlotsRequest**

1441 The AddSlotsRequest is used by a client to add slots to an existing RegistryObject in the registry.

1442 **Syntax:**



1443  
1444

Figure 20: AddSlotsRequest Syntax

1445 **Parameters:**

- 1446     ▪ *ObjectRef*: This parameter specifies a reference to a RegistryObject instance to
- 1447         which the requestor wishes to add slots via this request.
- 1448     ▪ *Slot*: This parameter specifies one or more Slot objects. Each Slot contains a
- 1449         ValueList with one or more Values. Each Slot also has a slot name and a slotType
- 1450         as described [ebRIM].

1451

1452 **Returns:**

1453 This request returns a RegistryResponse. See section 7.3.2 for details.

1454 **Exceptions:**

1455 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1456       ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an  
1457       operation for which she was not authorized.
- 1458       ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object  
1459       within the request that was not found.
- 1460       ▪ *SlotExistsException*: Indicates that the requestor tried to add a Slot using a name  
1461       that matched the name of an existing Slot in the RegistryObject.
- 1462       ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an  
1463       operation which was semantically invalid.

1464

1465 **7.6 The Remove Slots Protocol**

1466 This section describes the protocol of the Registry Service that allows a client to remove slots to  
1467 a previously submitted RegistryObject using the LifeCycleManager.

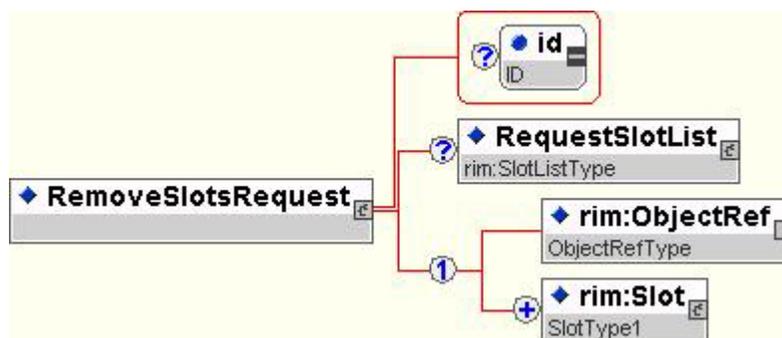


1468  
1469

**Figure 21: Remove Slots Sequence Diagram**

1470 **7.6.1 RemoveSlotsRequest**

1471 The RemoveSlotsRequest is used by a client to remove slots from an existing RegistryObject in  
1472 the registry.

1473 **Syntax:**1474  
1475 **Figure 22: RemoveSlotsRequest Syntax**1476 **Parameters:**

- 1477
- 1478
- 1479
- 1480
- 1481
- *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.

1482

1483 **Returns:**

1484 This request returns a RegistryResponse. See section 7.3.2 for details.

1485 **Exceptions:**

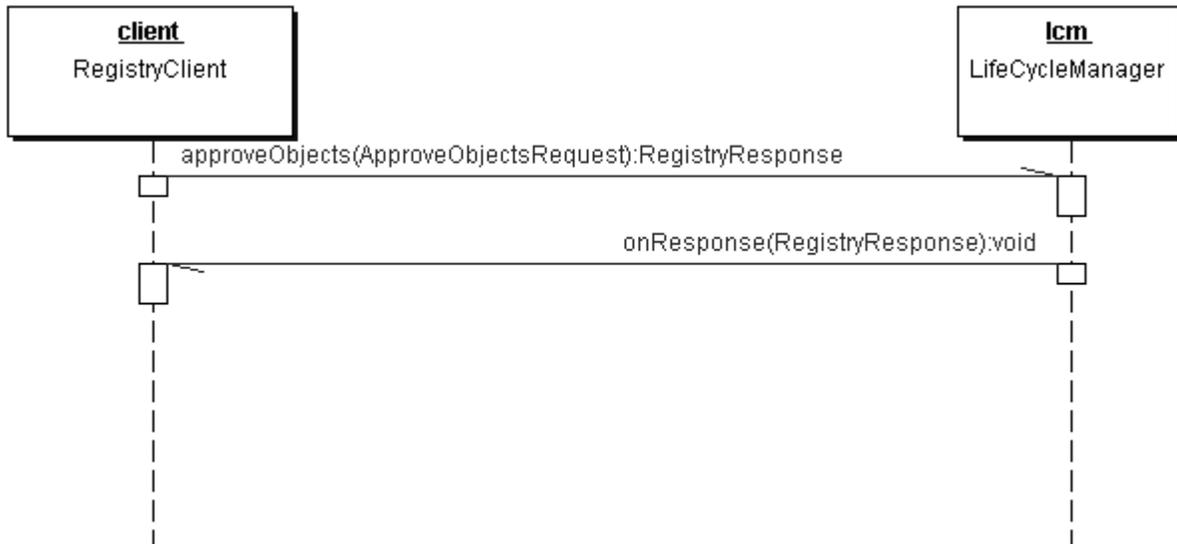
1486 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1487
- 1488
- 1489
- 1490
- 1491
- 1492
- 1493
- 1494
- 1495
- *AuthorizationException*: Indicates that the requestor attempted to perform an operation for which she was not authorized.
  - *ObjectNotFound*: Indicates that the requestor referenced an object within the request that was not found.
  - *SlotNotFound*: Indicates that the requestor attempted to remove a Slot by name where no Slot existed that matches the specified name.
  - *InvalidRequest*: Indicates that the requestor attempted to perform an operation which was semantically invalid.

1496 **7.7 The Approve Objects Protocol**

1497 This section describes the protocol of the Registry Service that allows a client to approve one or

1498 more previously submitted RegistryObject objects using the LifeCycleManager.



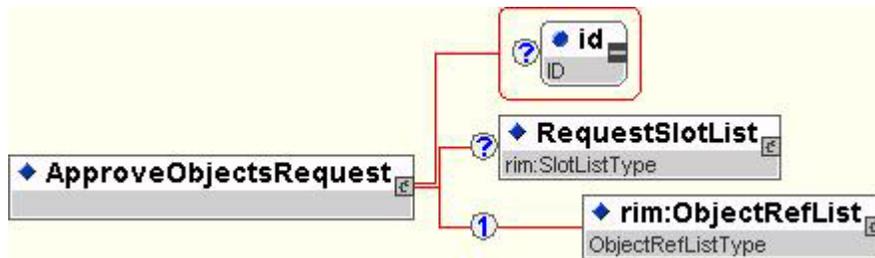
1499  
1500

Figure 23: Approve Objects Sequence Diagram

1501 **7.7.1 ApproveObjectsRequest**

1502 The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject  
1503 instances in the registry.

1504 **Syntax:**



1505  
1506

Figure 24: ApproveObjectsRequest Syntax

1507 **Parameters:**

- 1508     ▪ *ObjectRefList*: This parameter specifies a collection of reference to existing  
1509     RegistryObject instances in the registry. These are the objects that the requestor  
1510     wishes to approve via this request.

1511

1512 **Returns:**

1513 This request returns a RegistryResponse. See section 7.3.2 for details.

1514 **Exceptions:**

1515 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1516     ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an

- 1517 operation for which she was not authorized.
- 1518       ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object
- 1519       within the request that was not found.
- 1520       ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an
- 1521       operation that was semantically invalid.
- 1522

## 1523 7.7.2 Audit Trail

1524 The RS must create AuditableEvent objects with eventType *Approved* for each RegistryObject

1525 instance approved via an ApproveObjectsRequest.

## 1526 7.8 The Deprecate Objects Protocol

1527 This section describes the protocol of the Registry Service that allows a client to deprecate one or

1528 more previously submitted RegistryObject instances using the LifeCycleManager. Once a

1529 RegistryObject is deprecated, no new references (e.g. new Associations, Classifications and

1530 ExternalLinks) to that object can be submitted. However, existing references to a deprecated

1531 object continue to function normally.



1532

1533

Figure 25: Deprecate Objects Sequence Diagram

### 1534 7.8.1 DeprecateObjectsRequest

1535 The DeprecateObjectsRequest is used by a client to deprecate one or more existing

1536 RegistryObject instances in the registry.

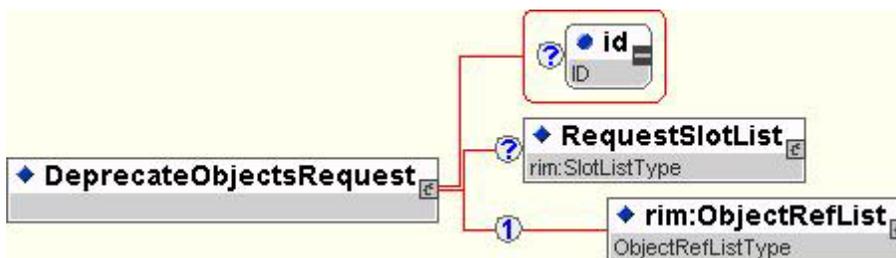
1537 **Syntax:**1538  
1539

Figure 26: DeprecateObjectsRequest Syntax

1540 **Parameters:**

- 1541       ▪ *ObjectRefList*: This parameter specifies a collection of reference to existing  
1542 RegistryObject instances in the registry. These are the objects that the requestor  
1543 wishes to deprecate via this request.

1544

1545 **Returns:**

1546 This request returns a RegistryResponse. See section 7.3.2 for details.

1547 **Exceptions:**

1548 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1549       ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an  
1550 operation for which she was not authorized.
- 1551       ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object  
1552 within the request that was not found.
- 1553       ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an  
1554 operation which was semantically invalid.

1555 **7.8.2 Audit Trail**

1556 The RS must create AuditableEvents object with eventType *Deprecated* for each RegistryObject  
1557 deprecated via a DeprecateObjectsRequest.

1558 **7.9 The Undeprecate Objects Protocol**

1559 This section describes the protocol of the Registry Service that allows a client to undo the  
1560 deprecation of one or more previously deprecate RegistryObject instances using the  
1561 LifeCycleManager. When a RegistryObject is un-deprecated, it goes back to the Submitted status  
1562 and new references (e.g. new Associations, Classifications and ExternalLinks) to that object can  
1563 be submitted.



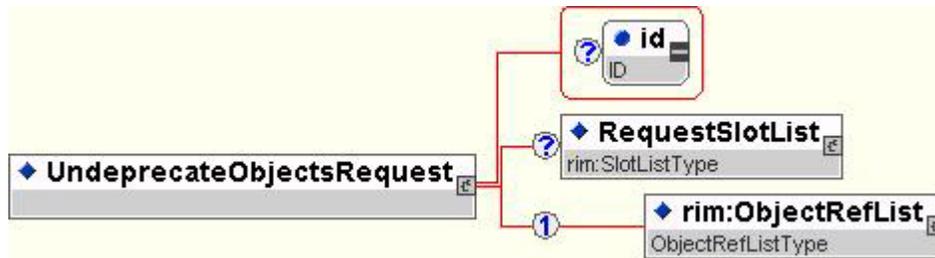
1564  
1565

Figure 27: Undeprecate Objects Sequence Diagram

1566 **7.9.1 UndeprecateObjectsRequest**

1567 The UndeprecateObjectsRequest is used by a client to un-deprecate one or more existing  
1568 RegistryObject instances in the registry.

1569 **Syntax:**



1570  
1571

Figure 28: UndeprecateObjectsRequest Syntax

1572 **Parameters:**

- 1573     ▪ *ObjectRefList*: This parameter specifies a collection of reference to existing  
1574     RegistryObject instances in the registry. These are the objects that the requestor  
1575     wishes to un-deprecate via this request. The registry should silently ignore any  
1576     reference to a RegistryObject that is not deprecated.

1577

1578 **Returns:**

1579 This request returns a RegistryResponse. See section 7.3.2 for details.

1580 **Exceptions:**

1581 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1582     ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an

- 1583 operation for which she was not authorized.
- 1584     ▪ *ObjectNotFoundException*: Indicates that the requestor referenced an object
- 1585     within the request that was not found.
- 1586     ▪ *InvalidRequestException*: Indicates that the requestor attempted to perform an
- 1587     operation which was semantically invalid.
- 1588

## 1589 7.9.2 Audit Trail

1590 The RS must create AuditableEvents object with eventType *Undeprecated* for each

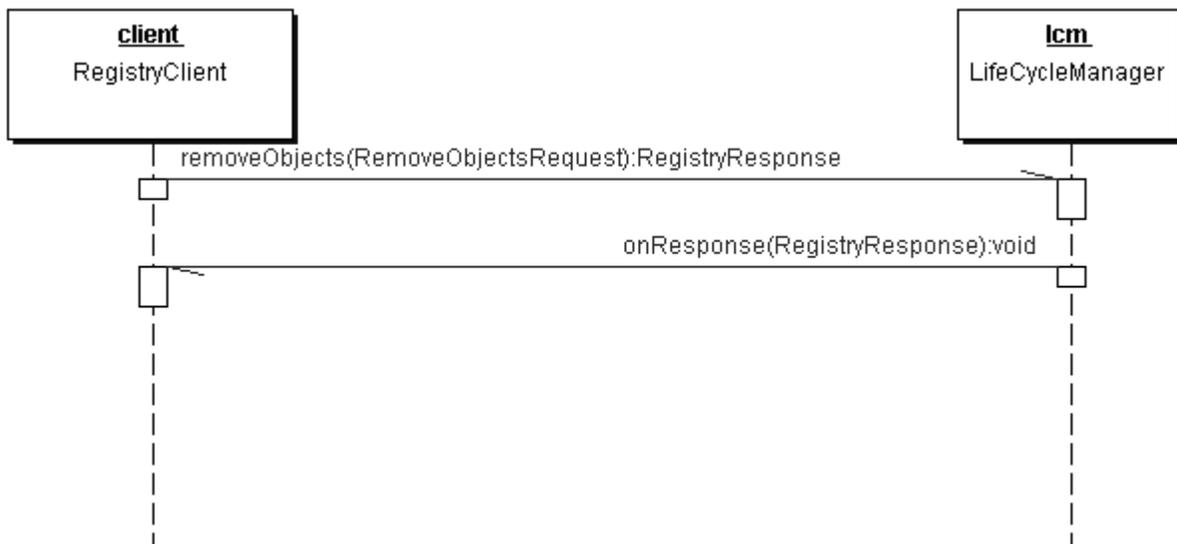
1591 RegistryObject undeprecated via an UndeprecateObjectsRequest.

## 1592 7.10 The Remove Objects Protocol

1593 This section describes the protocol of the Registry Service that allows a client to remove one or

1594 more RegistryObject instances and/or repository items using the LifeCycleManager.

1595 The remove object protocol is expressed in UML notation as described in Appendix C.



1596

1597

Figure 29: Remove Objects Sequence Diagram

1598 For details on the schema for the business documents shown in this process refer to Appendix B.

### 1599 7.10.1 RemoveObjectsRequest

1600 The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject

1601 and/or repository items from the registry.



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

## 1638 8 Query Management Service

1639 This section describes the capabilities of the Registry Service that allow a client  
 1640 (QueryManagerClient) to search for or query different kinds of registry objects in the ebXML  
 1641 Registry using the QueryManager interface of the Registry. The Registry supports the following  
 1642 query capabilities:

- 1643 • Filter Query
- 1644 • SQL Query

1645 The Filter Query mechanism in Section 8.2 *must* be supported by every Registry implementation.  
 1646 The SQL Query mechanism is an optional feature and *may* be provided by a registry  
 1647 implementation. However, if a vendor provides an SQL query capability to an ebXML Registry  
 1648 it *must* conform to this document. As such this capability is a normative yet optional capability.

1649 In a future version of this specification, the W3C XQuery syntax may be considered as another  
 1650 query syntax.

### 1651 8.1 Ad Hoc Query Request/Response

1652 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The  
 1653 AdhocQueryRequest contains a sub-element that defines a query in one of the supported  
 1654 Registry query mechanisms.

1655 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously  
 1656 back to the client. The AdhocQueryResponse returns a collection of objects whose element type  
 1657 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects  
 1658 representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate  
 1659 classes in [ebRIM] such as RegistryObject and RegistryEntry.

1660 Any errors in the query request messages are indicated in the corresponding  
 1661 AdhocQueryResponse message.



1662

1663

**Figure 31: Submit Ad Hoc Query Sequence Diagram**

1664 For details on the schema for the business documents shown in this process refer to Appendix

1665 **Error! Reference source not found..**

### 1666 8.1.1 AdhocQueryRequest

1667 The AdhocQueryRequest is used to submit queries to the registry.

1668 **Syntax:**

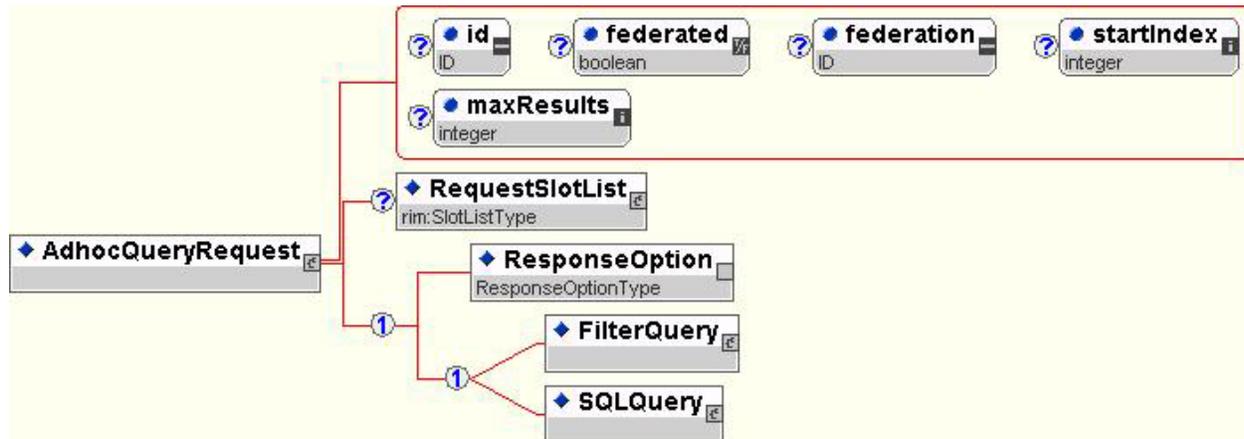


Figure 32: AdhocQueryRequest Syntax

1669  
1670

1671 **Parameters:**

- 1672     ▪ *federated*: This optional parameter specifies that the registry must process this  
1673 query as a federated query. By default its value is *false*.
- 1674     ▪ *federation*: This optional parameter specifies the id of the target Federation for a  
1675 federated query in case the registry is a member of multiple federations. In the  
1676 absence of this parameter a registry must route the federated query to all  
1677 federations that it is a member of.
- 1678     ▪ *FilterQuery*: This parameter specifies a registry Filter Query.
- 1679     ▪ *maxResults*: This optional parameter specifies a limit on the maximum number of  
1680 results the client wishes the query to return. If unspecified, the registry should  
1681 return either all the results, or in case the result set size exceeds a registry specific  
1682 limit, the registry should return a sub-set of results that are within the bounds of  
1683 the registry specific limit. See section 0 for an illustrative example.
- 1684     ▪ *ResponseOption*: This required parameter allows the client to control the format  
1685 and content of the AdhocQueryResponse to this request. See section 8.1.3 for  
1686 details.
- 1687     ▪ *SQLQuery*: This parameter specifies a registry SQL Query.
- 1688     ▪ *startIndex*: This optional integer value is used to indicate which result *must* be  
1689 returned as the first result when iterating over a large result set. The default  
1690 value is 0, which returns the result set starting with index 0 (first result). See  
1691 section 0 for an illustrative example.

1692

1693 **Returns:**

1694 This request returns an AdhocQueryResponse. See section 8.1.2 for details.

1695 **Exceptions:**

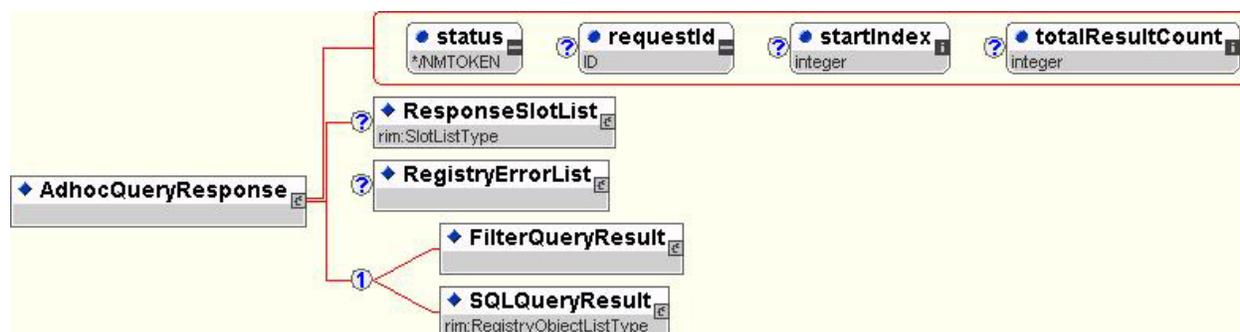
1696 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 1697     ▪ *AuthorizationException*: Indicates that the requestor attempted to perform an
- 1698         operation for which she was not authorized.
- 1699     ▪ *InvalidQueryException*: signifies that the query syntax or semantics was invalid.
- 1700         Client must fix the query syntax or semantic and re-submit the query.

1701 **8.1.2 AdhocQueryResponse**

1702 The AdhocQueryResponse is sent by the registry as a response to AdhocQueryRequest.

1703

1704 **Syntax:**

1705

1706

Figure 33: AdhocQueryResponse Syntax

1707 **Parameters:**

- 1708     ▪ *FilterQueryResult*: This parameter specifies the result of a registry Filter Query.
- 1709     ▪ *SQLQueryResult*: This parameter specifies the result of a registry SQL Query.
- 1710     ▪ *startIndex*: This optional integer value is used to indicate the index for the first
- 1711         result in the result set returned by the query, within the complete result set
- 1712         matching the query. By default, this value is 0. See section 0 for an illustrative
- 1713         example.
- 1714     ▪ *totalResultCount*: This optional parameter specifies the size of the complete
- 1715         result set matching the query within the registry. When this value is unspecified,
- 1716         the client should assume that value is the size of the result set contained within the
- 1717         result. See section 0 for an illustrative example.

1718

1719 **8.1.3 ResponseOption**

1720 A client specifies an ResponseOption structure within an AdhocQueryRequest to indicate the

1721 format of the results within the corresponding AdhocQueryResponse.

1722

1723 **Syntax:**



1724

1725

Figure 34: ResponseOption Syntax

1726

1727 **Parameters:**

1728     ▪ *returnComposedObjects*: This optional parameter specifies whether the  
1729 RegistryObjects returned should include composed objects as defined by Figure 1  
1730 in [ebRIM]. The default is to return all composed objects.

1731     ▪ *returnType*: This optional enumeration parameter specifies the type of  
1732 RegistryObject to return within the response. Values for returnType are as  
1733 follows:

1734

1735     • ObjectRef - This option specifies that the AdhocQueryResponse may contain  
1736 a collection of ObjectRef XML elements as defined in [ebRIM Schema].  
1737 Purpose of this option is to return references to registry objects.

1738     • RegistryObject - This option specifies that the AdhocQueryResponse may  
1739 contain a collection of RegistryObject XML elements as defined in [ebRIM  
1740 Schema]. In this case all attributes of the registry objects are returned  
1741 (objectType, name, description, ...).

1742     • RegistryEntry - This option specifies that the AdhocQueryResponse may  
1743 contain a collection of RegistryEntry or RegistryObject XML elements as  
1744 defined in [ebRIM Schema], which correspond to RegistryEntry or  
1745 RegistryObject attributes.

1746     • LeafClass - This option specifies that the AdhocQueryResponse may contain a  
1747 collection of XML elements that correspond to leaf classes as defined in  
1748 [ebRIM Schema].

1749     • LeafClassWithRepositoryItem - This option specifies that the  
1750 AdhocQueryResponse may contain a collection of ExtrinsicObject XML  
1751 elements as defined in [ebRIM Schema] accompanied with their repository  
1752 items or RegistryEntry or RegistryObject and their attributes. Linking of  
1753 ExtrinsicObject and its repository item is accomplished using the technique  
1754 explained in Section 8.4 -Content Retrieval.

1755 If “returnType” specified does not match a result returned by the query, then the  
1756 registry *must* use the closest matching semantically valid returnType that matches  
1757 the result.

1758 This can be illustrated with a case when OrganizationQuery is asked to return  
1759 LeafClassWithRepositoryItem. As this is not possible, QueryManager will  
1760 assume LeafClass option instead. If OrganizationQuery is asked to retrieve a

1761 RegistryEntry as a return type then RegistryObject metadata will be returned.

#### 1762 **8.1.4 Iterative Query Support**

1763 The iterative query feature is a normative optional feature of the registry. The  
 1764 AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large result  
 1765 set matching a logical query by allowing multiple AdhocQueryRequest requests to be submitted  
 1766 such that each query requests a different sliding window within the result set. This feature  
 1767 enables the registry to handle queries that match a very large result set, in a scalable manner.

1768 The iterative queries feature is not a true Cursor capability as found in databases. The registry is  
 1769 not required to maintain transactional consistency or state between iterations of a query. Thus it  
 1770 is possible for new objects to be added or existing objects to be removed from the complete  
 1771 result set in between iterations. As a consequence it is possible to have a result set element be  
 1772 skipped or duplicated between iterations.

1773 Note that while it is not required, it may be possible for implementations to be smart and  
 1774 implement a transactionally consistent iterative query feature. It is likely that a future version of  
 1775 this specification will require a transactionally consistent iterative query capability.

#### 1776 **Query Iteration Example**

1777 Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a  
 1778 query that matches all 1007 Organizations. The user wishes to do the query iteratively such that  
 1779 Organizations are retrieved in chunks of 100. The following table illustrates the parameters of  
 1780 the AdhocQueryRequest and those of the AdhocQueryResponses for each iterative query in this  
 1781 example.

1782

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

1783

#### 1784 **8.2 Filter Query Support**

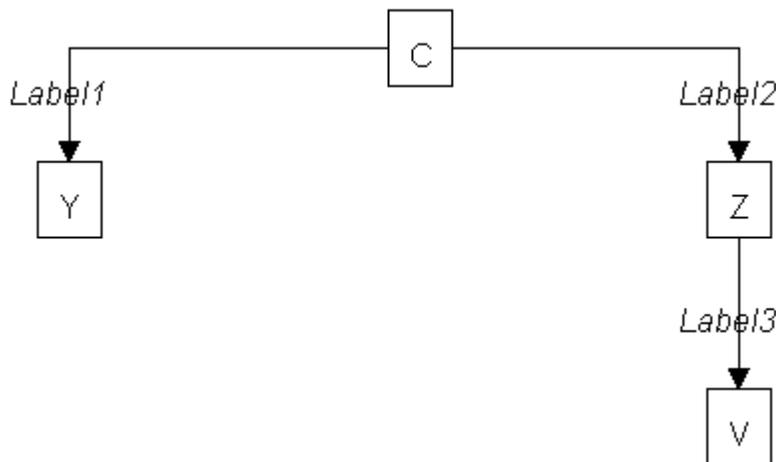
1785 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML  
 1786 conforming Registry implementation. Each query alternative is directed against a single class

1787 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter  
1788 queries depending on which classes are queried on.

- 1789 • Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic  
1790 queries that might return different subclasses of the class that is queried on. The result of  
1791 such a query is a set of XML elements that correspond to instances of any class that satisfies  
1792 the responseOption defined previously in Section 8.1.3. An example might be that  
1793 RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances  
1794 that satisfy the query. This implies that response might return XML elements that correspond  
1795 to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- 1796 • Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact  
1797 traversals of these classes. Responses to these queries are accordingly constrained.

1798 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an  
1799 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified  
1800 herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified  
1801 in Section 8.1.

1802 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of  
1803 classes derived from a single class and its associations with other classes as defined by ebRIM.  
1804 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For  
1805 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a  
1806 class that is associated with Z. The ebRIM Binding for C might be as in Figure 35



1807  
1808 **Figure 35: Example ebRIM Binding**

1809 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and  
1810 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to  
1811 which ebRIM association is intended. The name of the query is determined by the root class, i.e.  
1812 this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances  
1813 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are  
1814 limited to instances that are linked to their parent node by the identified association.

1815 Each FilterQuery alternative depends upon one or more class filters, where a class filter is a  
1816 restricted predicate clause over the attributes of a single class. Class methods that are defined in  
1817 ebRIM and that return simple types constitute “visible attributes” that are valid choices for  
1818 predicate clauses. Names of those attributes will be same as name of the corresponding method  
1819 just without the prefix ‘get’. For example, in case of “getLevelNumber” method the

1820 corresponding visible attribute is “levelNumber”. The supported class filters are specified in  
 1821 Section 8.2.17 and the supported predicate clauses are defined in Section 8.2.18. A FilterQuery  
 1822 will be composed of elements that traverse the tree to determine which branches satisfy the  
 1823 designated class filters, and the query result will be the set of instances that support such a  
 1824 branch.

1825 In the above example, the CQuery element will have three subelements, one a CFilter on the C  
 1826 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on  
 1827 the Y class to eliminate branches from C to Y where the target of the association does not satisfy  
 1828 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third  
 1829 element is called a branch element because it allows class filters on each class along the path  
 1830 from C to V. In general, a branch element will have subelements that are themselves class filters,  
 1831 other branch elements, or a full-blown query on the class in the path.

1832 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one  
 1833 branch, filter or query element on Y is allowed. However, if the association is one-to-many, then  
 1834 multiple branch, filter or query elements are allowed. This allows one to specify that an instance  
 1835 of C must have associations with multiple instances of Y before the instance of C is said to  
 1836 satisfy the branch element.

1837 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be  
 1838 stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then  
 1839 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax  
 1840 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their  
 1841 respective superclass queries. Structures of XML elements that match the ebRIM classes are  
 1842 explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in  
 1843 ebRIM whenever possible.

1844 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual  
 1845 hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify  
 1846 the effect of that binding on query semantics.

## 1847 8.2.1 FilterQuery

### 1848 Purpose

1849 To identify a set of queries that traverse specific registry class. Each alternative assumes a  
 1850 specific binding to ebRIM. The status is a success indication or a collection of warnings and/or  
 1851 exceptions.

### 1852 Definition

```

1853 <complexType name="FilterQueryType">
1854   <complexContent>
1855     <extension base="rim:AdhocQueryType">
1856       <choice minOccurs="1" maxOccurs="1">
1857         <element ref="tns:RegistryObjectQuery"/>
1858         <element ref="tns:RegistryEntryQuery"/>
1859         <element ref="tns:AssociationQuery"/>
1860         <element ref="tns:AuditableEventQuery"/>
1861         <element ref="tns:ClassificationQuery"/>
1862         <element ref="tns:ClassificationNodeQuery"/>
1863         <element ref="tns:ClassificationSchemeQuery"/>
1864       </choice>
     </extension>
   </complexContent>
 </complexType>

```

```

1865     <element ref="tns:RegistryPackageQuery"/>
1866     <element ref="tns:ExtrinsicObjectQuery"/>
1867     <element ref="tns:OrganizationQuery"/>
1868     <element ref="tns:ServiceQuery"/>
1869     <element ref="tns:FederationQuery"/>
1870     <element ref="tns:RegistryQuery"/>
1871     <element ref="tns:SubscriptionQuery"/>
1872     <element ref="tns:UserQuery"/>
1873   </choice>
1874 </extension>
1875 </complexContent>
1876 </complexType>
1877 <element name="FilterQuery" type="tns:FilterQueryType"/>
1878
1879 <element name="FilterQueryResult">
1880   <complexType>
1881     <choice minOccurs="1" maxOccurs="1">
1882       <element ref="tns:RegistryObjectQueryResult" />
1883       <element ref="tns:RegistryEntryQueryResult" />
1884       <element ref="tns:AssociationQueryResult" />
1885       <element ref="tns:AuditableEventQueryResult" />
1886       <element ref="tns:ClassificationQueryResult" />
1887       <element ref="tns:ClassificationNodeQueryResult" />
1888       <element ref="tns:ClassificationSchemeQueryResult" />
1889       <element ref="tns:RegistryPackageQueryResult" />
1890       <element ref="tns:ExtrinsicObjectQueryResult" />
1891       <element ref="tns:OrganizationQueryResult" />
1892       <element ref="tns:ServiceQueryResult" />
1893       <element ref="tns:FederationQueryResult" />
1894       <element ref="tns:RegistryQueryResult" />
1895       <element ref="tns:SubscriptionQueryResult" />
1896       <element ref="tns:UserQueryResult" />
1897     </choice>
1898   </complexType>
1899 </element>
1900

```

## 1901 Semantic Rules

- 1902 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 1903 2. Semantic rules specify the procedure for implementing the evaluation of Filter Queries.
- 1904 Implementations do not necessarily have to follow the same procedure provided that the
- 1905 same effect is achieved.
- 1906 3. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
- 1907 Each XML attribute carries a value derived from the value of an attribute specified in the
- 1908 Registry Information Model [ebRIM Schema].
- 1909 4. For each FilterQuery subelement there is only one corresponding FilterQueryResult
- 1910 subelement that must be returned as a response. Class name of the FilterQueryResult
- 1911 subelement has to match the class name of the FilterQuery subelement.
- 1912 5. If a Branch or Query element for a class has no sub-elements then every persistent instance
- 1913 of that class satisfies the Branch or Query.

- 1914 6. If an error condition is raised during any part of the execution of a FilterQuery, then the  
 1915 status attribute of the XML RegistryResult is set to “failure” and no AdHocQueryResult  
 1916 element is returned; instead, a RegistryErrorList element must be returned with its  
 1917 highestSeverity element set to “error”. At least one of the RegistryError elements in the  
 1918 RegistryErrorList will have its severity attribute set to “error”.
- 1919 7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of  
 1920 the XML RegistryResult is set to “success” and an appropriate FilterQueryResult element  
 1921 must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of  
 1922 the RegistryErrorList is set to “warning” and the severity attribute of each RegistryError is  
 1923 set to “warning”.

1924 **8.2.2 RegistryObjectQuery**

1925 **Purpose**

1926 To identify a set of registry object instances as the result of a query over selected registry  
 1927 metadata.

1928 **ebRIM Binding**

1929

1930

1931

1932

1933

1934

1935

1936

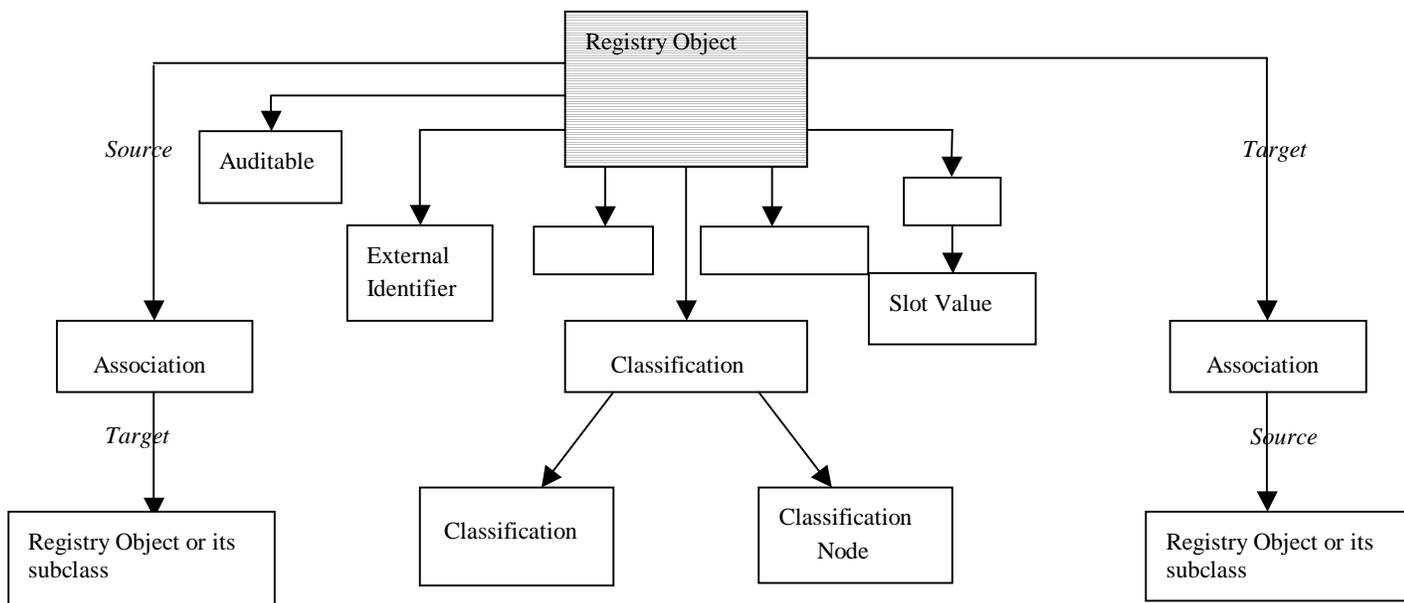
1937

1938

1939

1940

1941



1942 **Figure 37: ebRIM Binding for RegistryObjectQuery**

1943 **Definition**

```

1944 <complexType name="RegistryObjectQueryType">
1945   <sequence>
1946     <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
1947     <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
1948     <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
1949     <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
1950     <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
1951     <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
1952     <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
    
```

```

1953   <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1954   <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1955 </sequence>
1956 </complexType>
1957 <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
1958
1959 <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1960
1961 <complexType name="InternationalStringBranchType">
1962   <sequence>
1963     <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1964   </sequence>
1965 </complexType>
1966
1967 <complexType name="AssociationBranchType">
1968   <sequence>
1969     <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1970     <choice minOccurs="0" maxOccurs="1">
1971       <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1972       <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1973       <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1974       <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1975       <element ref="tns:AssociationQuery" minOccurs="0" maxOccurs="1" />
1976       <element ref="tns:ClassificationQuery" minOccurs="0" maxOccurs="1" />
1977       <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1978       <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1979       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1980       <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1981       <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1982       <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1983       <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1984       <element ref="tns:FederationQuery" minOccurs="0" maxOccurs="1" />
1985       <element ref="tns:RegistryQuery" minOccurs="0" maxOccurs="1" />
1986       <element ref="tns:SubscriptionQuery" minOccurs="0" maxOccurs="1" />
1987       <element ref="tns:UserQuery" minOccurs="0" maxOccurs="1" />
1988       <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1989       <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1990     </choice>
1991   </sequence>
1992 </complexType>
1993 <element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
1994 <element name="TargetAssociationBranch" type="tns:AssociationBranchType" />
1995
1996 <element name="ClassifiedByBranch">
1997   <complexType>
1998     <sequence>
1999       <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
2000       <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
2001       <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
2002     </sequence>
2003   </complexType>
2004 </element>
2005
2006 <element name="SlotBranch">
2007   <complexType>
2008     <sequence>
2009       <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />

```

```

2010     <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
2011     </sequence>
2012 </complexType>
2013 </element>
2014
2015 <complexType name="ServiceBindingBranchType">
2016     <sequence>
2017         <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
2018         <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
2019         <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
2020     </sequence>
2021 </complexType>
2022 <element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType" />
2023 <element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType" />
2024
2025 <element name="SpecificationLinkBranch">
2026     <complexType>
2027         <sequence>
2028             <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
2029             <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2030             <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2031         </sequence>
2032     </complexType>
2033 </element>
2034

```

### 2035 Semantic Rules

- 2036 1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The  
 2037 following steps will eliminate instances in RO that do not satisfy the conditions of the  
 2038 specified filters.
- 2039 a) If RO is empty then go to number 2 below.
- 2040 b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a  
 2041 registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from  
 2042 RO. If RO is empty then continue to the next numbered rule.
- 2043 c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise,  
 2044 let x be a remaining registry object in RO. If x is not linked to at least one  
 2045 ExternalIdentifier instance, then remove x from RO; otherwise, treat each  
 2046 ExternalIdentifierFilter element separately as follows: Let EI be the set of  
 2047 ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If  
 2048 EI is empty, then remove x from RO. If RO is empty then continue to the next numbered  
 2049 rule.
- 2050 d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a  
 2051 remaining registry object in RO. If x doesn't have an auditable event that satisfy  
 2052 AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is  
 2053 empty then continue to the next numbered rule.

- 2054 e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining  
2055 registry object in RO. If x does not have a name then remove x from RO. If RO is empty  
2056 then continue to the next numbered rule; otherwise treat NameBranch as follows: If any  
2057 LocalizedStringFilter that is specified is not satisfied by all of the LocalizedStrings that  
2058 constitute the name of the registry object then remove x from RO. If RO is empty then  
2059 continue to the next numbered rule.
- 2060 f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a  
2061 remaining registry object in RO. If x does not have a description then remove x from RO.  
2062 If RO is empty then continue to the next numbered rule; otherwise treat  
2063 DescriptionBranch as follows: If any LocalizedStringFilter that is specified is not  
2064 satisfied by all of the LocalizedStrings that constitute the description of the registry  
2065 object then remove x from RO. If RO is empty then continue to the next numbered rule.
- 2066 g) If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x  
2067 be a remaining registry object in RO. If x is not the classifiedObject of at least one  
2068 Classification instance, then remove x from RO; otherwise, treat each  
2069 ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified  
2070 within the ClassifiedByBranch, then let CL be the set of all Classification instances that  
2071 have x as the classifiedObject; otherwise, let CL be the set of Classification instances that  
2072 satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then  
2073 remove x from RO and continue to the next numbered rule. Otherwise, if CL is not  
2074 empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of  
2075 remaining Classification instances in CL whose defining classification scheme satisfies  
2076 the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and  
2077 continue to the next numbered rule. Otherwise, if CL remains not empty, and if a  
2078 ClassificationNodeQuery is specified, then replace CL by the set of remaining  
2079 Classification instances in CL for which a classification node exists and for which that  
2080 classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then  
2081 remove x from RO. If RO is empty then continue to the next numbered rule.
- 2082 h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a  
2083 remaining registry object in RO. If x is not linked to at least one Slot instance, then  
2084 remove x from RO. If RO is empty then continue to the next numbered rule; otherwise,  
2085 treat each SlotBranch element separately as follows: If a SlotFilter is not specified within  
2086 the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the  
2087 set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,  
2088 then remove x from RO and continue to the next numbered rule. Otherwise, if SL  
2089 remains not empty, and if a SlotValueFilter is specified, replace SL by the set of  
2090 remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is  
2091 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
- 2092 i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,  
2093 let x be a remaining registry object in RO. If x is not the source object of at least one  
2094 Association instance, then remove x from RO. If RO is empty then continue to the next  
2095 numbered rule; otherwise, treat each SourceAssociationBranch element separately as  
2096 follows:

2097 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be  
2098 the set of all Association instances that have x as a source object; otherwise, let AF be the  
2099 set of Association instances that satisfy the AssociationFilter and have x as the source  
2100 object. If AF is empty, then remove x from RO.

2101

2102 If RO is empty then continue to the next numbered rule.

2103

2104 If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT  
2105 be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target  
2106 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty  
2107 then continue to the next numbered rule.

2108

2109 If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let  
2110 ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and  
2111 are the target object of some element of AF. If ROT is empty, then remove x from RO. If  
2112 RO is empty then continue to the next numbered rule.

2113

2114 If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT  
2115 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the  
2116 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
2117 empty then continue to the next numbered rule.

2118

2119 If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT  
2120 be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the  
2121 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
2122 empty then continue to the next numbered rule.

2123

2124 If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let  
2125 ROT be the set of ClassificationScheme instances that satisfy the  
2126 ClassificationSchemeQuery and are the target object of some element of AF. If ROT is  
2127 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

2128

2129 If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let  
2130 ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery  
2131 and are the target object of some element of AF. If ROT is empty, then remove x from  
2132 RO. If RO is empty then continue to the next numbered rule.

2133

2134 If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT  
2135 be the set of Organization instances that satisfy the OrganizationQuery and are the target  
2136 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty  
2137 then continue to the next numbered rule.

2138

2139 If an AuditableEventQuery is specified within the SourceAssociationBranch, then let  
2140 ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are  
2141 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO  
2142 is empty then continue to the next numbered rule.

2143

2144 If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let  
2145 ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and  
2146 are the target object of some element of AF. If ROT is empty, then remove x from RO. If  
2147 RO is empty then continue to the next numbered rule.

2148

2149 If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let  
2150 ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are  
2151 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO  
2152 is empty then continue to the next numbered rule.

2153

2154 If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the  
2155 set of Service instances that satisfy the ServiceQuery and are the target object of some  
2156 element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue  
2157 to the next numbered rule.

2158

2159 If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT  
2160 be the set of Classification instances that satisfy the ClassificationQuery and are the  
2161 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
2162 empty then continue to the next numbered rule (Rule 2).

2163

2164 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let  
2165 ROT be the set of ServiceBinding instances that are the target object of some element of  
2166 AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next  
2167 numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is  
2168 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then  
2169 remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then  
2170 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the  
2171 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as  
2172 follows:

2173

2174 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link  
2175 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
2176 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
2177 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then  
2178 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
2179 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl  
2180 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:  
2181 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is  
2182 not a specification link for at least one registry object in RO, then remove sl from SL. If  
2183 SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO  
2184 is empty then continue to the next numbered rule. If a RegistryEntryQuery element is

2185 specified within the SpecificationLinkBranch then let sl be a remaining specification link  
2186 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the  
2187 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least  
2188 one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from  
2189 ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next  
2190 numbered rule. If a ServiceBindingTargetBranch is specified within the  
2191 ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy  
2192 the ServiceBindingTargetBranch and are the target service binding of some element of  
2193 ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from  
2194 RO. If RO is empty then continue to the next numbered rule.

2195  
2196 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let  
2197 ROT be the set of SpecificationLink instances that are the target object of some element  
2198 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the  
2199 next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is  
2200 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
2201 remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then  
2202 continue to the next numbered rule. If a RegistryObjectQuery element is specified within  
2203 the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat  
2204 RegistryObjectQuery element as follows: Let RO be the result set of the  
2205 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some  
2206 registry object in RO, then remove sl from ROT. If ROT is empty then remove x from  
2207 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery  
2208 element is specified within the SpecificationLinkBranch then let sl be a remaining  
2209 specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the  
2210 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification  
2211 link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then  
2212 remove x from RO. If RO is empty then continue to the next numbered rule.

2213  
2214 If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be  
2215 the set of Association instances that satisfy the AssociationQuery and are the target object  
2216 of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then  
2217 continue to the next numbered rule (Rule 2).

2218  
2219 If a Federation Query is specified within the SourceAssociationBranch, then let ROS be  
2220 the set of Federation instances that satisfy the FederationQuery and are the source object  
2221 of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then  
2222 continue to the next numbered rule.

2223  
2224 If a RegistryQuery is specified within the SourceAssociationBranch, then let ROS be the  
2225 set of Registry instances that satisfy the RegistryQuery and are the source object of some  
2226 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue  
2227 to the next numbered rule.

2228

2229 If a SubscriptionQuery is specified within the SourceAssociationBranch, then let ROS be  
2230 the set of Subscription instances that satisfy the SubscriptionQuery and are the source  
2231 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2232 then continue to the next numbered rule.

2233

2234 If a UserQuery is specified within the SourceAssociationBranch, then let ROS be the set  
2235 of User instances that satisfy the UserQuery and are the source object of some element of  
2236 AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next  
2237 numbered rule.

2238

2239 j) If a TargetAssociationBranch element is not specified then go to the next step; otherwise,  
2240 let x be a remaining registry object in RO. If x is not the target object of some  
2241 Association instance, then remove x from RO. If RO is empty then continue to the next  
2242 numbered rule; otherwise, treat each TargetAssociationBranch element separately as  
2243 follows:

2244

2245 If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be  
2246 the set of all Association instances that have x as a target object; otherwise, let AF be the  
2247 set of Association instances that satisfy the AssociationFilter and have x as the target  
2248 object. If AF is empty, then remove x from RO. If RO is empty then continue to the next  
2249 numbered rule.

2250

2251 If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be  
2252 the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source  
2253 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2254 then continue to the next numbered rule.

2255

2256 If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let  
2257 ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and  
2258 are the source object of some element of AF. If ROS is empty, then remove x from RO. If  
2259 RO is empty then continue to the next numbered rule.

2260

2261 If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS  
2262 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the  
2263 source object of some element of AF. If ROS is empty, then remove x from RO. If RO is  
2264 empty then continue to the next numbered rule.

2265

2266 If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS  
2267 be the set of

2268 RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of  
2269 some element of AF. If ROS is empty, then remove x from RO. If RO is empty then  
2270 continue to the next numbered rule.

2271

2272 If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let  
2273 ROS be the set of ClassificationScheme instances that satisfy the  
2274 ClassificationSchemeQuery and are the source object of some element of AF. If ROS is  
2275 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.  
2276

2277 If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let  
2278 ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery  
2279 and are the source object of some element of AF. If ROS is empty, then remove x from  
2280 RO. If RO is empty then continue to the next numbered rule.  
2281

2282 If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS  
2283 be the set of Organization instances that satisfy the OrganizationQuery and are the source  
2284 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2285 then continue to the next numbered rule.  
2286

2287 If an AuditableEventQuery is specified within the TargetAssociationBranch, then let  
2288 ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are  
2289 the source object of some element of AF. If ROS is empty, then remove x from RO. If  
2290 RO is empty then continue to the next numbered rule.  
2291

2292 If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let  
2293 ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and  
2294 are the source object of some element of AF. If ROS is empty, then remove x from RO. If  
2295 RO is empty then continue to the next numbered rule.  
2296

2297 If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let  
2298 ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are  
2299 the source object of some element of AF. If ROS is empty, then remove x from RO. If  
2300 RO is empty then continue to the next numbered rule.  
2301

2302 If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the  
2303 set of Service instances that satisfy the ServiceQuery and are the source object of some  
2304 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue  
2305 to the next numbered rule.  
2306

2307 If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be  
2308 the set of Classification instances that satisfy the ClassificationQuery and are the source  
2309 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2310 then continue to the next numbered rule (Rule 2).  
2311

2312 If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS  
2313 be the set of ServiceBinding instances that are the source object of some element of AF.  
2314 If ROS is empty, then remove x from RO. If RO is empty then continue to the next  
2315 numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is  
2316 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then  
2317 remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then  
2318 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the  
2319 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as  
2320 follows:  
2321 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link  
2322 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
2323 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
2324 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then  
2325 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
2326 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl  
2327 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:  
2328 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is  
2329 not a specification link for some registry object in RO, then remove sl from SL. If SL is  
2330 empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is  
2331 empty then continue to the next numbered rule. If a RegistryEntryQuery element is  
2332 specified within the SpecificationLinkBranch then let sl be a remaining specification link  
2333 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the  
2334 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some  
2335 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.  
2336 If ROS is empty then remove x from RO. If RO is empty then continue to the next  
2337 numbered rule.  
2338

2339 If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let  
2340 ROS be the set of SpecificationLink instances that are the source object of some element  
2341 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the  
2342 next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is  
2343 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
2344 remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then  
2345 continue to the next numbered rule. If a RegistryObjectQuery element is specified within  
2346 the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat  
2347 RegistryObjectQuery element as follows: Let RO be the result set of the  
2348 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some  
2349 registry object in RO, then remove sl from ROS. If ROS is empty then remove x from  
2350 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery  
2351 element is specified within the SpecificationLinkBranch then let sl be a remaining  
2352 specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the  
2353 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification  
2354 link for some registry entry in RE, then remove sl from ROS. If ROS is empty then  
2355 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
2356 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT  
2357 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and  
2358 are the target service binding of some element of ROT. If SBT is empty then remove sb  
2359 from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the  
2360 next numbered rule.

2361  
2362 If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be  
2363 the set of Association instances that satisfy the AssociationQuery and are the source  
2364 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2365 then continue to the next numbered rule (Rule 2).

2366  
2367 If a Federation Query is specified within the TargetAssociationBranch, then let ROS be  
2368 the set of Federation instances that satisfy the FederationQuery and are the source object  
2369 of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then  
2370 continue to the next numbered rule.

2371  
2372 If a RegistryQuery is specified within the TargetAssociationBranch, then let ROS be the  
2373 set of Registry instances that satisfy the RegistryQuery and are the source object of some  
2374 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue  
2375 to the next numbered rule.

2376  
2377 If a SubscriptionQuery is specified within the TargetAssociationBranch, then let ROS be  
2378 the set of Subscription instances that satisfy the SubscriptionQuery and are the source  
2379 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
2380 then continue to the next numbered rule.

2381

2382 If a UserQuery is specified within the TargetAssociationBranch, then let ROS be the set  
 2383 of User instances that satisfy the UserQuery and are the source object of some element of  
 2384 AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next  
 2385 numbered rule.

- 2386 2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set  
 2387 RO to be the result of the RegistryObjectQuery.
- 2388 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2389 within the RegistryResponse.

### 2390 Examples

2391 A client application needs all items that are classified by two different classification schemes,  
 2392 one based on "Industry" and another based on "Geography". Both schemes have been defined by  
 2393 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",  
 2394 respectively. The following query identifies registry entries for all registered items that are  
 2395 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of  
 2396 "Asia/Japan".

```

2397 <AdhocQueryRequest>
2398   <ResponseOption returnType = "RegistryEntry"/>
2399   <FilterQuery>
2400     <RegistryObjectQuery>
2401       <ClassifiedByBranch>
2402         <ClassificationFilter>
2403           <Clause>
2404             <SimpleClause leftArgument = "path">
2405               <StringClause stringPredicate = "Equal">//Automotive</StringClause>
2406             </SimpleClause>
2407           </Clause>
2408         </ClassificationFilter>
2409       </ClassifiedByBranch>
2410       <ClassificationSchemeQuery>
2411         <NameBranch>
2412           <LocalizedStringFilter>
2413             <Clause>
2414               <SimpleClause leftArgument = "value">
2415                 <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
2416               </SimpleClause>
2417             </Clause>
2418           </LocalizedStringFilter>
2419         </NameBranch>
2420       </ClassificationSchemeQuery>
2421     </ClassifiedByBranch>
2422     <ClassifiedByBranch>
2423       <ClassificationFilter>
2424         <Clause>
2425           <SimpleClause leftArgument = "path">
2426             <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
2427           </SimpleClause>
2428         </Clause>
2429       </ClassificationFilter>
2430     </ClassifiedByBranch>
2431     <ClassificationSchemeQuery>
2432       <NameBranch>
2433         <LocalizedStringFilter>
2434           <Clause>
  
```

```

2434     <SimpleClause leftArgument = "value">
2435         <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
2436     </SimpleClause>
2437 </Clause>
2438 </LocalizedStringFilter>
2439 </NameBranch>
2440 </ClassificationSchemeQuery>
2441 </ClassifiedByBranch>
2442 </RegistryObjectQuery>
2443 </FilterQuery>
2444 </AdhocQueryRequest>
2445

```

2446 A client application wishes to identify all RegistryObject instances that are classified by some  
 2447 internal classification scheme and have some given keyword as part of the description of one of  
 2448 the classification nodes of that classification scheme. The following query identifies all such  
 2449 RegistryObject instances. The query takes advantage of the knowledge that the classification  
 2450 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode  
 2451 instances.

```

2452
2453 <AdhocQueryRequest>
2454   <ResponseOption returnType = "RegistryObject"/>
2455   <FilterQuery>
2456     <RegistryObjectQuery>
2457       <ClassifiedByBranch>
2458         <ClassificationNodeQuery>
2459           <DescriptionBranch>
2460             <LocalizedStringFilter>
2461               <Clause>
2462                 <SimpleClause leftArgument = "value">
2463                   <StringClause stringPredicate = "Equal">transistor</StringClause>
2464                 </SimpleClause>
2465               </Clause>
2466             </LocalizedStringFilter>
2467           </DescriptionBranch>
2468         </ClassificationNodeQuery>
2469       </ClassifiedByBranch>
2470     </RegistryObjectQuery>
2471   </FilterQuery>
2472 </AdhocQueryRequest>
2473

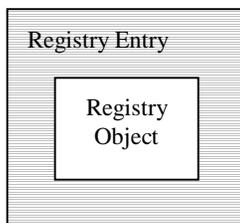
```

### 2474 8.2.3 RegistryEntryQuery

#### 2475 Purpose

2476 To identify a set of registry entry instances as the result of a query over selected registry  
 2477 metadata.

#### 2478 2479 ebRIM Binding



2480 **Figure 38: ebRIM Binding for RegistryEntryQuery**

2481 **Definition**

```

2482 <complexType name="RegistryEntryQueryType">
2483   <complexContent>
2484     <extension base="tns:RegistryObjectQueryType">
2485       <sequence>
2486         <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
2487       </sequence>
2488     </extension>
2489   </complexContent>
2490 </complexType>
2491 <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
2492
2493 <element name="RegistryEntryQueryResult">
2494   <complexType>
2495     <choice minOccurs="0" maxOccurs="unbounded">
2496       <element ref="rim:ObjectRef" />
2497       <element ref="rim:ClassificationScheme" />
2498       <element ref="rim:ExtrinsicObject" />
2499       <element ref="rim:RegistryEntry" />
2500       <element ref="rim:RegistryObject" />
2501       <element ref="rim:RegistryPackage" />
2502       <element ref="rim:Service" />
2503       <element ref="rim:Federation" />
2504       <element ref="rim:Registry" />
2505     </choice>
2506   </complexType>
2507 </element>
2508
2509
  
```

2510 **Semantic Rules**

- 2511 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
 2512 steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
  - 2513 a) If RE is empty then continue to the next numbered rule.
  - 2514 b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
 2515 registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
 2516 If RE is empty then continue to the next numbered rule.
  - 2517 c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
 2518 RegistryObjectQuery over RE as explained in Section 8.2.2.
- 2519 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, set RE
 2520 to be the result of the RegistryEntryQuery.

- 2521 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
2522 within the RegistryResponse.

### 2523 Examples

2524 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if  
2525 they have registered any of their business documents in the Registry. The following query  
2526 returns a set of registry entry identifiers for currently registered items submitted by any  
2527 organization whose name includes the string "XYZ". It does not return any registry entry  
2528 identifiers for superseded, replaced, deprecated, or withdrawn items.

```

2530 <AdhocQueryRequest>
2531   <ResponseOption returnType = "ObjectRef"/>
2532   <FilterQuery>
2533     <RegistryEntryQuery>
2534       <TargetAssociationBranch>
2535         <AssociationFilter>
2536           <Clause>
2537             <SimpleClause leftArgument = "associationType">
2538               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2539             </SimpleClause>
2540           </Clause>
2541         </AssociationFilter>
2542         <OrganizationQuery>
2543           <NameBranch>
2544             <LocalizedStringFilter>
2545               <Clause>
2546                 <SimpleClause leftArgument = "value">
2547                   <StringClause stringPredicate = "Contains">XYZ</StringClause>
2548                 </SimpleClause>
2549               </Clause>
2550             </LocalizedStringFilter>
2551           </NameBranch>
2552         </OrganizationQuery>
2553       </TargetAssociationBranch>
2554       <RegistryEntryFilter>
2555         <Clause>
2556           <SimpleClause leftArgument = "status">
2557             <StringClause stringPredicate = "Equal">Approved</StringClause>
2558           </SimpleClause>
2559         </Clause>
2560       </RegistryEntryFilter>
2561     </RegistryEntryQuery>
2562   </FilterQuery>
2563 </AdhocQueryRequest>
2564
```

2565 A client is using the United Nations Standard Product and Services Classification (UNSPSC)  
2566 scheme and wants to identify all companies that deal with products classified as "Integrated  
2567 circuit components", i.e. UNSPSC code "321118". The client knows that companies have  
2568 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each  
2569 such profile has been classified by UNSPSC according to the products the company deals with.  
2570 However, the client does not know if the UNSPSC classification scheme is internal or external to  
2571 this registry. The following query returns a set of approved registry entry instances for CPP's of  
2572 companies that deal with integrated circuit components.

```

2573 <AdhocQueryRequest>
2574   <ResponseOption returnType = "RegistryEntry"/>
2575   <FilterQuery>
2576     <RegistryEntryQuery>
2577       <ClassifiedByBranch>
2578         <ClassificationFilter>
2579           <Clause>
2580             <SimpleClause leftArgument = "nodeRepresentation">
2581               <StringClause stringPredicate = "Equal">321118</StringClause>
2582             </SimpleClause>
2583           </Clause>
2584         </ClassificationFilter>
2585       <ClassificationSchemeQuery>
2586         <NameBranch>
2587           <LocalizedStringFilter>
2588             <Clause>
2589               <SimpleClause leftArgument = "value">
2590                 <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
2591               </SimpleClause>
2592             </Clause>
2593           </LocalizedStringFilter>
2594         </NameBranch>
2595       </ClassificationSchemeQuery>
2596     </ClassifiedByBranch>
2597   <RegistryEntryFilter>
2598     <Clause>
2599       <CompoundClause connectivePredicate = "And">
2600         <Clause>
2601           <SimpleClause leftArgument = "objectType">
2602             <StringClause stringPredicate = "Equal">CPP</StringClause>
2603           </SimpleClause>
2604         </Clause>
2605         <Clause>
2606           <SimpleClause leftArgument = "status">
2607             <StringClause stringPredicate = "Equal">Approved</StringClause>
2608           </SimpleClause>
2609         </Clause>
2610       </CompoundClause>
2611     </Clause>
2612   </RegistryEntryFilter>
2613 </RegistryEntryQuery>
2614 </FilterQuery>
2615 </AdhocQueryRequest>
2616
2617

```

## 2618 8.2.4 AssociationQuery

### 2619 Purpose

2620 To identify a set of association instances as the result of a query over selected registry metadata.

2621

### 2622 ebRIM Binding

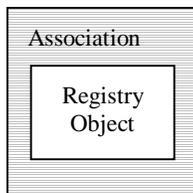


Figure 39: ebRIM Binding for AssociationQuery

2623

#### 2624 Definition

```

2625 <complexType name = "AssociationQueryType">
2626   <complexContent>
2627     <extension base = "tns:RegistryObjectQueryType">
2628       <sequence>
2629         <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
2630       </sequence>
2631     </extension>
2632   </complexContent>
2633 </complexType>
2634 <element name = "AssociationQuery" type = "tns:AssociationQueryType"/>
2635 <element name="AssociationQueryResult">
2636   <complexType>
2637     <choice minOccurs="0" maxOccurs="unbounded">
2638       <element ref="rim:ObjectRef" />
2639       <element ref="rim:RegistryObject" />
2640       <element ref="rim:Association" />
2641     </choice>
2642   </complexType>
2643 </element>
2644
2645
2646

```

#### 2647 Semantic Rules

- 2648 1. Let A denote the set of all persistent Association instances in the Registry. The following
 2649 steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
  - 2650 a) If A is empty then continue to the next numbered rule.
  - 2651 b) If an AssociationFilter element is not directly contained in the AssociationQuery element,
 2652 then go to the next step; otherwise let x be an association instance in A. If x does not
 2653 satisfy the AssociationFilter then remove x from A. If A is empty then continue to the
 2654 next numbered rule.
  - 2655 c) Let A be the set of remaining Association instances. Evaluate inherited
 2656 RegistryObjectQuery over A as explained in Section 8.2.2.
- 2657 2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to
 2658 be the result of the AssociationQuery.
- 2659 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2660 within the RegistryResponse.

#### 2661 Examples

2662 A client application wishes to identify a set of associations that are 'equivalentTo' a set of other  
 2663 associations.

```
2664 <AdhocQueryRequest">
2665   <ResponseOption returnType="LeafClass" />
2666   <FilterQuery>
2667     <AssociationQuery>
2668       <SourceAssociationBranch>
2669         <AssociationFilter>
2670           <Clause>
2671             <SimpleClause leftArgument="associationType">
2672               <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
2673             </SimpleClause>
2674           </Clause>
2675         </AssociationFilter>
2676       </SourceAssociationBranch>
2677     </AssociationQuery>
2678     <AssociationFilter>
2679       <Clause>
2680         <SimpleClause leftArgument="associationType">
2681           <StringClause stringPredicate="StartsWith">Sin</StringClause>
2682         </SimpleClause>
2683       </Clause>
2684     </AssociationFilter>
2685   </AssociationQuery>
2686 </SourceAssociationBranch>
2687 </AssociationFilter>
2688 <Clause>
2689   <SimpleClause leftArgument="associationType">
2690     <StringClause stringPredicate="StartsWith">Son</StringClause>
2691   </SimpleClause>
2692 </Clause>
2693 </AssociationFilter>
2694 </AssociationQuery>
2695 </FilterQuery>
2696 </AdhocQueryRequest>
2697
```

2697

2698 **8.2.5 AuditableEventQuery**2699 **Purpose**

2700 To identify a set of auditable event instances as the result of a query over selected registry  
2701 metadata.

2702 **ebRIM Binding**

2703

2704

2705

2706

2707

2708

2709

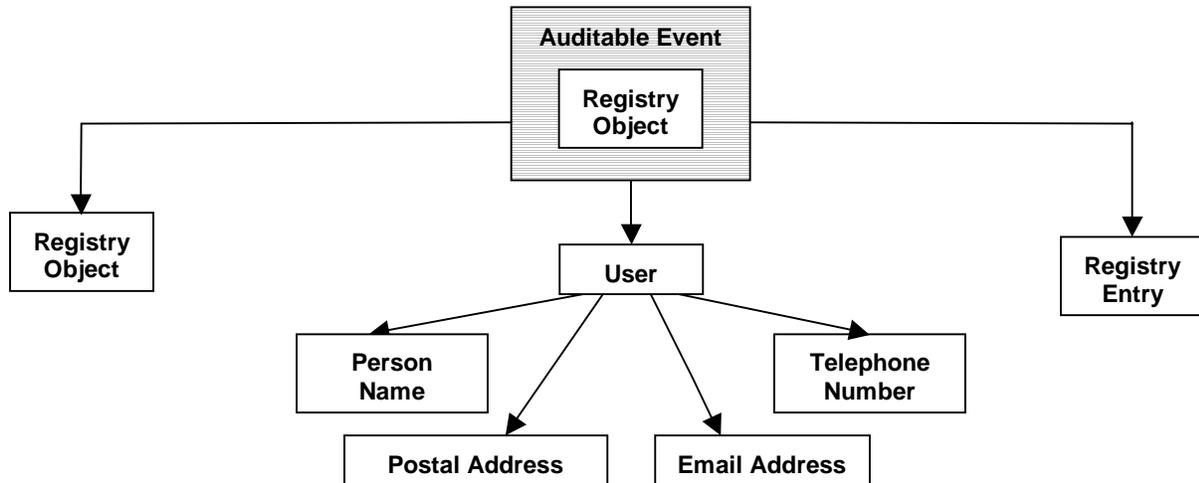
2710

2711

2712

2713

2714



2715 **Figure 40: ebRIM Binding for AuditableEventQuery**

2715 **Definition**

2716

2717

2718

2719

2720

2721

2722

2723

2724

2725

2726

2727

2728

2729

2730

2731

2732

2733

2734

2735

2736

2737

2738

2739

```

<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>
  
```

2740

2741 **Semantic Rules**

- 2742 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The  
 2743 following steps will eliminate instances in AE that do not satisfy the conditions of the  
 2744 specified filters.
- 2745 a) If AE is empty then continue to the next numbered rule.
- 2746 b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an  
 2747 auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from  
 2748 AE. If AE is empty then continue to the next numbered rule.
- 2749 c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let  
 2750 x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:  
 2751 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is  
 2752 not an auditable event for some registry object in RO, then remove x from AE. If AE is  
 2753 empty then continue to the next numbered rule.
- 2754 d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x  
 2755 be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let  
 2756 RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an  
 2757 auditable event for some registry entry in RE, then remove x from AE. If AE is empty  
 2758 then continue to the next numbered rule.
- 2759 e) If a UserQuery is not specified then go to the next step; otherwise, let x be a remaining  
 2760 auditable event in AE. If the defining user of x does not satisfy the UserQuery, then  
 2761 remove x from AE. If AE is empty then continue to the next numbered rule.
- 2762 f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited  
 2763 RegistryObjectQuery over AE as explained in Section 8.2.2.
- 2764 2. If AE is empty, then raise the warning: *auditable event query result is empty*; otherwise set  
 2765 AE to be the result of the AuditableEventQuery.
- 2766 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2767 within the RegistryResponse.

2768 **Examples**

2769 A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The  
 2770 client is now interested in all events since the beginning of the year that have impacted that item.  
 2771 The following query will return a set of AuditableEvent instances for all such events.

```

2772
2773 <AdhocQueryRequest>
2774   <ResponseOption returnType = "LeafClass"/>
2775   <FilterQuery>
2776     <AuditableEventQuery>
2777       <AuditableEventFilter>
2778         <Clause>
2779           <SimpleClause leftArgument = "timestamp">
2780             <RationalClause logicalPredicate = "GE">
2781               <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2782             </RationalClause>
2783           </SimpleClause>

```

```

2784     </Clause>
2785   </AuditableEventFilter>
2786   <RegistryEntryQuery>
2787     <NameBranch>
2788       <LocalizedStringFilter>
2789         <Clause>
2790           <SimpleClause leftArgument = "value">
2791             <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
2792           </SimpleClause>
2793         </Clause>
2794       </LocalizedStringFilter>
2795     </NameBranch>
2796   </RegistryEntryQuery>
2797 </AuditableEventQuery>
2798 </FilterQuery>
2799 </AdhocQueryRequest
2800

```

2801 A client company has many registered objects in the Registry. The Registry allows events  
 2802 submitted by other organizations to have an impact on your registered items, e.g. new  
 2803 classifications and new associations. The following query will return a set of identifiers for all  
 2804 auditable events, invoked by some other party, that had an impact on an item submitted by  
 2805 “myorg”.

```

2806 <AdhocQueryRequest>
2807   <ResponseOption returnType = "LeafClass"/>
2808   <FilterQuery>
2809     <AuditableEventQuery>
2810       <RegistryEntryQuery>
2811         <TargetAssociationBranch>
2812           <AssociationFilter>
2813             <Clause>
2814               <SimpleClause leftArgument = "associationType">
2815                 <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2816               </SimpleClause>
2817             </Clause>
2818           </AssociationFilter>
2819         <OrganizationQuery>
2820           <NameBranch>
2821             <LocalizedStringFilter>
2822               <Clause>
2823                 <SimpleClause leftArgument = "value">
2824                   <StringClause stringPredicate = "Equal">myorg</StringClause>
2825                 </SimpleClause>
2826               </Clause>
2827             </LocalizedStringFilter>
2828           </NameBranch>
2829         </OrganizationQuery>
2830       </TargetAssociationBranch>
2831     </RegistryEntryQuery>
2832   <UserBranch>
2833     <OrganizationQuery>
2834       <NameBranch>
2835         <LocalizedStringFilter>
2836           <Clause>
2837             <SimpleClause leftArgument = "value">
2838               <StringClause stringPredicate = "NotEqual">myorg</StringClause>
2839             </SimpleClause>
2840           </Clause>
2841         </LocalizedStringFilter>
2842       </NameBranch>
2843     </OrganizationQuery>
2844   </UserBranch>
2845 </FilterQuery>
2846 </AdhocQueryRequest>

```

```

2840     </SimpleClause>
2841     </Clause>
2842     </LocalizedStringFilter>
2843     </NameBranch>
2844     </OrganizationQuery>
2845     </UserBranch>
2846     </AuditableEventQuery>
2847     </FilterQuery>
2848 </AdhocQueryRequest>
2849

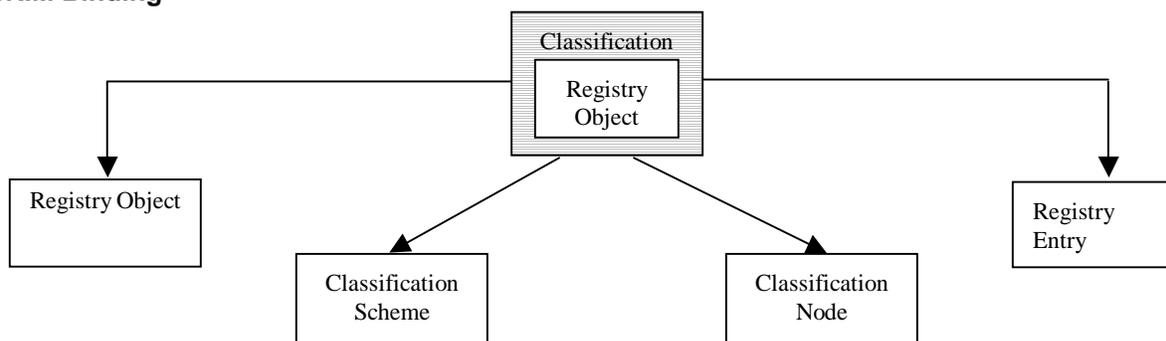
```

## 2850 8.2.6 ClassificationQuery

### 2851 Purpose

2852 To identify a set of classification instances as the result of a query over selected registry  
2853 metadata.

### 2854 ebRIM Binding



2855 **Figure 41: ebRIM Binding for ClassificationQuery**

### 2856 Definition

```

2857
2858 <complexType name = "ClassificationQueryType">
2859   <complexContent>
2860     <extension base = "tns:RegistryObjectQueryType">
2861       <sequence>
2862         <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
2863         <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
2864         <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
2865         <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
2866         <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
2867       </sequence>
2868     </extension>
2869   </complexContent>
2870 </complexType>
2871 <element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>
2872
2873 <element name="ClassificationQueryResult">
2874   <complexType>
2875     <choice minOccurs="0" maxOccurs="unbounded">
2876       <element ref="rim:ObjectRef" />
2877       <element ref="rim:RegistryObject" />
2878       <element ref="rim:Classification" />
2879     </choice>

```

```

2880 </complexType>
2881 </element>
2882

```

### 2883 Semantic Rules

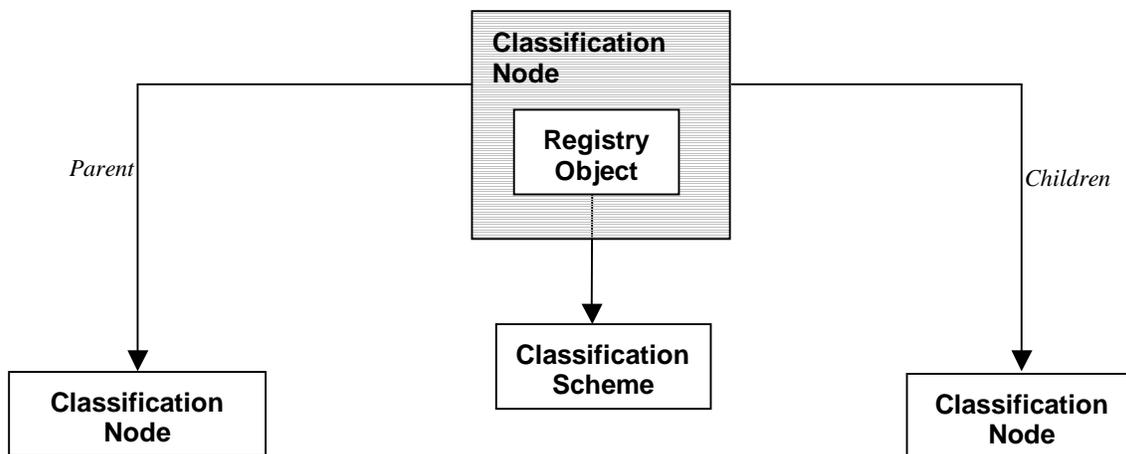
- 2884 1. Let C denote the set of all persistent Classification instances in the Registry. The following  
2885 steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
- 2886 a) If C is empty then continue to the next numbered rule.
- 2887 b) If a ClassificationFilter element is not directly contained in the ClassificationQuery  
2888 element, then go to the next step; otherwise let x be an classification instance in C. If x  
2889 does not satisfy the ClassificationFilter then remove x from C. If C is empty then  
2890 continue to the next numbered rule.
- 2891 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x  
2892 be a remaining classification in C. If the defining classification scheme of x does not  
2893 satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C.  
2894 If C is empty then continue to the next numbered rule.
- 2895 d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be  
2896 a remaining classification in C. If the classification node of x does not satisfy the  
2897 ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is  
2898 empty then continue to the next numbered rule.
- 2899 e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let  
2900 x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let  
2901 RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a  
2902 classification of at least one registry object in RO, then remove x from C. If C is empty  
2903 then continue to the next numbered rule.
- 2904 f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x  
2905 be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE  
2906 be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a  
2907 classification of at least one registry entry in RE, then remove x from C. If C is empty  
2908 then continue to the next numbered rule.
- 2909 2. If C is empty, then raise the warning: *classification query result is empty*; otherwise  
2910 otherwise, set C to be the result of the ClassificationQuery.
- 2911 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
2912 within the RegistryResponse.

### 2913 8.2.7 ClassificationNodeQuery

#### 2914 Purpose

2915 To identify a set of classification node instances as the result of a query over selected registry  
2916 metadata.

#### 2917 ebRIM Binding



2918 **Figure 42: ebRIM Binding for ClassificationNodeQuery**

2919 **Definition**

```

2920 <complexType name="ClassificationNodeQueryType">
2921   <complexContent>
2922     <extension base="tns:RegistryObjectQueryType">
2923       <sequence>
2924         <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
2925         <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
2926         <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
2927           maxOccurs="1" />
2928         <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
2929           minOccurs="0" maxOccurs="unbounded" />
2930       </sequence>
2931     </extension>
2932   </complexContent>
2933 </complexType>
2934 <element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />
2935
2936
2937 <element name="ClassificationNodeQueryResult">
2938   <complexType>
2939     <choice minOccurs="0" maxOccurs="unbounded">
2940       <element ref="rim:ObjectRef" />
2941       <element ref="rim:RegistryObject" />
2942       <element ref="rim:ClassificationNode" />
2943     </choice>
2944   </complexType>
2945 </element>
2946
  
```

2947 **Semantic Rules**

- 2948 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The  
 2949 following steps will eliminate instances in CN that do not satisfy the conditions of the  
 2950 specified filters.
- 2951 a) If CN is empty then continue to the next numbered rule.
- 2952 b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a  
 2953 classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove  
 2954 x from CN. If CN is empty then continue to the next numbered rule.

- 2955 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x  
 2956 be a remaining classification node in CN. If the defining classification scheme of x does  
 2957 not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x  
 2958 from CN. If CN is empty then continue to the next numbered rule.
- 2959 d) If a ClassificationNodeParentBranch element is not specified, then go to the next step;  
 2960 otherwise, let x be a remaining classification node in CN and execute the following  
 2961 paragraph with n=x.  
 2962 Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base  
 2963 level node), then remove x from CN and go to the next step; otherwise, let p be the parent  
 2964 node of n. If a ClassificationNodeFilter element is directly contained in the  
 2965 ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,  
 2966 then remove x from CN. If CN is empty then continue to the next numbered rule. If a  
 2967 ClassificationSchemeQuery element is directly contained in the  
 2968 ClassificationNodeParentBranch and if defining classification scheme of p does not  
 2969 satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then  
 2970 continue to the next numbered rule.  
 2971 If another ClassificationNodeParentBranch element is directly contained within this  
 2972 ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.
- 2973 e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next  
 2974 numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the  
 2975 parent node of some ClassificationNode instance, then remove x from CN and if CN is  
 2976 empty continue to the next numbered rule; otherwise, treat each  
 2977 ClassificationNodeChildrenBranch element separately and execute the following  
 2978 paragraph with n = x.  
 2979 Let n be a classification node instance. If a ClassificationNodeFilter element is not  
 2980 specified within the ClassificationNodeChildrenBranch element then let CNC be the set  
 2981 of all classification nodes that have n as their parent node; otherwise, let CNC be the set  
 2982 of all classification nodes that satisfy the ClassificationNodeFilter and have n as their  
 2983 parent node. If CNC is empty, then remove x from CN and if CN is empty continue to the  
 2984 next numbered rule; otherwise, let c be any member of CNC. If a  
 2985 ClassificationSchemeQuery element is directly contained in the  
 2986 ClassificationNodeChildrenBranch and if the defining classification scheme of c does not  
 2987 satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then  
 2988 remove x from CN. If CN is empty then continue to the next numbered rule; otherwise,  
 2989 let y be an element of CNC and continue with the next paragraph.  
 2990 If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly  
 2991 contain another ClassificationNodeChildrenBranch element, then continue to the next  
 2992 numbered rule; otherwise, repeat the previous paragraph with the new  
 2993 ClassificationNodeChildrenBranch element and with n = y.
- 2994 f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited  
 2995 RegistryObjectQuery over CN as explained in Section 8.2.2.
- 2996 2. If CN is empty, then raise the warning: *classification node query result is empty*; otherwise  
 2997 set CN to be the result of the ClassificationNodeQuery.
- 2998 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2999 within the RegistryResponse.

### 3000 Path Filter Expression usage in ClassificationNodeFilter

3001 The path filter expression is used to match classification nodes in ClassificationNodeFilter  
3002 elements involving the path attribute of the ClassificationNode class as defined by the getPath  
3003 method in [ebRIM].

3004 The path filter expressions are based on a very small and proper sub-set of location path syntax  
3005 of XPath.

3006 The path filter expression syntax includes support for matching multiple nodes by using wild  
3007 card syntax as follows:

- 3008 • Use of '\*' as a wildcard in place of any path element in the pathFilter.
- 3009 • Use of '/' syntax to denote any descendent of a node in the pathFilter. Support for '/' syntax  
3010 is optional.

3011 It is defined by the following BNF grammar:

```
3012 pathFilter ::= '/' schemeId nodePath
3013 nodePath ::= slashes nodeCode
3014           | slashes '*'
3015           | slashes nodeCode ( nodePath )?
3016 Slashes  ::= '/' | '/'
```

3019 In the above grammar, schemeId is the id attribute of the ClassificationScheme instance. In the  
3020 above grammar nodeCode is defined by NCName production as defined by  
3021 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

3022 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a  
3023 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause  
3024 is a PATH Filter expression.

3025 This is illustrated in the following example that matches all second level nodes in  
3026 ClassificationScheme with id 'Geography-id' and with code 'Japan':

```
3027 <ClassificationNodeQuery>
3028   <ClassificationNodeFilter>
3029     <Clause>
3030       <SimpleClause leftArgument = "path">
3031         <StringClause stringPredicate = "Equal"> //Geography-id/* /Japan </StringClause>
3032       </SimpleClause>
3033     </Clause>
3034   </ClassificationNodeFilter>
3035 </ClassificationNodeQuery>
```

### 3038 Use Cases and Examples of Path Filter Expressions

3039 The following table lists various use cases and examples using the sample Geography scheme  
3040 below:

```
3041 <ClassificationScheme id='Geography-id' name="Geography" />
3042
3043 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
3044 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
3045
3046 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
3047 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
3048 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

3051

**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'

**3052 Examples**

3053 A client application wishes to identify all of the classification nodes in the first three levels of a  
 3054 classification scheme hierarchy. The client knows that the name of the underlying classification  
 3055 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three  
 3056 levels.

```

3057 <AdhocQueryRequest>
3058   <ResponseOption returnType = "LeafClass"/>
3059   <FilterQuery>
3060     <ClassificationNodeQuery>
3061       <ClassificationNodeFilter>
3062         <Clause>
3063           <SimpleClause leftArgument = "levelNumber">
3064             <RationalClause logicalPredicate = "LE">
3065               <IntClause>3</IntClause>
3066             </RationalClause>
3067           </SimpleClause>
3068         </Clause>
3069       </ClassificationNodeFilter>
3070     </ClassificationNodeQuery>
3071     <ClassificationSchemeQuery>
3072       <NameBranch>
3073         <LocalizedStringFilter>
3074           <Clause>
3075             <SimpleClause leftArgument = "value">
3076               <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
3077             </SimpleClause>
3078           </Clause>

```

```

3079     </LocalizedStringFilter>
3080     </NameBranch>
3081     </ClassificationSchemeQuery>
3082     </ClassificationNodeQuery>
3083     </FilterQuery>
3084 </AdhocQueryRequest>
3085

```

3086 If, instead, the client wishes all levels returned, they could simply delete the  
3087 ClassificationNodeFilter element from the query.

3088 The following query finds all children nodes of a first level node whose code is NorthAmerica.

```

3089 <AdhocQueryRequest>
3090   <ResponseOption returnType = "LeafClass"/>
3091   <FilterQuery>
3092     <ClassificationNodeQuery>
3093       <ClassificationNodeFilter>
3094         <Clause>
3095           <SimpleClause leftArgument = "path">
3096             <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
3097           </SimpleClause>
3098         </Clause>
3099       </ClassificationNodeFilter>
3100     </ClassificationNodeQuery>
3101   </FilterQuery>
3102 </AdhocQueryRequest>
3103
3104

```

3105 The following query finds all third level nodes with code of Tokyo.

```

3106 <AdhocQueryRequest>
3107   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
3108   <FilterQuery>
3109     <ClassificationNodeQuery>
3110       <ClassificationNodeFilter>
3111         <Clause>
3112           <SimpleClause leftArgument = "path">
3113             <StringClause stringPredicate = "Equal">/Geography-id-*/*/Tokyo</StringClause>
3114           </SimpleClause>
3115         </Clause>
3116       </ClassificationNodeFilter>
3117     </ClassificationNodeQuery>
3118   </FilterQuery>
3119 </AdhocQueryRequest>
3120
3121

```

## 3122 8.2.8 ClassificationSchemeQuery

### 3123 Purpose

3124 To identify a set of classification scheme instances as the result of a query over selected registry  
3125 metadata.

### 3126 ebRIM Binding

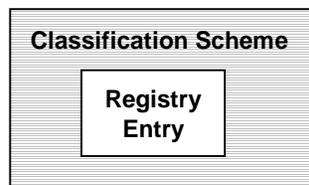


Figure 43: ebRIM Binding for ClassificationSchemeQuery

3127

### 3128 Definition

```

3129 <complexType name="ClassificationSchemeQueryType">
3130   <complexContent>
3131     <extension base="tns:RegistryEntryQueryType">
3132       <sequence>
3133         <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
3134       </sequence>
3135     </extension>
3136   </complexContent>
3137 </complexType>
3138 </complexType>
3139 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
3140
3141 <element name="ClassificationSchemeQueryResult">
3142   <complexType>
3143     <choice minOccurs="0" maxOccurs="unbounded">
3144       <element ref="rim:ObjectRef"/>
3145       <element ref="rim:RegistryObject"/>
3146       <element ref="rim:RegistryEntry"/>
3147       <element ref="rim:ClassificationScheme"/>
3148     </choice>
3149   </complexType>
3150 </element>
3151
  
```

### 3152 Semantic Rules

- 3153 1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The  
3154 following steps will eliminate instances in CS that do not satisfy the conditions of the  
3155 specified filters.
  - 3156 a) If CS is empty then continue to the next numbered rule.
  - 3157 b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x  
3158 be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,  
3159 then remove x from CS. If CS is empty then continue to the next numbered rule.
  - 3160 c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited  
3161 RegistryEntryQuery over CS as explained in Section 8.2.3.
- 3162 2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,  
3163 set CS to be the result of the ClassificationSchemeQuery.
- 3164 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
3165 within the RegistryResponse.

### 3166 Examples

3167 A client application wishes to identify all classification scheme instances in the Registry.

```

3168 <AdhocQueryRequest>
3169   <ResponseOption returnType = "LeafClass"/>
3170   <FilterQuery>
3171     <ClassificationSchemeQuery/>
3172   </FilterQuery>
3173 </AdhocQueryRequest>

```

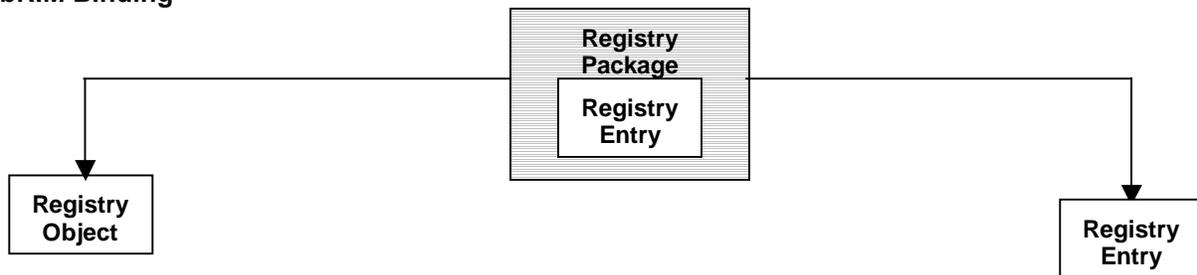
3174

## 3175 8.2.9 RegistryPackageQuery

### 3176 Purpose

3177 To identify a set of registry package instances as the result of a query over selected registry  
3178 metadata.

### 3179 ebRIM Binding



3180

Figure 44: ebRIM Binding for RegistryPackageQuery

### 3181 Definition

```

3182 <complexType name="RegistryPackageQueryType">
3183   <complexContent>
3184     <extension base="tns:RegistryEntryQueryType">
3185       <sequence>
3186         <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
3187         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
3188         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
3189       </sequence>
3190     </extension>
3191   </complexContent>
3192 </complexType>
3193 <element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />
3194
3195 <element name="RegistryPackageQueryResult">
3196   <complexType>
3197     <choice minOccurs="0" maxOccurs="unbounded">
3198       <element ref="rim:ObjectRef" />
3199       <element ref="rim:RegistryEntry" />
3200       <element ref="rim:RegistryObject" />
3201       <element ref="rim:RegistryPackage" />
3202     </choice>
3203   </complexType>
3204 </element>
3205
3206

```

3207 **Semantic Rules**

- 3208 1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The  
 3209 following steps will eliminate instances in RP that do not satisfy the conditions of the  
 3210 specified filters.
- 3211 a) If RP is empty then continue to the next numbered rule.
- 3212 b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule;  
 3213 otherwise, let x be a registry package instance in RP. If x does not satisfy the  
 3214 RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next  
 3215 numbered rule.
- 3216 c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery  
 3217 element then treat each RegistryObjectQuery as follows: let RO be the set of  
 3218 RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2  
 3219 and let PO be the subset of RO that are members of the package x. If PO is empty, then  
 3220 remove x from RP. If RP is empty then continue to the next numbered rule. If a  
 3221 RegistryEntryQuery element is directly contained in the RegistryPackageQuery element  
 3222 then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry  
 3223 instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be  
 3224 the subset of RE that are members of the package x. If PE is empty, then remove x from  
 3225 RP. If RP is empty then continue to the next numbered rule.
- 3226 d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited  
 3227 RegistryEntryQuery over RP as explained in Section 8.2.3.
- 3228 2. If RP is empty, then raise the warning: *registry package query result is empty*; otherwise set  
 3229 RP to be the result of the RegistryPackageQuery.
- 3230 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 3231 within the RegistryResponse.

3232 **Examples**

3233 A client application wishes to identify all package instances in the Registry that contain an  
 3234 Invoice extrinsic object as a member of the package.

```

3235 <AdhocQueryRequest>
3236   <ResponseOption returnType = "LeafClass"/>
3237   <FilterQuery>
3238     <RegistryPackageQuery>
3239       <RegistryEntryQuery>
3240         <RegistryEntryFilter>
3241           <Clause>
3242             <SimpleClause leftArgument = "objectType">
3243               <StringClause stringPredicate = "Equal">Invoice</StringClause>
3244             </SimpleClause>
3245           </Clause>
3246         </RegistryEntryFilter>
3247       </RegistryEntryQuery>
3248     </RegistryPackageQuery>
3249   </FilterQuery>
3250 </AdhocQueryRequest>
3251
  
```

3253 A client application wishes to identify all package instances in the Registry that are not empty.

```
3254
3255 <AdhocQueryRequest>
3256   <ResponseOption returnType = "LeafClass"/>
3257   <FilterQuery>
3258     <RegistryPackageQuery>
3259       <RegistryObjectQuery/>
3260     </RegistryPackageQuery>
3261   </FilterQuery>
3262 </AdhocQueryRequest>
3263
```

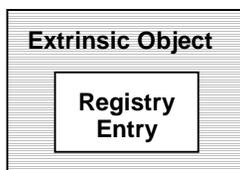
3264 A client application wishes to identify all package instances in the Registry that are empty. Since  
 3265 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate  
 3266 RegistryPackageQuery requests, one to find all packages and another to find all non-empty  
 3267 packages, and then do the set difference themselves. Alternatively, they could do a more  
 3268 complex RegistryEntryQuery and check that the packaging association between the package and  
 3269 its members is non-existent.

3270 Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by  
 3271 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an  
 3272 equivalent RegistryEntryQuery using appropriate "Source" and "Target" associations. However,  
 3273 the equivalent RegistryEntryQuery is often more complicated to write.

## 3274 8.2.10 ExtrinsicObjectQuery

### 3275 Purpose

3276 To identify a set of extrinsic object instances as the result of a query over selected registry  
 3277 metadata.



### 3278 ebRIM Binding

3279 **Figure 45: ebRIM Binding for ExtrinsicObjectQuery**

### 3280 Definition

```
3281
3282 <complexType name="ExtrinsicObjectQueryType">
3283   <complexContent>
3284     <extension base="tns:RegistryEntryQueryType">
3285       <sequence>
3286         <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
3287       </sequence>
3288     </extension>
3289   </complexContent>
3290 </complexType>
3291 <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
3292
```

```
3293 <element name="ExtrinsicObjectQueryResult">
3294   <complexType>
3295     <choice minOccurs="0" maxOccurs="unbounded">
3296       <element ref="rim:ObjectRef" />
3297       <element ref="rim:RegistryEntry" />
3298       <element ref="rim:RegistryObject" />
3299       <element ref="rim:ExtrinsicObject" />
3300     </choice>
3301   </complexType>
3302 </element>
3303
```

### 3304 Semantic Rules

- 3305 1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The  
3306 following steps will eliminate instances in EO that do not satisfy the conditions of the  
3307 specified filters.
  - 3308 a) If EO is empty then continue to the next numbered rule.
  - 3309 b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an  
3310 extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from  
3311 EO. If EO is empty then continue to the next numbered rule.
  - 3312 c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited  
3313 RegistryEntryQuery over EO as explained in Section 8.2.3.
- 3314 2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set  
3315 EO to be the result of the ExtrinsicObjectQuery.
- 3316 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
3317 within the RegistryResponse.  
3318

3318

3319 **8.2.11 OrganizationQuery**3320 **Purpose**

3321 To identify a set of organization instances as the result of a query over selected registry  
3322 metadata.

3323 **ebRIM Binding**

3324

3325

3326

3327

3328

3329

3330

3331

3332

3333

3334

3335

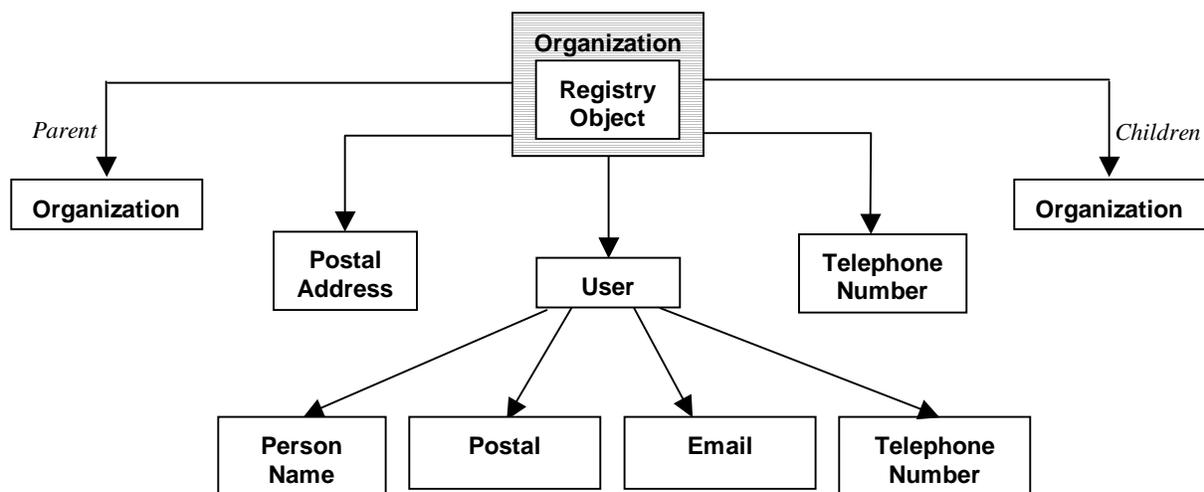


Figure 46: ebRIM Binding for OrganizationQuery

3336 **Definition**

3337

3338

3339

3340

3341

3342

3343

3344

3345

3346

3347

3348

3349

3350

3351

3352

3353

3354

3355

3356

3357

3358

3359

```

<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" />
    </choice>
  </complexType>
  
```

```

3360 <element ref="rim:RegistryObject" />
3361 <element ref="rim:Organization" />
3362 </choice>
3363 </complexType>
3364 </element>
3365

```

### 3366 Semantic Rules

- 3367 1. Let ORG denote the set of all persistent Organization instances in the Registry. The  
 3368 following steps will eliminate instances in ORG that do not satisfy the conditions of the  
 3369 specified filters.
- 3370 a) If ORG is empty then continue to the next numbered rule.
- 3371 b) If an OrganizationFilter element is not directly contained in the OrganizationQuery  
 3372 element, then go to the next step; otherwise let x be an organization instance in ORG. If x  
 3373 does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then  
 3374 continue to the next numbered rule.
- 3375 c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery  
 3376 element then go to the next step; otherwise, let x be an organization in ORG. If postal  
 3377 address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is  
 3378 empty then continue to the next numbered rule.
- 3379 d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery  
 3380 element then go to the next step; otherwise, let x be an organization in ORG. If any of the  
 3381 TelephoneNumberFilters isn't satisfied by all of the telephone numbers of x then remove  
 3382 x from ORG. If ORG is empty then continue to the next numbered rule.
- 3383 e) If a UserQuery is not specified then go to the next step; otherwise, let x be a remaining  
 3384 organization in ORG. If the defining primary contact of x does not satisfy the UserQuery,  
 3385 then remove x from ORG. If ORG is empty then continue to the next numbered rule.
- 3386 f) If a OrganizationParentBranch element is not specified within the OrganizationQuery,  
 3387 then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the  
 3388 following paragraph with o = x:  
 3389 Let o be an organization instance. If an OrganizationFilter is not specified within the  
 3390 OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the  
 3391 Organization hierarchy), then remove x from ORG; otherwise, let p be the parent  
 3392 organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.  
 3393 If ORG is empty then continue to the next numbered rule.  
 3394 If another OrganizationParentBranch element is directly contained within this  
 3395 OrganizationParentBranch element, then repeat the previous paragraph with o = p.
- 3396 g) If a OrganizationChildrenBranch element is not specified, then continue to the next  
 3397 numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent  
 3398 node of some organization instance, then remove x from ORG and if ORG is empty  
 3399 continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch  
 3400 element separately and execute the following paragraph with n = x.

- 3401 Let  $n$  be an organization instance. If an `OrganizationFilter` element is not specified within  
 3402 the `OrganizationChildrenBranch` element then let `ORGC` be the set of all organizations  
 3403 that have  $n$  as their parent node; otherwise, let `ORGC` be the set of all organizations that  
 3404 satisfy the `OrganizationFilter` and have  $n$  as their parent node. If `ORGC` is empty, then  
 3405 remove  $x$  from `ORG` and if `ORG` is empty continue to the next numbered rule; otherwise,  
 3406 let  $c$  be any member of `ORGC`. If a `PostalAddressFilter` element is directly contained in  
 3407 the `OrganizationChildrenBranch` and if the postal address of  $c$  does not satisfy the  
 3408 `PostalAddressFilter` then remove  $c$  from `ORGC`. If `ORGC` is empty then remove  $x$  from  
 3409 `ORG`. If `ORG` is empty then continue to the next numbered rule. If no  
 3410 `TelephoneNumberFilter` element is directly contained in the `OrganizationChildrenBranch`  
 3411 and if any of the `TelephoneNumberFilters` isn't satisfied by all of the telephone numbers  
 3412 of  $c$  then remove  $c$  from `ORGC`. If `ORGC` is empty then remove  $x$  from `ORG`. If `ORG` is  
 3413 empty then continue to the next numbered rule; otherwise, let  $y$  be an element of `ORGC`  
 3414 and continue with the next paragraph.
- 3415 If the `OrganizationChildrenBranch` element is terminal, i.e. if it does not directly contain  
 3416 another `OrganizationChildrenBranch` element, then continue to the next numbered rule;  
 3417 otherwise, repeat the previous paragraph with the new `OrganizationChildrenBranch`  
 3418 element and with  $n = y$ .
- 3419 h) Let `ORG` be the set of remaining `Organization` instances. Evaluate inherited  
 3420 `RegistryObjectQuery` over `ORG` as explained in Section 8.2.2.
- 3421 2. If `ORG` is empty, then raise the warning: *organization query result is empty*; otherwise set  
 3422 `ORG` to be the result of the `OrganizationQuery`.
- 3423 3. Return the result and any accumulated warnings or exceptions (in the `RegistryErrorList`)  
 3424 within the `RegistryResponse`.

### 3425 Examples

3426 A client application wishes to identify a set of organizations, based in France, that have  
 3427 submitted a `PartyProfile` extrinsic object this year.

```

3428 <AdhocQueryRequest>
3429   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
3430   <FilterQuery>
3431     <OrganizationQuery>
3432       <SourceAssociationBranch>
3433         <AssociationFilter>
3434           <Clause>
3435             <SimpleClause leftArgument = "associationType">
3436               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
3437             </SimpleClause>
3438           </Clause>
3439         </AssociationFilter>
3440       </OrganizationQuery>
3441     <RegistryObjectQuery>
3442       <RegistryObjectFilter>
3443         <Clause>
3444           <SimpleClause leftArgument = "objectType">
3445             <StringClause stringPredicate = "Equal">CPP</StringClause>
3446           </SimpleClause>
3447         </Clause>
3448       </RegistryObjectFilter>
3449     </AuditEventQuery>
  
```

```

3450         <AuditableEventFilter>
3451             <Clause>
3452                 <SimpleClause leftArgument = "timestamp">
3453                     <RationalClause logicalPredicate = "GE">
3454                         <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
3455                     </RationalClause>
3456                 </SimpleClause>
3457             </Clause>
3458         </AuditableEventFilter>
3459     </AuditableEventQuery>
3460 </RegistryObjectQuery>
3461 </SourceAssociationBranch>
3462 <PostalAddressFilter>
3463     <Clause>
3464         <SimpleClause leftArgument = "country">
3465             <StringClause stringPredicate = "Equal">France</StringClause>
3466         </SimpleClause>
3467     </Clause>
3468 </PostalAddressFilter>
3469 </OrganizationQuery>
3470 </FilterQuery>
3471 </AdhocQueryRequest>
3472

```

3473 A client application wishes to identify all organizations that have Corporation named XYZ as a  
3474 parent.

```

3475 <AdhocQueryRequest>
3476     <ResponseOption returnType = "LeafClass"/>
3477     <FilterQuery>
3478         <OrganizationQuery>
3479             <OrganizationParentBranch>
3480                 <NameBranch>
3481                     <LocalizedStringFilter>
3482                         <Clause>
3483                             <SimpleClause leftArgument = "value">
3484                                 <StringClause stringPredicate = "Equal">XYZ</StringClause>
3485                             </SimpleClause>
3486                         </Clause>
3487                     </LocalizedStringFilter>
3488                 </NameBranch>
3489             </OrganizationParentBranch>
3490         </OrganizationQuery>
3491     </FilterQuery>
3492 </AdhocQueryRequest>
3493
3494

```

## 3495 8.2.12 ServiceQuery

### 3496 Purpose

3497  
3498 To identify a set of service instances as the result of a query over selected registry metadata.

### 3499 ebRIM Binding

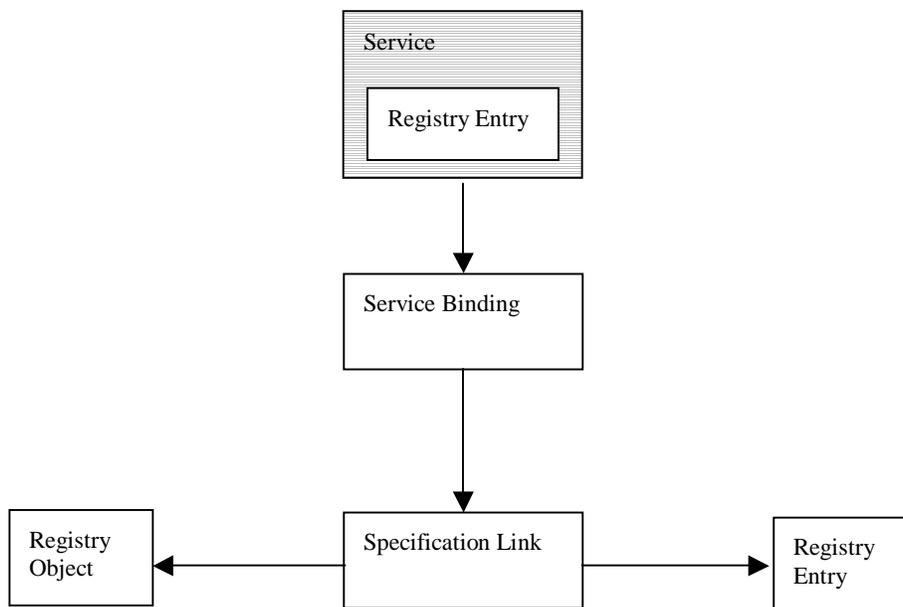


Figure 47: ebRIM Binding for ServiceQuery

3500

### 3501 Definition

```

3502 <complexType name="ServiceQueryType">
3503 <complexContent>
3504 <extension base="tns:RegistryEntryQueryType">
3505 <sequence>
3506 <element ref="tns:ServiceFilter" minOccurs="0"
3507 maxOccurs="1" />
3508 <element ref="tns:ServiceBindingBranch" minOccurs="0"
3509 maxOccurs="unbounded" />
3510 </sequence>
3511 </extension>
3512 </complexContent>
3513 </complexType>
3514 <element name="ServiceQuery" type="tns:ServiceQueryType" />
3515
3516 <element name="ServiceQueryResult">
3517 <complexType>
3518 <choice minOccurs="0" maxOccurs="unbounded">
3519 <element ref="rim:ObjectRef" />
3520 <element ref="rim:RegistryObject" />
3521 <element ref="rim:RegistryEntry" />
3522 <element ref="rim:Service" />
3523 </choice>
3524 </complexType>
3525 </element>
3526
3527
  
```

### 3528 Semantic Rules

- 3529 1. Let  $S$  denote the set of all persistent Service instances in the Registry. The following steps  
3530 will eliminate instances in  $S$  that do not satisfy the conditions of the specified filters.
- 3531 a) If  $S$  is empty then continue to the next numbered rule.

- 3532 b) If a ServiceFilter is not specified then go to the next step; otherwise, let x be a service in  
3533 S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then  
3534 continue to the next numbered rule.
- 3535 c) If a ServiceBindingBranch is not specified then continue to the next numbered rule;  
3536 otherwise, consider each ServiceBindingBranch element separately as follows:  
3537 Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the  
3538 member of SB. If a ServiceBindingFilter element is specified within the  
3539 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If  
3540 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.  
3541 If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then  
3542 continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch  
3543 element separately as follows:  
3544 Let sb be a remaining service binding in SB. Let SL be the set of all specification link  
3545 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
3546 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
3547 remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove  
3548 x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery  
3549 element is specified within the SpecificationLinkBranch then let sl be a remaining  
3550 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the  
3551 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a  
3552 specification link for some registry object in RO, then remove sl from SL. If SL is empty  
3553 then remove sb from SB. If SB is empty then remove x from S. If S is empty then  
3554 continue to the next numbered rule. If a RegistryEntryQuery element is specified within  
3555 the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat  
3556 RegistryEntryQuery element as follows: Let RE be the result set of the  
3557 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some  
3558 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If  
3559 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
- 3560 d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery  
3561 over AE as explained in Section 8.2.3.
- 3562 2. If S is empty, then raise the warning: *service query result is empty*; otherwise set S to be the  
3563 result of the ServiceQuery.
- 3564 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
3565 within the RegistryResponse.  
3566

3566

3567 **8.2.13 FederationQuery**3568 **Purpose**

3569 To identify a set of federation instances as the result of a query over selected registry metadata.

3570 **ebRIM Binding**

3571

3572

3573

3574

3575

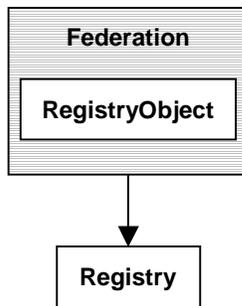
3576

3577

3578

3579

3580

3580 **Figure 48: ebRIM Binding for FederationQuery**3581 **Definition**

3582

3583 `<complexType name = "FederationQueryType">`3584 `<complexContent>`3585 `<extension base = "tns:RegistryEntryQueryType">`3586 `<sequence>`3587 `<element ref = "tns:FederationFilter" minOccurs = "0" maxOccurs="1"/>`3588 `<element ref = "tns:RegistryQuery" minOccurs = "0" maxOccurs = "unbounded"/>`3589 `</sequence>`3590 `</extension>`3591 `</complexContent>`3592 `</complexType>`3593 `<element name = "FederationQuery" type = "tns:FederationQueryType"/>`

3594

3595 `<element name = "FederationQueryResult">`3596 `<complexType>`3597 `<choice minOccurs = "0" maxOccurs = "unbounded">`3598 `<element ref = "rim:ObjectRef"/>`3599 `<element ref = "rim:RegistryObject"/>`3600 `<element ref = "rim:RegistryEntry"/>`3601 `<element ref = "rim:Federation"/>`3602 `</choice>`3603 `</complexType>`3604 `</element>`

3605

3606 **Semantic Rules**3607 1. Let F denote the set of all persistent Federation instances in the Registry. The following steps  
3608 will eliminate instances in F that do not satisfy the conditions of the specified filters.

3609 a) If F is empty then continue to the next numbered rule.

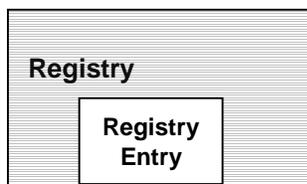
- 3610 b) If a FederationFilter is not specified, then continue to the next numbered rule; otherwise,  
 3611 let x be a federation instance in F. If x does not satisfy the FederationFilter then remove x  
 3612 from F. If F is empty then continue to the next numbered rule.
- 3613 c) If a RegistryQuery element is directly contained in the FederationQuery element then  
 3614 treat each RegistryQuery as follows: let R be the set of Registry instances returned by the  
 3615 RegistryQuery as defined in Section and let FR be the subset of R that are members of  
 3616 the federation x. If FR is empty, then remove x from F. If F is empty then continue to the  
 3617 next numbered rule.
- 3618 d) Let F be the set of remaining Federation instances. Evaluate inherited  
 3619 RegistryEntryQuery over F as explained in Section.
- 3620 2. If F is empty, then raise the warning: *federation query result is empty*; otherwise set F to be  
 3621 the result of the FederationQuery.
- 3622 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 3623 within the RegistryResponse.

## 3624 8.2.14 RegistryQuery

### 3625 Purpose

3626 To identify a set of registry instances as the result of a query over selected registry metadata.

### 3627 ebRIM Binding



3632 **Figure 49: ebRIM Binding for RegistryQuery**

### 3633 Definition

```

3634 <complexType name="RegistryQueryType">
3635   <complexContent>
3636     <extension base="tns:RegistryEntryQueryType">
3637       <sequence>
3638         <element ref="tns:RegistryFilter" minOccurs="0" maxOccurs="1" />
3639       </sequence>
3640     </extension>
3641   </complexContent>
3642 </complexType>
3643 <element name="RegistryQuery" type="tns:RegistryQueryType" />
3644 <element name="RegistryQueryResult">
3645   <complexType>
3646     <choice minOccurs="0" maxOccurs="unbounded">
3647       <element ref="rim:ObjectRef"/>
3648       <element ref="rim:RegistryObject"/>
3649       <element ref="rim:RegistryEntry"/>
3650       <element ref="rim:Registry"/>
3651     </choice>
3652   </complexType>

```

```

3653     </choice>
3654     </complexType>
3655 </element>
3656

```

### 3657 Semantic Rules

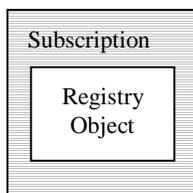
- 3658 1. Let R denote the set of all persistent Registry instances in the Registry. The following steps  
3659 will eliminate instances in R that do not satisfy the conditions of the specified filters.
- 3660 a) If R is empty then continue to the next numbered rule.
- 3661 b) If a RegistryFilter is not specified then go to the next step; otherwise, let x be a registry in  
3662 R. If x does not satisfy the RegistryFilter, then remove x from R. If R is empty then  
3663 continue to the next numbered rule.
- 3664 c) Let R be the set of remaining Registry instances. Evaluate inherited RegistryEntryQuery  
3665 over R as explained in Section.
- 3666 2. If R is empty, then raise the warning: *registry query result is empty*; otherwise, set R to be the  
3667 result of the RegistryQuery.
- 3668 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
3669 within the RegistryResponse.

## 3670 8.2.15 SubscriptionQuery

### 3671 Purpose

3672 To identify a set of subscription instances as the result of a query over selected registry metadata.

### 3673 ebRIM Binding



3674

3675 **Figure 50: ebRIM Binding for SubscriptionQuery**

### 3676 Definition

```

3677 <complexType name = "SubscriptionQueryType">
3678   <complexContent>
3679     <extension base = "tns:RegistryObjectQueryType">
3680       <sequence>
3681         <element ref = "tns:SubscriptionFilter" minOccurs = "0" maxOccurs = "1"/>
3682       </sequence>
3683     </extension>
3684   </complexContent>
3685 </complexType>
3686 <element name = "SubscriptionQuery" type = "tns:SubscriptionQueryType"/>
3687

```

```

3688
3689 <element name="SubscriptionQueryResult">
3690   <complexType>
3691     <choice minOccurs="0" maxOccurs="unbounded">
3692       <element ref="rim:ObjectRef" />
3693       <element ref="rim:RegistryObject" />
3694       <element ref="rim:Subscription" />
3695     </choice>
3696   </complexType>
3697 </element>
3698

```

### 3699 Semantic Rules

- 3700 1. Let S denote the set of all persistent Subscription instances in the Registry. The following
 3701 steps will eliminate instances in S that do not satisfy the conditions of the specified filters.
  - 3702 a) If S is empty then continue to the next numbered rule.
  - 3703 b) If a SubscriptionFilter element is not directly contained in the SubscriptionQuery
 3704 element, then go to the next step; otherwise let x be a subscription instance in S. If x does
 3705 not satisfy the SubscriptionFilter then remove x from S. If S is empty then continue to the
 3706 next numbered rule.
  - 3707 c) Let S be the set of remaining Subscription instances. Evaluate inherited
 3708 RegistryObjectQuery over S as explained in Section.
- 3709 2. If S is empty, then raise the warning: *subscription query result is empty*; otherwise, set S to
 3710 be the result of the SubscriptionQuery.
- 3711 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 3712 within the RegistryResponse.

3713

### 3714 8.2.16 UserQuery

#### 3715 Purpose

3716 To identify a set of user instances as the result of a query over selected registry metadata.

#### 3717 ebRIM Binding

3718

3719

3720

3721

3722

3723

3724

3725

3726

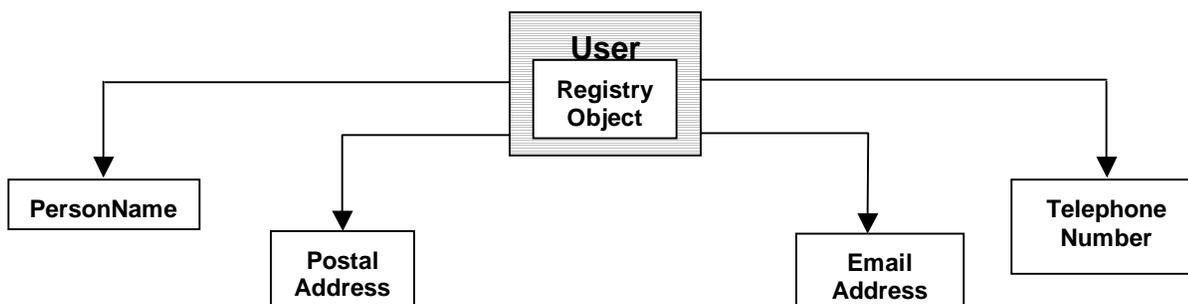


Figure 51: ebRIM Binding for OrganizationQuery Add PersonName under User and line up

3727 **Definition**

```

3728
3729 <complexType name = "UserQueryType">
3730   <complexContent>
3731     <extension base = "tns:RegistryObjectQueryType">
3732       <sequence>
3733         <element ref = "tns:UserFilter" minOccurs = "0" maxOccurs="1"/>
3734         <element ref = "tns:EmailAddressFilter" minOccurs = "0" maxOccurs="unbounded"/>
3735         <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
3736         <element ref = "tns:PersonNameFilter" minOccurs = "0" maxOccurs="1"/>
3737         <element ref = "tns:TelephoneFilter" minOccurs = "0" maxOccurs="unbounded"/>
3738       </sequence>
3739     </extension>
3740   </complexContent>
3741 </complexType >
3742 <element name = "UserQuery" type = "tns:UserQueryType"/>
3743
3744 <element name = "UserQueryResult">
3745   <complexType>
3746     <choice minOccurs = "0" maxOccurs = "unbounded">
3747       <element ref = "rim:ObjectRef"/>
3748       <element ref = "rim:RegistryObject"/>
3749       <element ref = "rim:User"/>
3750     </choice>
3751   </complexType>
3752 </element>
3753

```

3754 **Semantic Rules**

- 3755 1. Let U denote the set of all persistent User instances in the Registry. The following steps will  
3756 eliminate instances in U that do not satisfy the conditions of the specified filters.
- 3757 a) If U is empty then continue to the next numbered rule.
- 3758 b) If an UserFilter element is not directly contained in the UserQuery element, then go to the  
3759 next step; otherwise let x be an user instance in U. If x does not satisfy the UserFilter then  
3760 remove x from U. If U is empty then continue to the next numbered rule.
- 3761 c) If a EmailAddressFilter element is not directly contained in the UserQuery element then  
3762 go to the next step; otherwise, let x be an user in U. If email address of x does not satisfy  
3763 the EmailAddressFilter then remove x from U. If U is empty then continue to the next  
3764 numbered rule.
- 3765 d) If a PostalAddressFilter element is not directly contained in the UserQuery element then  
3766 go to the next step; otherwise, let x be an user in U. If postal address of x does not satisfy  
3767 the PostalAddressFilter then remove x from U. If U is empty then continue to the next  
3768 numbered rule.
- 3769 e) If a PersonNameFilter element is not directly contained in the UserQuery element then go  
3770 to the next step; otherwise, let x be an user in U. If reson name of x does not satisfy the  
3771 PersonNameFilter then remove x from U. If U is empty then continue to the next  
3772 numbered rule.

- 3773 f) If no TelephoneNumberFilter element is directly contained in the UserQuery element  
 3774 then go to the next step; otherwise, let x be an user in U. If any of the  
 3775 TelephoneNumberFilters isn't satisfied by all of the telephone numbers of x then remove  
 3776 x from U. If U is empty then continue to the next numbered rule.
- 3777 g) Let U be the set of remaining User instances. Evaluate inherited RegistryObjectQuery  
 3778 over U as explained in Section 8.2.2.
- 3779 2. If U is empty, then raise the warning: *user query result is empty*; otherwise set U to be the  
 3780 result of the UserQuery.
- 3781 Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within  
 3782 the RegistryResponse.

## 3783 8.2.17 Registry Filters

### 3784 Purpose

3785 To identify a subset of the set of all persistent instances of a given registry class.

### 3786 Definition

```

3787 <complexType name="FilterType">
3788   <sequence>
3789     <element ref="tns:Clause" />
3790   </sequence>
3791 </complexType>
3792 <element name="RegistryObjectFilter" type="tns:FilterType" />
3793 <element name="RegistryEntryFilter" type="tns:FilterType" />
3794 <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
3795 <element name="RegistryPackageFilter" type="tns:FilterType" />
3796 <element name="OrganizationFilter" type="tns:FilterType" />
3797 <element name="ClassificationNodeFilter" type="tns:FilterType" />
3798 <element name="AssociationFilter" type="tns:FilterType" />
3799 <element name="ClassificationFilter" type="tns:FilterType" />
3800 <element name="ClassificationSchemeFilter" type="tns:FilterType" />
3801 <element name="ExternalLinkFilter" type="tns:FilterType" />
3802 <element name="ExternalIdentifierFilter" type="tns:FilterType" />
3803 <element name="SlotFilter" type="tns:FilterType" />
3804 <element name="AuditableEventFilter" type="tns:FilterType" />
3805 <element name="UserFilter" type="tns:FilterType" />
3806 <element name="SlotValueFilter" type="tns:FilterType" />
3807 <element name="PostalAddressFilter" type="tns:FilterType" />
3808 <element name="TelephoneNumberFilter" type="tns:FilterType" />
3809 <element name="EmailAddressFilter" type="tns:FilterType" />
3810 <element name="ServiceFilter" type="tns:FilterType" />
3811 <element name="ServiceBindingFilter" type="tns:FilterType" />
3812 <element name="SpecificationLinkFilter" type="tns:FilterType" />
3813 <element name="LocalizedStringFilter" type="tns:FilterType" />
3814 <element name="FederationFilter" type="tns:FilterType"/>
3815 <element name="PersonNameFilter" type="tns:FilterType"/>
3816 <element name="RegistryFilter" type="tns:FilterType"/>
3817 <element name="SubscriptionFilter" type="tns:FilterType"/>
3818
3819

```

**3820 Semantic Rules**

- 3821 1. The Clause element is defined in Section 8.2.18.
- 3822 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing  
3823 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in  
3824 [ebRIM]. If not, raise exception: *registry object attribute error*. The RegistryObjectFilter  
3825 returns a set of identifiers for RegistryObject instances whose attribute values evaluate to  
3826 *True* for the Clause predicate.
- 3827 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing  
3828 SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in  
3829 [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter  
3830 returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*  
3831 for the Clause predicate.
- 3832 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing  
3833 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in  
3834 [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter  
3835 returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to  
3836 *True* for the Clause predicate.
- 3837 5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing  
3838 SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in  
3839 [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns  
3840 a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for  
3841 the Clause predicate.
- 3842 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing  
3843 SimpleClause shall identify a public attribute of the Organization or PostalAddress UML  
3844 classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The  
3845 OrganizationFilter returns a set of identifiers for Organization instances whose attribute  
3846 values evaluate to *True* for the Clause predicate.
- 3847 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any  
3848 containing SimpleClause shall identify a public attribute of the ClassificationNode UML  
3849 class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the  
3850 leftAttribute is the visible attribute “path” then if stringPredicate of the StringClause is not  
3851 “Equal” then raise exception: *classification node path attribute error*. The  
3852 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose  
3853 attribute values evaluate to *True* for the Clause predicate.
- 3854 8. For every AssociationFilter XML element, the leftArgument attribute of any containing  
3855 SimpleClause shall identify a public attribute of the Association UML class defined in  
3856 [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a  
3857 set of identifiers for Association instances whose attribute values evaluate to *True* for the  
3858 Clause predicate.
- 3859 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing  
3860 SimpleClause shall identify a public attribute of the Classification UML class defined in  
3861 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter  
3862 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*  
3863 for the Clause predicate.

- 3864 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any  
3865 containing SimpleClause shall identify a public attribute of the ClassificationNode UML  
3866 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The  
3867 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances  
3868 whose attribute values evaluate to *True* for the Clause predicate.
- 3869 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing  
3870 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in  
3871 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns  
3872 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the  
3873 Clause predicate.
- 3874 12. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing  
3875 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in  
3876 [ebRIM]. If not, raise exception: *external identifier attribute error*. The  
3877 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose  
3878 attribute values evaluate to *True* for the Clause predicate.
- 3879 13. For every SlotFilter XML element, the leftArgument attribute of any containing  
3880 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If  
3881 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot  
3882 instances whose attribute values evaluate to *True* for the Clause predicate.
- 3883 14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing  
3884 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in  
3885 [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter  
3886 returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to  
3887 *True* for the Clause predicate.
- 3888 15. For every UserFilter XML element, the leftArgument attribute of any containing  
3889 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If  
3890 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User  
3891 instances whose attribute values evaluate to *True* for the Clause predicate.
- 3892 16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one  
3893 SlotValue instance for each “value” in the “values” list of a Slot instance. The visible  
3894 attribute of SlotValue is “value”. It is a character string. The dynamic instances of SlotValue  
3895 are derived from the “values” attribute defined in ebRIM for a Slot instance. For every  
3896 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause  
3897 shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception:  
3898 *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value”  
3899 attribute evaluates to *True* for the Clause predicate.
- 3900 17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing  
3901 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in  
3902 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter  
3903 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*  
3904 for the Clause predicate.

- 3905 18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any  
3906 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML  
3907 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.  
3908 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances  
3909 whose attribute values evaluate to *True* for the Clause predicate.
- 3910 19. For every EmailAddressFilter XML element, the leftArgument attribute of any containing  
3911 SimpleClause shall identify a public attribute of the EmailAddress UML class defined in  
3912 [ebRIM]. If not, raise exception: *email address attribute error*. The EmailAddressFilter  
3913 returns a set of identifiers for EmailAddress instances whose attribute values evaluate to  
3914 *True* for the Clause predicate.
- 3915 20. For every ServiceFilter XML element, the leftArgument attribute of any containing  
3916 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].  
3917 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for  
3918 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 3919 21. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing  
3920 SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in  
3921 [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter  
3922 returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to  
3923 *True* for the Clause predicate.
- 3924 22. For every SpecificationLinkFilter XML element, the leftArgument attribute of any  
3925 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class  
3926 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The  
3927 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose  
3928 attribute values evaluate to *True* for the Clause predicate.
- 3929 23. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing  
3930 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in  
3931 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter  
3932 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to  
3933 *True* for the Clause predicate.
- 3934 24. For every FederationFilter XML element, the leftArgument attribute of any containing  
3935 SimpleClause shall identify a public attribute of the Federation UML class defined in  
3936 [ebRIM]. If not, raise exception: *federation attribute error*. The FederationFilter returns a set  
3937 of identifiers for Federation instances whose attribute values evaluate to *True* for the Clause  
3938 predicate.
- 3939 25. For every PersonNameFilter XML element, the leftArgument attribute of any containing  
3940 SimpleClause shall identify a public attribute of the PersonName UML class defined in  
3941 [ebRIM]. If not, raise exception: *person name attribute error*. The PersonNameFilter returns  
3942 a set of identifiers for PersonName instances whose attribute values evaluate to *True* for the  
3943 Clause predicate.
- 3944 26. For every RegistryFilter XML element, the leftArgument attribute of any containing  
3945 SimpleClause shall identify a public attribute of the Registry UML class defined in [ebRIM].  
3946 If not, raise exception: *registry attribute error*. The RegistryFilter returns a set of identifiers  
3947 for Registry instances whose attribute values evaluate to *True* for the Clause predicate.

3948 27. For every SubscriptionFilter XML element, the leftArgument attribute of any containing  
 3949 SimpleClause shall identify a public attribute of the Subscription UML class defined in  
 3950 [ebRIM]. If not, raise exception: *subscription attribute error*. The SubscriptionFilter returns a  
 3951 set of identifiers for Subscription instances whose attribute values evaluate to *True* for the  
 3952 Clause predicate.

3953 **8.2.18 XML Clause Constraint Representation**

3954 **Purpose**

3955 The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses.  
 3956 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to  
 3957 simply as Clauses in this specification.

3958 **Conceptual Diagram**

3959 The following is a conceptual diagram outlining the Clause structure.

3960

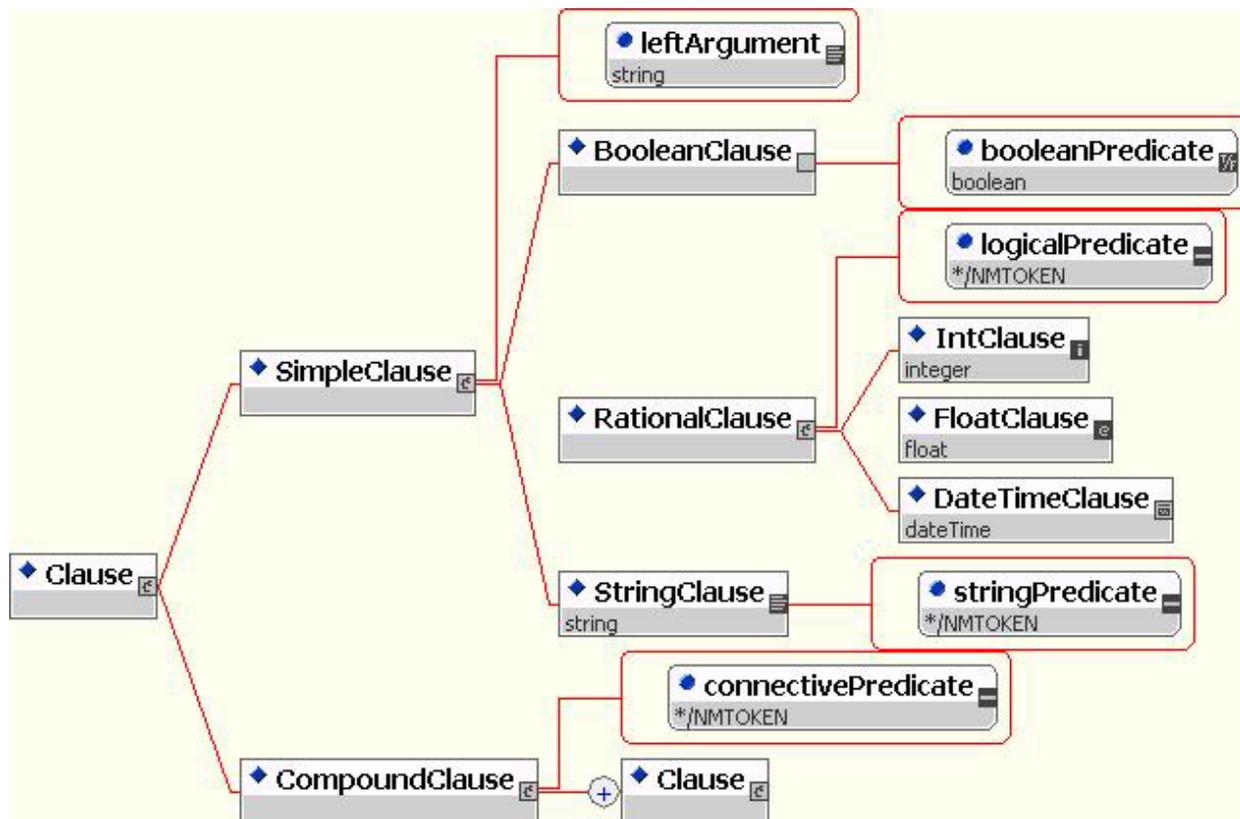


Figure 52: The Clause Structure

3961  
3962

3963 **Semantic Rules**

3964 Predicates and Arguments are combined into a "LeftArgument - Predicate - RightArgument"  
 3965 format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.

3966 SimpleClauses

3967 A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the

3968 Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.  
3969 SimpleClause is extended to support BooleanClause, StringClause, and RationalClause  
3970 (abstract).

3971

3972 **BooleanClause**

3973 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a  
3974 boolean.

3975

3976 **StringClauses**

3977 StringClause defines the predicate as an enumerated attribute of appropriate string-compare  
3978 operations and a right argument as the element's text data. String compare operations are defined  
3979 as follow:

- 3980       • Contains: Evaluates to true if left argument contains the content of the StringClause.  
3981       Evaluates to false otherwise.
- 3982       • NotContains: Evaluates to true if left argument does not contain the content of the  
3983       StringClause. Evaluates to false otherwise.
- 3984       • StartsWith: Evaluates to true if left argument starts with the content of the StringClause.  
3985       Evaluates to false otherwise.
- 3986       • NotStartsWith: Evaluates to true if left argument does not start with the content of the  
3987       StringClause. Evaluates to false otherwise.
- 3988       • Like: Evaluates to true if left argument matches the pattern specified by the content of the  
3989       StringClause. Evaluates to false otherwise. The pattern for the Like operation is a sub-  
3990       set of the LIKE syntax in SQL-92. The '\*' or '%' character matches any number of  
3991       characters while the '?' or '\_' character matches a single character.
- 3992       • NotLike: Evaluates to true if left argument does not match the pattern specified by the  
3993       content of the StringClause. Evaluates to false otherwise.
- 3994       • Equal: Evaluates to true if left argument is lexically equal to the content of the  
3995       StringClause. Evaluates to false otherwise.
- 3996       • NotEqual: Evaluates to true if left argument is lexically not equal to the content of the  
3997       StringClause. Evaluates to false otherwise.
- 3998       • EndsWith: Evaluates to true if left argument ends with the content of the StringClause.  
3999       Evaluates to false otherwise.
- 4000       • NotEndsWith: Evaluates to true if left argument does not end with the content of the  
4001       StringClause. Evaluates to false otherwise.

4002

4003 **RationalClauses**

4004 Rational number support is provided through a common RationalClause providing an  
4005 enumeration of appropriate rational number compare operations, which is further extended to  
4006 IntClause and FloatClause, each with appropriate signatures for the right argument.

4007

4008 **CompoundClauses**

4009 A CompoundClause contains two or more Clauses (Simple or Compound) and a connective  
 4010 predicate. This provides for arbitrarily complex Clauses to be formed.

4011 **Definition**

```
4012 <element name = "Clause">
```

```
4013   <annotation>
```

```
4014     <documentation xml:lang = "en">
```

4015 The following lines define the XML syntax for Clause.

```
4016   </documentation>
```

```
4017 </annotation>
```

```
4018 <complexType>
```

```
4019   <choice>
```

```
4020     <element ref = "tns:SimpleClause"/>
```

```
4021     <element ref = "tns:CompoundClause"/>
```

```
4022   </choice>
```

```
4023 </complexType>
```

```
4024 </element>
```

```
4025 <element name = "SimpleClause">
```

```
4026   <complexType>
```

```
4027   <choice>
```

```
4028     <element ref = "tns:BooleanClause"/>
```

```
4029     <element ref = "tns:RationalClause"/>
```

```
4030     <element ref = "tns:StringClause"/>
```

```
4031   </choice>
```

```
4032   <attribute name = "leftArgument" use = "required" type = "string"/>
```

```
4033 </complexType>
```

```
4034 </element>
```

```
4035 <element name = "CompoundClause">
```

```
4036   <complexType>
```

```
4037   <sequence>
```

```
4038     <element ref = "tns:Clause" maxOccurs = "unbounded"/>
```

```
4039   </sequence>
```

```
4040   <attribute name = "connectivePredicate" use = "required">
```

```
4041     <simpleType>
```

```
4042       <restriction base = "NMTOKEN">
```

```
4043         <enumeration value = "And"/>
```

```
4044         <enumeration value = "Or"/>
```

```
4045       </restriction>
```

```
4046     </simpleType>
```

```
4047   </attribute>
```

```
4048 </complexType>
```

```
4049 </element>
```

```
4050 <element name = "BooleanClause">
```

```
4051   <complexType>
```

```
4052   <attribute name = "booleanPredicate" use = "required" type = "boolean"/>
```

```
4053 </complexType>
```

```
4054 </element>
```

```
4055 <element name = "RationalClause">
```

```
4056   <complexType>
```

```
4057   <choice>
```

```
4058     <element ref = "tns:IntClause"/>
```

```
4059     <element ref = "tns:FloatClause"/>
```

```
4060     <element ref = "tns:DateTimeClause"/>
```

```
4061   </choice>
```

```
4062   <attribute name = "logicalPredicate" use = "required">
```

```

4065     <simpleType>
4066         <restriction base = "NMTOKEN">
4067             <enumeration value = "LE"/>
4068             <enumeration value = "LT"/>
4069             <enumeration value = "GE"/>
4070             <enumeration value = "GT"/>
4071             <enumeration value = "EQ"/>
4072             <enumeration value = "NE"/>
4073         </restriction>
4074     </simpleType>
4075 </attribute>
4076 </complexType>
4077 </element>
4078 <element name = "IntClause" type = "integer"/>
4079 <element name = "FloatClause" type = "float"/>
4080 <element name = "DateTimeClause" type = "dateTime"/>
4081
4082 <element name = "StringClause">
4083     <complexType>
4084         <simpleContent>
4085             <extension base = "string">
4086                 <attribute name = "stringPredicate" use = "required">
4087                     <simpleType>
4088                         <restriction base = "NMTOKEN">
4089                             <enumeration value = "Contains"/>
4090                             <enumeration value = "NotContains"/>
4091                             <enumeration value = "StartsWith"/>
4092                             <enumeration value = "NotStartsWith"/>
4093                             <enumeration value = "Like"/>
4094                             <enumeration value = "Equal"/>
4095                             <enumeration value = "NotEqual"/>
4096                             <enumeration value = "EndsWith"/>
4097                             <enumeration value = "NotEndsWith"/>
4098                         </restriction>
4099                     </simpleType>
4100                 </attribute>
4101             </extension>
4102         </simpleContent>
4103     </complexType>
4104 </element>
4105

```

#### 4106 Examples

##### 4107 Simple BooleanClause: "Smoker" = True

```

4108
4109 <Clause>
4110     <SimpleClause leftArgument="Smoker">
4111         <BooleanClause booleanPredicate="True"/>
4112     </SimpleClause>
4113 </Clause>
4114
4115 <BooleanClause param="sqlQuerySupported" operation="Equals">
4116     <value>true</value>
4117 </BooleanClause>

```

##### 4118 Simple StringClause: "Smoker" contains "mo"

```

4119
4120 <Clause>
4121   <SimpleClause leftArgument = "Smoker">
4122     <StringClause stringPredicate = "Contains">mo</StringClause>
4123   </SimpleClause>
4124 </Clause>

```

4125 **Simple IntClause: "Age" >= 7**

```

4126
4127 <Clause>
4128   <SimpleClause leftArgument="Age">
4129     <RationalClause logicalPredicate="GE">
4130       <IntClause>7</IntClause>
4131     </RationalClause>
4132   </SimpleClause>
4133 </Clause>
4134

```

4135 **Simple FloatClause: "Size" = 4.3**

```

4136
4137 <Clause>
4138   <SimpleClause leftArgument="Size">
4139     <RationalClause logicalPredicate="Equal">
4140       <FloatClause>4.3</FloatClause>
4141     </RationalClause>
4142   </SimpleClause>
4143 </Clause>
4144

```

4145 **Compound with two Simples ("Smoker" = False)AND("Age" =< 45))**

```

4146
4147 <Clause>
4148   <CompoundClause connectivePredicate="And">
4149     <Clause>
4150       <SimpleClause leftArgument="Smoker">
4151         <BooleanClause booleanPredicate="False"/>
4152       </SimpleClause>
4153     </Clause>
4154     <Clause>
4155       <SimpleClause leftArgument="Age">
4156         <RationalClause logicalPredicate="LE">
4157           <IntClause>45</IntClause>
4158         </RationalClause>
4159       </SimpleClause>
4160     </Clause>
4161   </CompoundClause>
4162 </Clause>
4163

```

4164 **Coumpound with one Simple and one Compound**

4165 **( ("Smoker" = False)And(("Age" =< 45)Or("American"=True)) )**

```

4166
4167 <Clause>
4168   <CompoundClause connectivePredicate="And">
4169     <Clause>
4170       <SimpleClause leftArgument="Smoker">

```

```

4171     <BooleanClause booleanPredicate="False"/>
4172     </SimpleClause>
4173   </Clause>
4174   <Clause>
4175     <CompoundClause connectivePredicate="Or">
4176       <Clause>
4177         <SimpleClause leftArgument="Age">
4178           <RationalClause logicalPredicate="LE">
4179             <IntClause>45</IntClause>
4180           </RationalClause>
4181         </SimpleClause>
4182       </Clause>
4183       <Clause>
4184         <SimpleClause leftArgument="American">
4185           <BooleanClause booleanPredicate="True"/>
4186         </SimpleClause>
4187       </Clause>
4188     </CompoundClause>
4189   </Clause>
4190 </CompoundClause>
4191 <Clause>
4192

```

### 4193 8.3 SQL Query Support

4194 The Registry may optionally support an SQL based query capability that is designed for Registry  
 4195 clients that demand more advanced query capability. The optional SQLQuery element in the  
 4196 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query  
 4197 language.

4198 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of  
 4199 the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database  
 4200 Language SQL [SQL], extended to include <sql invoked routines> (also known as  
 4201 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined  
 4202 in template form in Appendix D.3. The syntax of the Registry query language is defined by the  
 4203 BNF grammar in D.1.

4204 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use  
 4205 relational databases in a Registry implementation.

#### 4206 8.3.1 SQL Query Syntax Binding To [ebRIM]

4207 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational  
 4208 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as  
 4209 described in the following sections.

##### 4210 Class Binding

4211 A subset of the class names defined in [ebRIM] map to table names that may be queried by an  
 4212 SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an  
 4213 SQL query.

4214 The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3  
 4215 is as follows:

- 4216 • Classes that have concrete instances are mapped to relational tables. In addition entity classes  
4217 (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- 4218 • The intermediate classes in the inheritance hierarchy, namely RegistryObject and  
4219 RegistryEntry, map to relational views.
- 4220 • The names of relational tables and views are the same as the corresponding [ebRIM] class  
4221 name. However, the name binding is case insensitive.
- 4222 • Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in  
4223 Appendix D.3 where the column definitions are based on a subset of attributes defined for  
4224 that class in [ebRIM]. The attributes that map to columns include the inherited attributes for  
4225 the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed  
4226 which column definitions.

4227 An SQLQuery against a table not defined in Appendix D.3 may raise an error condition:  
4228 InvalidQueryException.

4229 The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn  
4230 definitions.

#### 4231 **Primitive Attributes Binding**

4232 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same  
4233 way as column names in SQL. Again the exact attribute names are defined in the class  
4234 definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is  
4235 therefore valid for a query to contain attribute names that do not exactly match the case defined  
4236 in [ebRIM].

#### 4237 **Reference Attribute Binding**

4238 A few of the [ebRIM] class attributes are of type ObjectRef and are a reference to an instance of  
4239 a class defined by [ebRIM]. For example, the sourceObject attribute of the Association class  
4240 returns a reference to an instance of a RegistryObject.

4241 In such cases the reference maps to the id attribute for the referenced object. The name of the  
4242 resulting column is the same as the attribute name in [ebRIM] as defined by 0. The data type for  
4243 the column is VARCHAR(64) as defined in Appendix D.3.

4244 When a reference attribute value holds a null reference, it maps to a null value in the SQL  
4245 binding and may be tested with the <null specification> (“IS [NOT] NULL” syntax) as defined  
4246 by [SQL].

4247 Reference attribute binding is a special case of a primitive attribute mapping.

#### 4248 **Complex Attribute Binding**

4249 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of  
4250 a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type  
4251 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

4252 The SQL query schema does not map complex attributes as columns in the table for the class for  
4253 which the attribute is defined. Instead the complex attributes are mapped to columns in the table  
4254 for the domain class that represents the data type for the complex attribute (e.g.  
4255 TelephoneNumber). A column links the row in the domain table to the row in the parent table  
4256 (e.g. User). An additional column named 'attribute\_name' identifies the attribute name in the  
4257 parent class, in case there are multiple attributes with the same complex attribute type.

4258 This mapping also easily allows for attributes that are a collection of a complex type. For  
4259 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the  
4260 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier  
4261 and an attribute\_name.

#### 4262 **Binding of Methods Returning Collections**

4263 Several of the [ebRIM] classes define methods in addition to attributes, where these methods  
4264 return collections of references to instances of classes defined by [ebRIM]. For example, the  
4265 getPackages method of the RegistryObject class returns a Collection of references to instances of  
4266 Packages that the object is a member of.

4267 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in  
4268 Appendix D.3 such that these stored procedures return a collection of i.d attribute values. The  
4269 returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

4270 These stored procedures may be used as the right-hand-side of an SQL IN clause to test for  
4271 membership of an object in such collections of references.

#### 4272 **8.3.2 Semantic Constraints On Query Syntax**

4273 This section defines simplifying constraints on the query syntax that cannot be expressed in the  
4274 BNF for the query syntax. These constraints *must* be applied in the semantic analysis of the  
4275 query.

- 4276 1. Class names and attribute names must be processed in a case insensitive manner.
- 4277 2. The syntax used for stored procedure invocation must be consistent with the syntax of an  
4278 SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 4279 3. For this version of the specification, the SQL select column list consists of exactly one  
4280 column, and must always be t.i.d, where t is a table reference in the FROM clause.

#### 4281 **8.3.3 SQL Query Results**

4282 The result of an SQL query resolves to a collection of objects within the registry. It never  
4283 resolves to partial attributes. The objects related to the result set may be returned as an  
4284 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the returnType  
4285 attribute of the responseOption parameter specified by the client on the AdHocQueryRequest.  
4286 The entire result set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in  
4287 Section 8.1.

#### 4288 **8.3.4 Simple Metadata Based Queries**

4289 The simplest form of an SQL query is based upon metadata attributes specified for a single class

4290 within [ebRIM]. This section gives some examples of simple metadata based queries.

4291 For example, to retrieve the collection of ExtrinsicObjects whose name contains the word  
4292 ‘Acme’ and that have a version greater than 1.3, the following query must be submitted:

```
4293 SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND  
4294 eo.id = nm.parent AND  
4295 eo.majorVersion >= 1 AND  
4296 (eo.majorVersion >= 2 OR eo.minorVersion > 3);  
4297  
4298
```

4299 Note that the query syntax allows for conjugation of simpler predicates into more complex  
4300 queries as shown in the simple example above.

### 4301 8.3.5 RegistryObject Queries

4302 The schema for the SQL query defines a view called RegistryObject that allows doing a  
4303 polymorphic query against all RegistryObject instances regardless of their actual concrete type or  
4304 table name.

4305 The following example is similar to the example in Section 8.3.4 except that it is applied against  
4306 all RegistryObject instances rather than just ExtrinsicObject instances. The result set will include  
4307 id for all qualifying RegistryObject instances whose name contains the word ‘Acme’ and whose  
4308 description contains the word “bicycle”.

```
4309 SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND  
4310 d.value LIKE '%bicycle%' AND  
4311 ro.id = nm.parent AND ro.id = d.parent;  
4312  
4313
```

### 4314 8.3.6 RegistryEntry Queries

4315 The schema for the SQL query defines a view called RegistryEntry that allows doing a  
4316 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or  
4317 table name.

4318 The following example is the same as the example in Section 8.3.4 except that it is applied  
4319 against all RegistryEntry instances rather than just ExtrinsicObject instances. The result set will  
4320 include id for all qualifying RegistryEntry instances whose name contains the word ‘Acme’ and  
4321 that have a version greater than 1.3.

```
4322 SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND  
4323 re.id = nm.parent AND  
4324 re.majorVersion >= 1 AND  
4325 (re.majorVersion >= 2 OR re.minorVersion > 3);  
4326  
4327
```

### 4328 8.3.7 Classification Queries

4329 This section describes various classification related queries.

#### 4330 Identifying ClassificationNodes

4331 ClassificationNodes are identified by their “id” attribute, as are all objects in [ebRIM]. However,  
4332 they may also be identified by their a “path” attribute that specifies an XPATH expression [XPT]  
4333 from a root classification node to the specified classification node in the XML document that  
4334 would represent the ClassificationNode tree including the said ClassificationNode.

**4335 Retrieving ClassificationSchemes**

4336 The following query retrieves the collection of ClassificationSchemes :

```
4337
4338 SELECT scheme.id FROM ClassificationScheme scheme;
4339
```

4340 The above query returns all ClassificationSchemes. Note that the above query may also specify  
4341 additional predicates (e.g. name, description etc.) if desired.

**4342 Retrieving Children of Specified ClassificationNode**

4343 The following query retrieves the children of a ClassificationNode given the “id” attribute of that  
4344 node:

```
4345
4346 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
4347
```

4348 The above query returns all ClassificationNodes that have the node specified by <id> as their  
4349 parent attribute.

**4350 Retrieving Objects Classified By a ClassificationNode**

4351 The following query retrieves the collection of ExtrinsicObjects classified by specified  
4352 ClassificationNodes:

```
4353
4354 SELECT id FROM ExtrinsicObject
4355 WHERE
4356     id IN (SELECT classifiedObject FROM Classification
4357           WHERE
4358             classificationNode IN (SELECT id FROM ClassificationNode
4359                                   WHERE path = "/Geography/Asia/Japan'))
4360 AND
4361     id IN (SELECT classifiedObject FROM Classification
4362           WHERE
4363             classificationNode IN (SELECT id FROM ClassificationNode
4364                                   WHERE path = '/Industry/Automotive'))
4365
```

4366 The above query retrieves the collection of ExtrinsicObjects that are classified by the  
4367 Automotive Industry and the Japan Geography. Note that according to the semantics defined for  
4368 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by  
4369 descendents of the specified ClassificationNodes.

**4370 Retrieving Classifications That Classify an Object**

4371 The following query retrieves the collection of Classifications that classify a specified Object:

```
4372
4373 SELECT id FROM Classification c
4374 WHERE c.classifiedObject = <id>;
4375
```

**4376 8.3.8 Association Queries**

4377 This section describes various Association related queries.

**4378 Retrieving All Association With Specified Object As Its Source**

4379 The following query retrieves the collection of Associations that have the specified Object as its  
4380 source:

```
4381
```

4382  
4383

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

### 4384 Retrieving All Association With Specified Object As Its Target

4385 The following query retrieves the collection of Associations that have the specified Object as its  
4386 target:

4387  
4388  
4389

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

### 4390 Retrieving Associated Objects Based On Association Attributes

4391 The following query retrieves the collection of Associations that have specified Association  
4392 attributes:

4393 Select Associations that have the specified name.

4394  
4395  
4396

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

4397 Select Associations that have the specified association type, where association type is a string  
4398 containing the corresponding field name described in [ebRIM].

4399  
4400  
4401  
4402

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

### 4403 Complex Association Queries

4404 The various forms of Association queries may be combined into complex predicates. The  
4405 following query selects Associations that have a specific sourceObject, targetObject and  
4406 associationType:

4407  
4408  
4409  
4410  
4411  
4412

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

### 4413 8.3.9 Package Queries

4414 The following query retrieves all Packages that a specified RegistryObject belongs to:

4415  
4416  
4417

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

### 4418 Complex Package Queries

4419 The following query retrieves all Packages that a specified object belongs to, that are not  
4420 deprecated and where name contains "RosettaNet."

4421  
4422  
4423  
4424  
4425  
4426

```
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'
```

### 4427 8.3.10 ExternalLink Queries

4428 The following query retrieves all ExternalLinks that a specified ExtrinsicObject is linked to:

```
4429
4430 SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
4431
```

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

```
4432
4433 SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
4434
4435
```

### 4436 **Complex ExternalLink Queries**

4437 The following query retrieves all ExternalLinks that a specified ExtrinsicObject belongs to, that  
4438 contain the word 'legal' in their description and have a URL for their externalURI.  
4439

```
4440 SELECT id FROM ExternalLink WHERE
4441     id IN (RegistryObject_externalLinks(<id>)) AND
4442     description LIKE '%legal%' AND
4443     externalURI LIKE '%http://%'
4444
```

### 4445 **8.3.11 Audit Trail Queries**

4446 The following query retrieves all the AuditableEvents for a specified RegistryObject:

```
4447
4448 SELECT id FROM AuditableEvent WHERE registryObject = <id>
4449
```

### 4450 **8.3.12 Object Export Queries**

4451 The standard Ad hoc Query protocol may be used to export RegistryObjects from a registry.

#### 4452 **Export Objects Owned By Specified User**

4453 The following query retrieves all RegistryObjects for a specified User id:

```
4454
4455 SELECT * from RegistryObject ro, AuditableEvent ae, User u WHERE
4456     ae.user_ = <userId> AND
4457     ae.eventType = 'Created' AND
4458     ae.registryObject = ro.id
4459
```

#### 4460 **Export Objects Owned By Users Affiliated With Specified Organization**

4461 The following query retrieves all RegistryObjects that are owned by any User that is affiliated  
4462 with the Organization matching the specified name pattern:  
4463

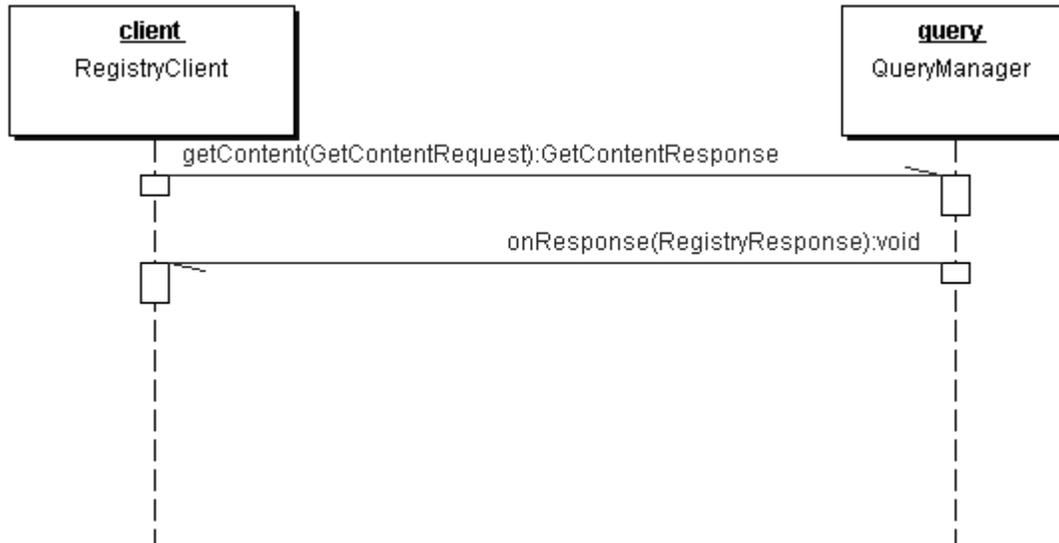
```
4464 SELECT * from RegistryObject ro, AuditableEvent ae, User u WHERE
4465     ae.user_ = u.id AND ae.eventType = 'Created' AND ae.registryObject = ro.id AND
4466     u.id IN (
4467         SELECT u1.id from User u1, Organization o, Name n WHERE
4468             n.value LIKE '%Sun%' AND u1.organization = o.id AND n.parent = o.id
4469     )
```

## 4470 **8.4 Content Retrieval**

4471 A client retrieves content via the Registry by sending the GetContentRequest to the  
4472 QueryManager. The GetContentRequest specifies a list of ObjectRefs for Objects that need to be  
4473 retrieved. The QueryManager returns the specified content by sending a GetContentResponse  
4474 message to the RegistryClient interface of the client.

4475 If there are no errors encountered, the GetContentResponse message includes the specified

4476 content(s) as mime multipart attachment(s) within the message.  
 4477 If there are errors encountered, the RegistryResponse payload includes the errors and there are  
 4478 no additional mime multipart attachments.



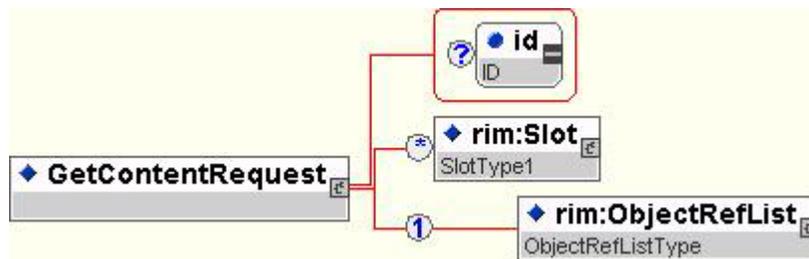
4479  
 4480

Figure 53: Content Retrieval Sequence Diagram

4481 **8.4.1 GetContentRequest**

4482 The GetContentRequest is used to retrieve repository item content from the registry.

4483 **Syntax:**



4484  
 4485

Figure 54: GetContentRequest Syntax

4486 **Parameters:**

- 4487     ▪ *ObjectRefList*: This parameter specifies a collection of ObjectRef elements that  
 4488         specify references to the ExtrinsicObjects whose corresponding repository items  
 4489         are being retrieved.

4490

4491 **Returns:**

4492 This request returns a GetContentResponse. See section 8.4.2 for details.

4493 **Exceptions:**

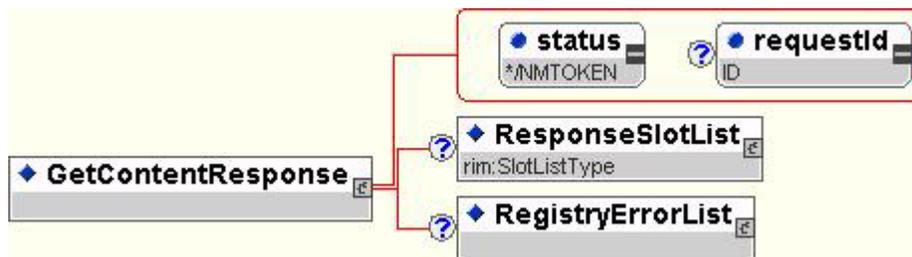
4494 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 4495       ▪ *ObjectNotFoundException*: signifies that one or more ObjectRef elements in the  
4496       ObjectRefList did not match any objects in the registry.

4497 **8.4.2 GetContentResponse**

4498 The GetContentResponse is sent by the registry as a response to GetContentRequest.

4499

4500 **Syntax:**

4501

4502

Figure 55: GetContentResponse Syntax

4503 **Parameters:**

4504 The GetContentResponse does not define any new parameters beyond those inherited by  
4505 RegistryResponseType as defined in 6.9.2.

4506 **8.4.3 Identification Of Content Payloads**

4507 Since the GetContentResponse message may include several repository items as additional  
4508 payloads, it is necessary to have a way to identify each mime multipart attachment in the  
4509 message. To facilitate this identification, the Registry must do the following:

- 4510 • Use the “id” attribute of the ExtrinsicObject instance as the value of the Content-ID header  
4511 parameter for the mime multipart that contains the corresponding repository item for the  
4512 ExtrinsicObject.
- 4513 • In case of [ebMS] transport, use the “id” attribute of the ExtrinsicObject instance in the  
4514 Reference element for that object in the Manifest element of the ebXMLHeader.

4515 **8.4.4 GetContentResponse Message Structure**

4516 The following message fragment illustrates the structure of the GetContentResponse Message  
4517 that is returning a Collection of Collaboration Protocol Profiles as a result of a  
4518 GetContentRequest that specified the “id” attributes for the requested objects. Note that the  
4519 boundary parameter in the Content-Type headers in the example below are meant to be  
4520 illustrative not prescriptive.

4521

4522 `Content-type: multipart/related; boundary="MIME_boundary"; type="text/xml";`

4523

4524 `--MIME_boundary`4525 `Content-ID: <GetContentRequest@example.com>`

```

4526 Content-Type: text/xml
4527
4528 <?xml version="1.0" encoding="UTF-8"?>
4529 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
4530   xmlns:eb= 'http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd' >
4531   <SOAP-ENV:Header>
4532
4533     <!--ebMS header goes here if using ebMS-->
4534     ...
4535
4536     <ds:Signature ...>
4537     <!--signature over soap envelope-->
4538     ...
4539   </ds:Signature>
4540
4541 </SOAP-ENV:Header>
4542
4543 <SOAP-ENV:Body>
4544
4545   <!--ebMS manifest goes here if using ebMS-->
4546   ...
4547
4548   <?xml version="1.0" encoding="UTF-8"?>
4549
4550   <GetContentResponse>
4551     <ObjectRefList>
4552       <ObjectRef id="urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff"/>
4553       <ObjectRef id="urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f"/>
4554     </ObjectRefList>
4555   </GetContentResponse>
4556
4557 </SOAP-ENV:Body>
4558 </SOAP-ENV:Envelope>
4559
4560 --MIME_boundary
4561
4562 Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4563 Content-Type: Multipart/Related; boundary=payload1_boundary; type=text/xml
4564 Content-Description: Optionally describe payload1 here
4565
4566 --payload1_boundary
4567 Content-Type: text/xml; charset=UTF-8
4568 Content-ID: signature:urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4569
4570 <ds:Signature ...>
4571   ... Signature for payload1
4572 </ds:Signature>
4573
4574 --payload1_boundary
4575 Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4576 Content-Type: text/xml
4577
4578 <?xml version="1.0" encoding="UTF-8"?>
4579 <tp:CollaborationProtocolProfile ...>
4580   .....
4581 </tp:CollaborationProtocolProfile>
4582 --payload1_boundary--
4583
4584 --MIME_boundary
4585
4586 Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
4587 Content-Type: Multipart/Related; boundary=payload2_boundary; type=text/xml
4588 Content-Description: Optionally describe payload2 here
4589
4590 --payload2_boundary
4591 Content-Type: text/xml; charset=UTF-8
4592 Content-ID: signature:urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
4593
4594 <ds:Signature ...>
4595   ... Signature for payload2
4596 </ds:Signature>
4597 --payload2_boundary
4598 Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f

```

```
4599 Content-Type: text/xml
4600
4601 <?xml version="1.0" encoding="UTF-8"?>
4602 <tp:CollaborationProtocolProfile ...>
4603     ....
4604 </tp:CollaborationProtocolProfile>
4605
4606 --payload2_boundary--
4607
4608
4609 --MIME_boundary--
4610
```

## 4611 9 Content Management Services

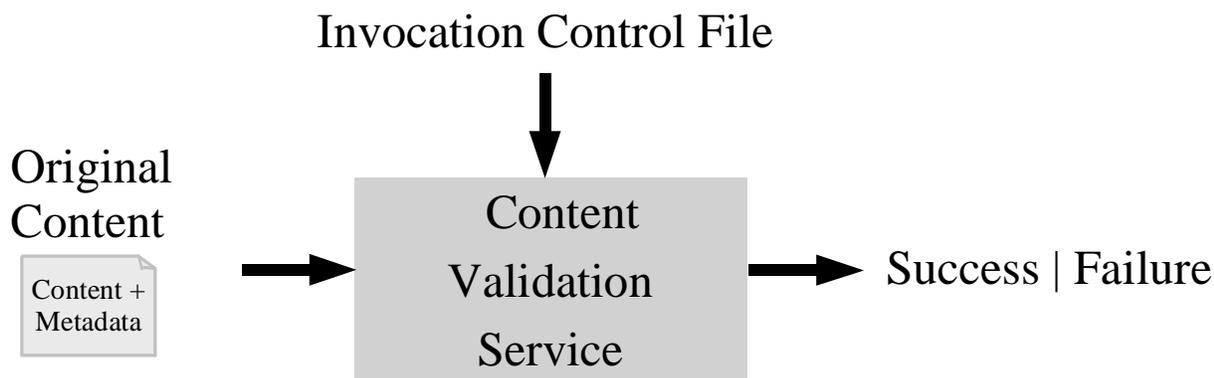
4612 This chapter describes the Content Management services of the ebXML Registry. Examples of  
 4613 Content Management Services include, but are not limited to, content validation and content  
 4614 cataloging. Content Management Services result in improved quality and integrity of registry  
 4615 content and metadata as well as improved ability for clients to discover that content and  
 4616 metadata.

4617 The Content Management Services facility of the registry is based upon a pluggable architecture  
 4618 that allows clients to publish and discover new Content Management Services as Service objects  
 4619 that conform to a normative web service interface specified in this chapter. Clients may define a  
 4620 Content Management Services that is specialized for managing a specific type of content.

4621 The Content Management Services facility as a whole is an optional normative feature of  
 4622 ebXML Registries compliant to version 3 or later of this specification. Note however that some  
 4623 aspects of the Content Management Services facility are required normative features of ebXML  
 4624 Registries.

### 4625 9.1 Content Validation

4626 The Content Validation feature provides the ability to enforce validation rules upon submitted  
 4627 content and metadata in a content specific manner.



4628  
4629

Figure 56: Content Validation Service

4630 A registry uses one or more Content Validation Services to automatically validate the  
 4631 RegistryObjects and repository items when they are submitted to the registry. A registry must  
 4632 reject a submission request in its entirety if it contains invalid data. In such cases a  
 4633 ValidationException must be returned to the client.

4634 Content Validation feature improves the quality of data in the registry.

#### 4635 9.1.1 Content Validation: Use Cases

4636 The following use cases illustrates use cases of the Content Validation feature:

##### 4637 Validation of HL7 Conformance Profiles

4638 The Healthcare Standards organization HL7 uses content validation to enforce consistency rules  
 4639 and semantic checks whenever an HL7 member submits an HL7 Conformance Profile. HL7 is  
 4640 also planning to use the feature to improve the quality of other types of HL7 artifacts.

#### 4641 **Validation of Business Processes**

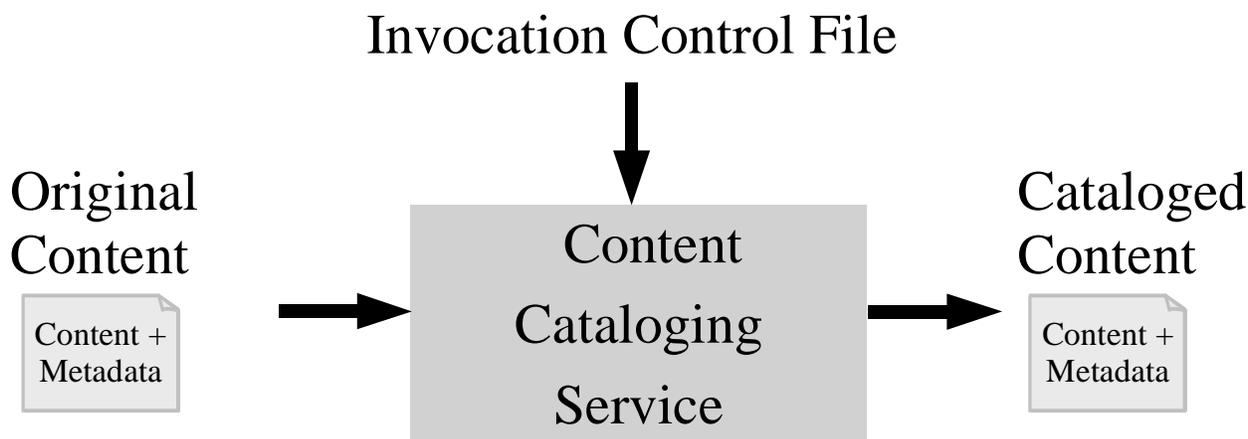
4642 Content validation may be used to enforce consistency rules and semantic checks whenever a  
4643 Business Process is submitted to the registry. This feature may be used by organizations such as  
4644 UN/CEFACT, OAG, and RosettaNet.

#### 4645 **Validation of UBL Business Documents**

4646 Content validation may be used by the UBL technical committee to enforce consistency rules  
4647 and semantic checks whenever a UBL business document is submitted to the registry.

## 4648 **9.2 Content Cataloging**

4649 The Content Cataloging feature provides the ability to selectively convert submitted  
4650 RegistryObject and repository items into metadata defined by [ebRIM], in a content specific  
4651 manner.



4652  
4653 **Figure 57: Content Cataloging Service**

4654 A registry uses one or more Content Cataloging Services to automatically catalog  
4655 RegistryObjects and repository items. Cataloging creates and/or updates RegistryObject  
4656 metadata such as ExtrinsicObject or Classification instances. The cataloged metadata enables  
4657 clients to discover the repository item based upon content from the repository item, using  
4658 standard query capabilities of the registry. This is referred to as *Content-based Discovery*.  
4659 The main benefit of the Content Cataloging feature is to enable Content-based Discovery.

### 4660 **9.2.1 Content-based Discovery: Use Cases**

4661 There are many scenarios where content-based discovery is necessary.

#### 4662 **Find All CPPs Where Role is “Buyer”**

4663 A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to  
4664 find CPPs for other companies where the Role element of the CPP is that of “Buyer”.

#### 4665 **Find All XML Schema’s That Use Specified Namespace**

4666 A client may wish to discover all XML Schema documents in the registry that use an XML  
4667 namespace containing the word “oasis”.

## 4668 Find All WSDL Descriptions with a SOAP Binding

4669 An ebXML registry client is attempting to discover all repository items that are WSDL  
4670 descriptions that have a SOAP binding defined. Note that SOAP binding related information is  
4671 content within the WSDL document and not metadata.

## 4672 9.3 Abstract Content Management Service

4673 This section describes in abstract terms how the registry supports pluggable, user-defined  
4674 Content Management Services. A Content Management Service is invoked in response to  
4675 content being submitted to the registry via the standard Submit/UpdateObjectsRequest method.  
4676 The Service invocation is on a per request basis where one request may result in many  
4677 invocations, one for each RegistryObject for which a Content Management Service is configured  
4678 within the registry.

4679 The registry may perform such invocation in one of two ways.

- 4680
- 4681 • *Inline Invocation Model:* Content Management Service may be invoked inline with the  
4682 processing of the Submit/UpdateObjectsRequest and prior to committing the content.  
4683 This is referred to as Inline Invocation Model.
- 4684 • *De-coupled Invocation Model:* Content Management Service may be invoked de-coupled  
4685 from the processing of the Submit/UpdateObjectsRequest and some time after  
4686 committing the content. This is referred to as De-coupled Invocation Model.
- 4687

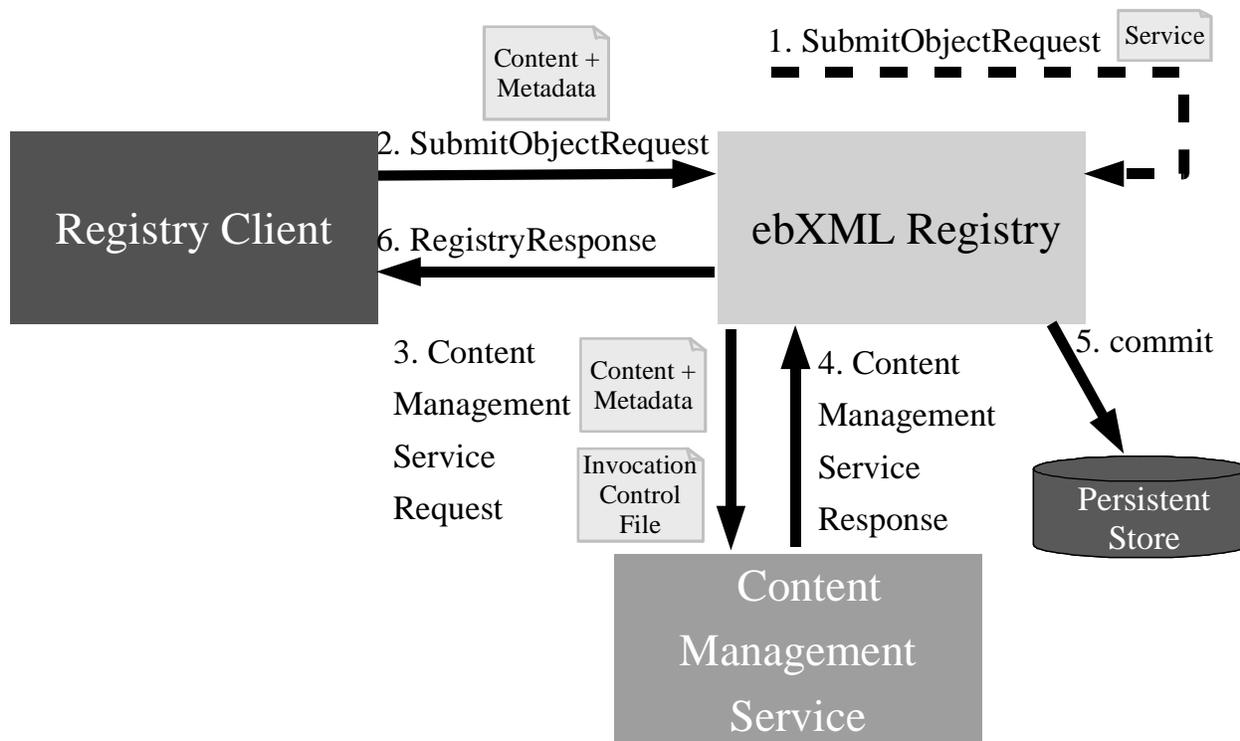
### 4688 9.3.1 Inline Invocation Model

4689 In an inline invocation model a registry must invoke a Content Management Service inline with  
4690 Submit/UpdateObjectsRequest processing and prior to committing the  
4691 Submit/UpdateObjectsRequest. All metadata and content from the original  
4692 Submit/UpdateObjectsRequest request or from the Content Management Service invocation  
4693 must be committed as an atomic transaction.

4694 Figure 58 shows an abstract Content Management Service and how it is used by an ebXML  
4695 Registry using an inline invocation model. The steps are as follows:

- 4696
- 4697 1. A client submits a Content Management Service S1 to an ebXML Registry. The client  
4698 typically belongs to an organization responsible for defining a specific type of content.  
4699 For example the client may belong to RosettaNet.org and submits a Content Validation  
4700 Service for validating RosettaNet PIPs. The client uses the standard  
4701 Submit/UpdateObjectsRequest interface to submit the Service. This is a one-time step to  
4702 configure this Content Management Service in the registry.
- 4703 2. Once the Content Management Service has been submitted, a potentially different client  
4704 may submit content to the registry that is of the same object type for which the Content  
4705 Management Service has been submitted. The client uses the standard  
4706 Submit/UpdateObjectsRequest interface to submit the content.
- 4707 3. The registry determines there is a Content Management Service S1 configured for the  
4708 object type for the content submitted. It invokes S1 using a  
4709 ContentManagementServiceRequest and passes it the content.

- 4710 4. The Content Management Service S1 processes the content and sends back a  
 4711 ContentManagementServiceResponse.  
 4712 5. The registry then commits the content to the registry if there are no errors encountered.  
 4713 6. The registry returns a RegistryResponse to the client for the  
 4714 Submit/UpdateObjectsRequest in step 2.  
 4715  
 4716  
 4717



4718  
 4719 **Figure 58: Content Management Service: Inline Invocation Model**

### 4720 9.3.2 De-coupled Invocation Model

4721 In a de-coupled invocation model a registry must invoke a Content Management Service  
 4722 independent of or de-coupled from the Submit/UpdateObjectsRequest processing. Any errors  
 4723 encountered during Content Management Service invocation must not have any impact on the  
 4724 original Submit/UpdateObjectsRequest processing.

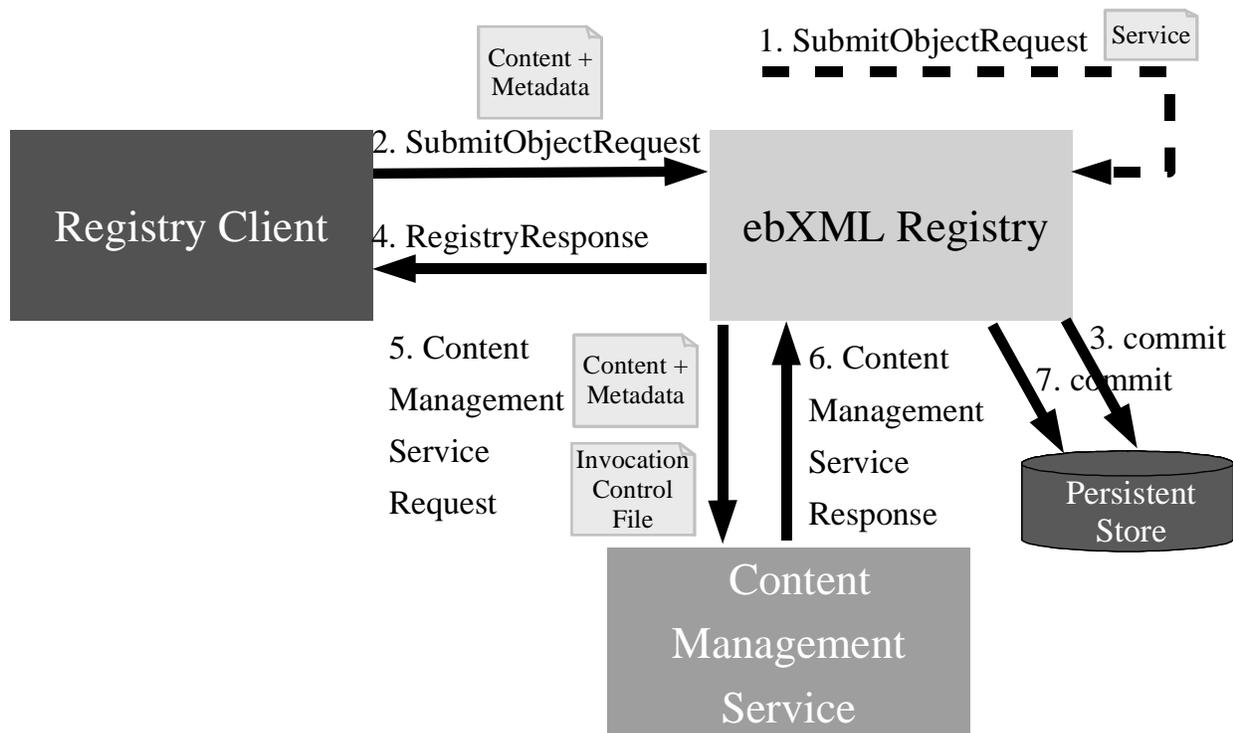
4725 All metadata and content from the original Submit/UpdateObjectsRequest request must be  
 4726 committed as an atomic transaction that is decoupled from the metadata and content that may be  
 4727 generated by the Content Management Service invocation.

4728 Figure 60 shows an abstract Content Management Service and how it is used by an ebXML  
 4729 Registry using a de-coupled invocation model. The steps are as follows:

4730

- 4731 1. Same as in inline invocation model (Content Management Service is submitted).  
 4732 2. Same as in inline invocation model (client submits content using  
 4733 Submit/UpdateObjectsRequest).

- 4734 3. The registry processes the Submit/UpdateObjectsRequest and commits it to persistent  
4735 store.
- 4736 4. The registry returns a RegistryResponse to the client for the  
4737 Submit/UpdateObjectsRequest in step 2.
- 4738 5. The registry determines there is a Content Management Service S1 configured for the  
4739 object type for the content submitted. It invokes S1 using a  
4740 ContentManagementServiceRequest and passes it the content.
- 4741 6. The Content Management Service S1 processes the content and sends back a  
4742 ContentManagementServiceResponse.
- 4743 7. If the ContentManagementServiceResponse includes any generated or modified content it  
4744 is committed to the persistent store as separate transaction. If there are any errors  
4745 encountered during de-coupled invocation of a Content Management Service then these  
4746 errors are logged by the registry in a registry specific manner and must not be reported  
4747 back to the client.  
4748



4749  
4750

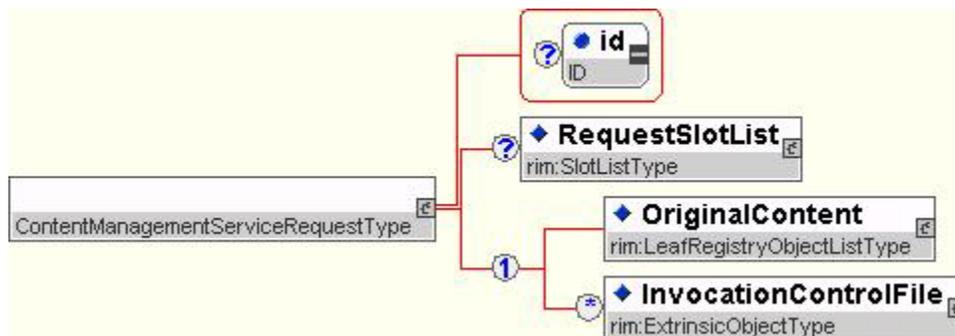
Figure 60: Content Management Service: De-coupled Invocation Model

## 4751 9.4 Content Management Service Protocol

4752 This section describe the abstract Content Management Service protocol that is the base-  
4753 protocol for other concrete protocols such as Validate Content protocol and Catalog Content  
4754 protocol. The concrete protocols will be defined later in this document.

### 4755 9.4.1 ContentManagementServiceRequestType

4756 The ContentManagementServiceRequestType must be the abstract base type for all requests sent  
4757 from a registry to a Content Management Service.

4758 **Syntax:**4759  
4760 **Figure 62: ContentManagementServiceRequestType Syntax**4761 **Parameters:**

4762 The following parameters are parameters that are either newly defined for this type or are  
4763 inherited and have additional semantics beyond those defined in the base type description.

- 4764
- 4765 ▪ *InvocationControlFile*: This parameter specifies the ExtrinsicObject for a  
4766 repository item that the caller wishes to specify as the Invocation Control File.  
4767 This specification does not specify the format of this file. There must be a  
4768 corresponding repository item as an attachment to this request. The corresponding  
4769 repository item should follow the same rules as attachments in  
Submit/UpdateObjectsRequest.
  - 4770 ▪ *OriginalContent*: This parameter specifies the RegistryObjects that will be  
4771 processed by the content management service. In case of ExtrinsicObject  
4772 instances within the OriginalContent there may be repository items present as  
4773 attachments to the ContentManagementServiceRequest. This specification does  
4774 not specify the format of such repository items. The repository items should  
4775 follow the same rules as attachments in Submit/UpdateObjectsRequest.

4776

4777 **Returns:**

4778 This request returns a ContentManagementServiceResponse. See section 9.4.2 for details.

4779 **Exceptions:**

4780 In addition to the exceptions returned by base request types, the following exceptions may be  
4781 returned:

- 4782
- 4783 ▪ *MissingRepositoryItemException*: signifies that the caller did not provide a  
repository item as an attachment to this request when the Service requires it.
  - 4784 ▪ *InvocationControlFileException*: signifies that the InvocationControlFile(s)  
4785 provided by the caller do not match the InvocationControlFile(s) expected by the  
4786 Service.
  - 4787 ▪ *UnsupportedContentException*: signifies that this Service does not support the  
4788 content provided by the caller.

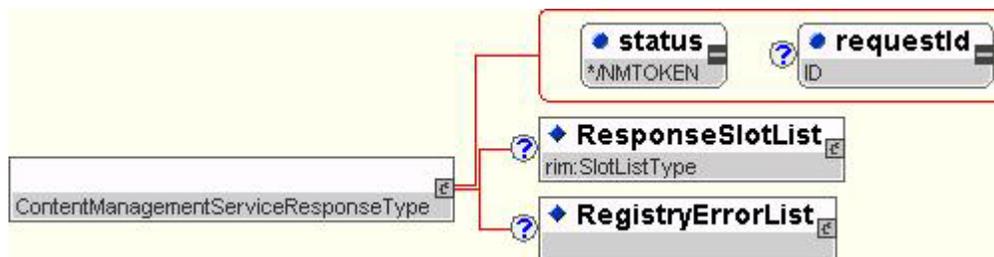
4789

## 4790 9.4.2 ContentManagementServiceResponseType

4791 The ContentManagementServiceResponseType is sent by a Content Management Service as a  
 4792 response to a ContentManagementServiceRequestType. The  
 4793 ContentManagementServiceResponseType is the abstract base type for all responses sent to a  
 4794 registry from a Content Management Service. It extends the RegistryResponseType and does not  
 4795 define any new parameters.

4796

### 4797 Syntax:



4798

4799

Figure 63: Content ContentManagementServiceResponseType Syntax

### 4800 Parameters:

4801 No new parameters are defined other than those inherited from RegistryResponseType.

4802

## 4803 9.5 Publishing / Configuration of a Content Management Service

4804 Any publisher may publish an arbitrary Content Management Service to an ebXML Registry.  
 4805 The Content Management Service must be published using the standard LifeCycleManager  
 4806 interface.

4807 The publisher must use the standard Submit/UpdateObjectsRequest to publish:

- 4808 ○ A Service instance for the Content Management Service. In Figure 64 this is exemplified  
 4809 by the defaultXMLCatalogingService in the upper-left corner. The Service instance must  
 4810 have an Association with a ClassificationNode in the canonical ObjectType  
 4811 ClassificationScheme as defined by [ebRIM]. The Service must be the sourceObject  
 4812 while a ClassificationNode must be the targetObject. This association binds the Service  
 4813 to that specific ObjectType. The associationType for this Association instance must be  
 4814 “ContentManagementServiceFor”. The Service must be classified by the canonical  
 4815 ContentManagementService ClassificationScheme as defined by [ebRIM]. For example it  
 4816 may be classified as a “ContentValidationService” or a “ContentCatalogingService”.
- 4817 ○ The Service instance may be classified by a ClassificationNode under the canonical  
 4818 InvocationModel ClassificationScheme as defined by [ebRIM], to determine whether it  
 4819 uses the Inline Invocation model or the De-coupled Invocation model.
- 4820 ○ The Service instance may be classified by a ClassificationNode under the canonical  
 4821 ErrorHandlerModel ClassificationScheme as defined by [ebRIM], to determine whether  
 4822 the Service should fail on first error or simply log the error as warning and continue. See  
 4823 9.6.4 section for details.

- 4824 ○ A ServiceBinding instance contained within the Service instance that must provide the
- 4825 accessURI to the Cataloging Service.
- 4826 ○ An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page
- 4827 describing:
- 4828 ○ The format of the supported content to be Cataloged
- 4829 ○ The format of the supported Invocation Control File
- 4830 Note that no SpecificationLink is required since this specification [ebRS] is implicit for
- 4831 Content Cataloging Services.
- 4832 ○ One or more Invocation Control File(s) that must be an ExtrinsicObject and a repository
- 4833 item pair. The ExtrinsicObject for the Invocation Control File must have a required
- 4834 Association with associationType of "InvocationControlFileFor". This is exemplified by
- 4835 the cppCatalogingServiceXSLT and the oagBODCatalogingServiceXSLT objects in
- 4836 Figure 64 (left side of picture). The Invocation Control File must be the sourceObject
- 4837 while a ClassificationNode in the canonical Object Type ClassificationScheme must be
- 4838 the targetObject.
- 4839

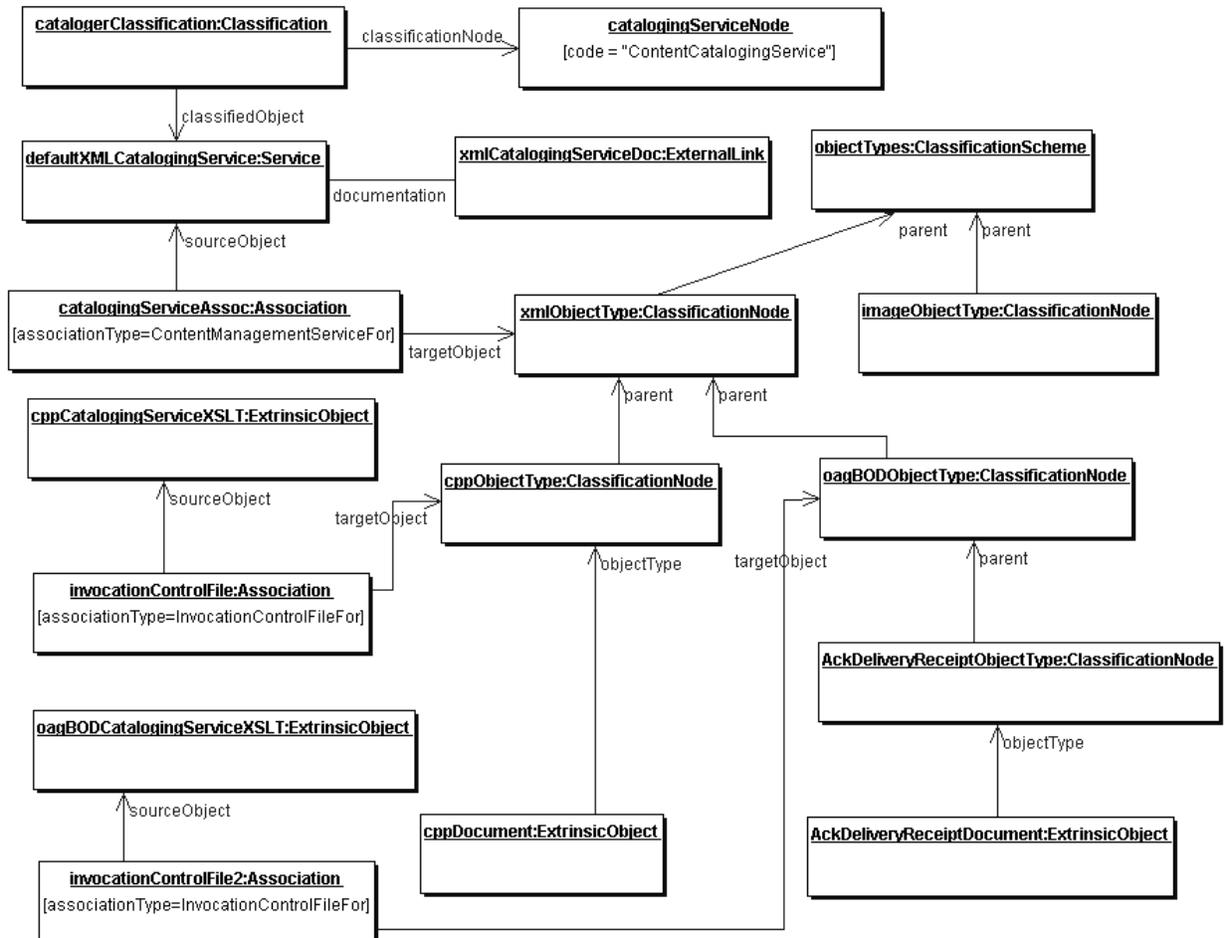


Figure 64: Cataloging Service Configuration

4840  
4841

4842 Figure 64 shows an example of the configuration of the default XML Cataloging Service  
4843 associated with the objectType for XML content. This Cataloging Service may be used with any  
4844 XML content that has its objectType attribute hold a reference to the xmlObjectType  
4845 ClassificationNode or one of its descendents.

4846 The figure also shows two different Invocation Control Files, cppCatalogingServiceXSLT and  
4847 oagBODCatalogingServiceXSLT that may be used to catalog ebXML CPP and OAG Business  
4848 Object Documents (BOD) respectively.

### 4849 **9.5.1 Multiple Content Management Services and Invocation Control Files**

4850 This specification allows clients to submit multiple Content Management Services of the same  
4851 type (e.g. validation, cataloging) and multiple Invocation Control Files for the same objectType.  
4852 Content Management Services of the same type of service for the same ObjectType are referred  
4853 to as peer Content Management Services.

4854  
4855 When there are multiple Content Management Services and Invocation Control Files for the  
4856 same ObjectType there must be an unambiguous association between a Content Management  
4857 Service and its Invocation Control File(s). This must be defined by an Association instance with  
4858 Association type of "InvocationControlFileFor" where the ExtrinsicObject for each Invocation  
4859 Control File is the sourceObject and the Service is the targetObject.

4860 The order of invocation of peer Content Management Services is undefined and may be  
4861 determined in a registry specific manner.

## 4862 **9.6 Invocation of a Content Management Service**

4863 This section describes how a registry invokes a Content Management Service.

### 4864 **9.6.1 Resolution Algorithm For Service and Invocation Control File**

4865 When a registry receives a submission of a RegistryObject, it must use the following algorithm  
4866 to determine or resolve the Content Management Services and Invocation Control Files to be  
4867 used for dynamic content management for the RegistryObject:

- 4868
- 4869 1. Get the objectType attribute of the RegistryObject.
  - 4870 2. Query to see if the ClassificationNode referenced by the objectType is the targetObject of  
4871 an Association with associationType of *ContentManagementServiceFor*. If desired  
4872 Association is not found for this ClassificationNode then repeat this step with its parent  
4873 ClassificationNode. Repeat until the desired Association is found or until the parent is the  
4874 ClassificationScheme. If desired Association(s) is found then repeat following steps for  
4875 each such Association instance.
  - 4876 3. Check if the sourceObject of the desired Association is a Service instance. If not, log an  
4877 InvalidConfigurationException. If it is a Service instance, then use this Service as the  
4878 Content Management service for the RegistryObject.

- 4879 4. Query to see if the objectType ClassificationNode is the targetObject of one or more  
4880 Association with associationType of *InvocationControlFileFor*. If desired Association is  
4881 not found for this ClassificationNode then repeat this step with its parent  
4882 ClassificationNode. Repeat until the desired Association is found or until the parent is the  
4883 ClassificationScheme.
- 4884 5. If desired Association(s) are found then check if the sourceObject of the desired  
4885 Association is an ExtrinsicObject instance. If not, log an InvalidConfigurationException.  
4886 If sourceObject is an ExtrinsicObject instance, then use its repository item as an  
4887 Invocation Control File. If there are multiple InvocationControlFiles then all of them  
4888 must be provided when invoking the Service.

4889 The above algorithm allows for objectType hierarchy to be used to configure Content  
4890 Management Services and Invocation Control Files with varying degrees of specificity or  
4891 specialization with respect to the type of content.

## 4892 **9.6.2 Audit Trail and Cataloged Content**

4893 The Cataloged Content generated as a result of the invocation of a Content Management Service  
4894 has an audit trail consistent with RegistryObject instances that are submitted by Registry Clients.  
4895 However, since a Registry Client does not submit Cataloged Content, the user attribute of the  
4896 AuditableEvent instances for such Cataloged Content references the Service object for the  
4897 Content Management Service that generated the Cataloged Content. This allows an efficient way  
4898 to distinguish Cataloged Content from content submitted by Registry Clients.

## 4899 **9.6.3 Referential Integrity**

4900 A registry must maintain referential integrity between the RegistryObjects and repository items  
4901 invocation of a Content Management Service.

## 4902 **9.6.4 Error Handling**

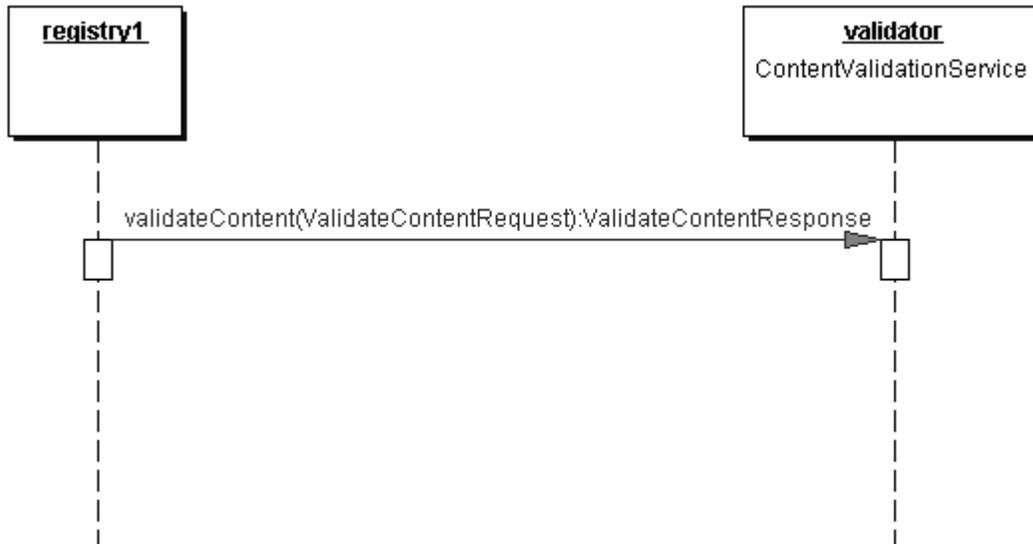
4903 If the Content Management Service is classified by the “FailOnError” ClassificationNode under  
4904 canonical ErrorHandlingModel ClassificationScheme as defined by [ebRIM], then the registry  
4905 must stop further processing of the Submit/UpdateObjectsRequest and return status of “Failure”  
4906 upon first error returned by a Content Management Service Invocation.

4907 If the Content Management Service is classified by the “LogErrorAndContinue”  
4908 ClassificationNode under ErrorHandlingModel then the registry must continue to process the  
4909 Submit/UpdateObjectsRequest and not let any Content Management Service invocation error to  
4910 affect the storing of the RegistryObjects and repository items that were submitted. Such errors  
4911 should be logged as Warnings within the RegistryResponse returned to the client. In this case a  
4912 registry must return a normal response with status = “Success” if the submitted content and  
4913 metadata is stored successfully even when there are errors encountered during dynamic  
4914 invocation of one or more Content Management Service.

## 4915 **9.7 Validate Content Protocol**

4916 The interface of a Content Validation Service must implement a single method called  
4917 validateContent. The validateContent method accepts a ValidateContentRequest as parameter  
4918 and returns a ValidateContentResponse as its response if there are no errors.

4919 The OriginalContent element within a ValidateContentRequest must contain exactly one  
 4920 RegistryObject that needs to be cataloged. The resulting ValidateContentResponse contains the  
 4921 status attribute that communicates whether the RegistryObject (and its content) are valid or not.  
 4922 The Validate Content protocol does not specify the implementation details of any specific  
 4923 Content Validation Service.



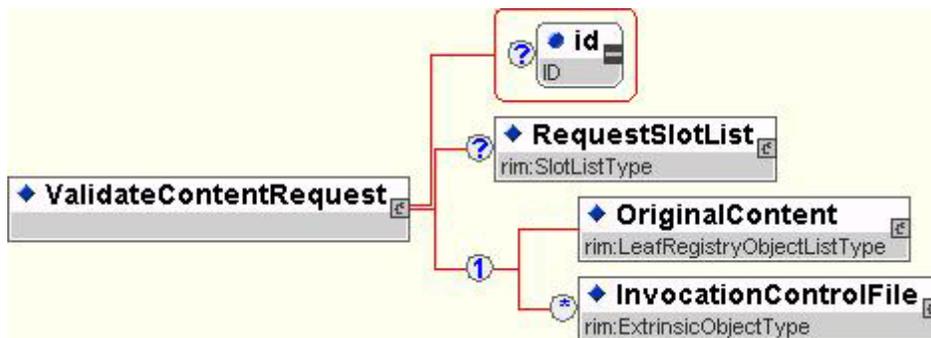
4924  
 4925

Figure 66: Validate Content Protocol

4926 **9.7.1 ValidateContentRequest**

4927 The ValidateContentRequest is used to pass content to a Content Validation Service so that it can  
 4928 validate the specified RegistryObject and any associated content. The RegistryObject typically is  
 4929 an ExternalLink (in case of external content) or an ExtrinsicObject. The ValidateContentRequest  
 4930 extends the base type ContentManagementServiceRequestType.

4931 **Syntax:**



4932  
 4933

Figure 67: ValidateContentRequest Syntax

4934 **Parameters:**

4935 The following parameters are parameters that are either newly defined for this type or are  
 4936 inherited and have additional semantics beyond those defined in the base type description.

- 4937 ▪ *InvocationControlFile*: Inherited from base type. This parameter may not be

4938 present. If present its format is defined by the Content Validation Service.

4939     ▪ *OriginalContent*: Inherited from base type. This parameter must contain exactly

4940 one RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an

4941 associated content. This specification does not specify the format of the content. If

4942 it is an ExtrinsicObject then there may be a corresponding repository item as an

4943 attachment to this request that is the content. The corresponding repository item

4944 should follow the same rules as attachments in Submit/UpdateObjectsRequest.

4945

4946 **Returns:**

4947 This request returns a ValidateContentResponse. See section 9.7.2 for details.

4948 **Exceptions:**

4949 In addition to the exceptions returned by base request types, the following exceptions may be

4950 returned:

4951     ▪ *InavlidContentException*: signifies that the specified content was found to be

4952 invalid. The exception should include enough detail for the client to be able to

4953 determine how to make the content valid.

4954

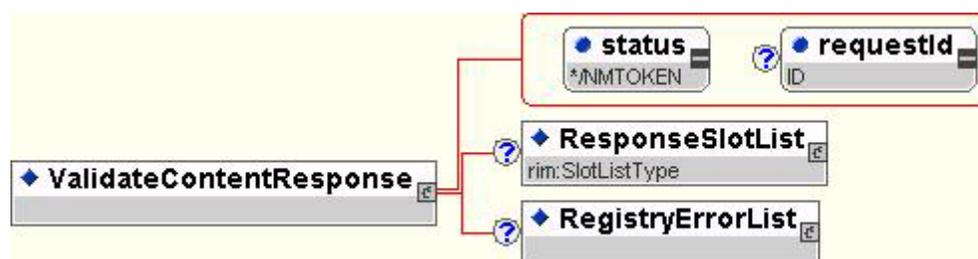
4955 **9.7.2 ValidateContentResponse**

4956 The ValidateContentResponse is sent by the Content Validation Service as a response to a

4957 ValidateContentRequest.

4958

4959 **Syntax:**



4960

4961

Figure 68: ValidateContentResponse Syntax

4962 **Parameters:**

4963 The following parameters are parameters that are either newly defined for this type or are

4964 inherited and have additional semantics beyond those defined in the base type description.

4965     ▪ *status*: Inherited attribute. This enumerated value is used to indicate the status of

4966 the request. Values for status are as follows:

4967

4968     • Success - This status specifies that the content specified in the

4969 ValidateContentRequest was valid.

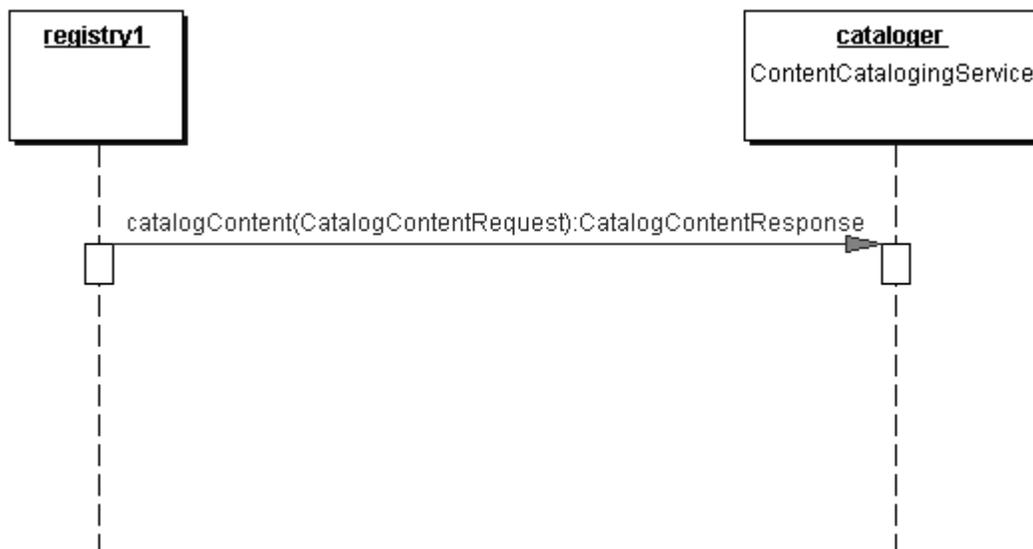
- 4970
- 4971
- 4972
- 4973
- 4974
- 4975
- Failure - This status specifies that the request failed. If the error returned is an `InvalidContentException` then the content specified in the `ValidateContentRequest` was invalid. If there was some other failure encountered during the processing of the request then a different error may be returned.

## 4976 9.8 Catalog Content Protocol

4977 The interface of the Content Cataloging Service must implement a single method called  
 4978 `catalogContent`. The `catalogContent` method accepts a `CatalogContentRequest` as parameter and  
 4979 returns a `CatalogContentResponse` as its response if there are no errors.

4980 The `CatalogContentRequest` may contain repository items that need to be cataloged. The  
 4981 resulting `CatalogContentResponse` contains the metadata and possibly content that gets generated  
 4982 or updated by the Content Cataloging Service as a result of cataloging the specified repository  
 4983 items.

4984 The Catalog Content protocol does not specify the implementation details of any specific  
 4985 Content Cataloging Service.



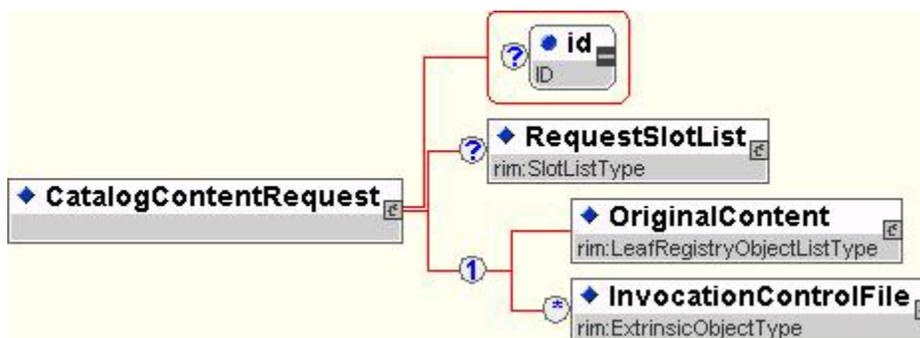
4986

4987

Figure 69: Catalog Content Protocol

### 4988 9.8.1 CatalogContentRequest

4989 The `CatalogContentRequest` is used to pass content to a Content Cataloging Service so that it can  
 4990 create catalog metadata for the specified `RegistryObject` and any associated content. The  
 4991 `RegistryObject` typically is an `ExternalLink` (in case of external content) or an `ExtrinsicObject`.  
 4992 The `CatalogContentRequest` extends the base type `ContentManagementServiceRequestType`.

4993 **Syntax:**4994 **Figure 70: CatalogContentRequest Syntax**4995 **Parameters:**

4996 The following parameters are parameters that are either newly defined for this type or are  
 4997 inherited and have additional semantics beyond those defined in the base type description.  
 4998

- 4999
- 5000 ▪ *InvocationControlFile*: Inherited from base type. If present its format is defined by the Content Cataloging Service.
  - 5001 ▪ *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.

5007

5008 **Returns:**

5009 This request returns a CatalogContentResponse. See section 9.8.2 for details.

5010 **Exceptions:**

5011 In addition to the exceptions returned by base request types, the following exceptions may be  
 5012 returned:

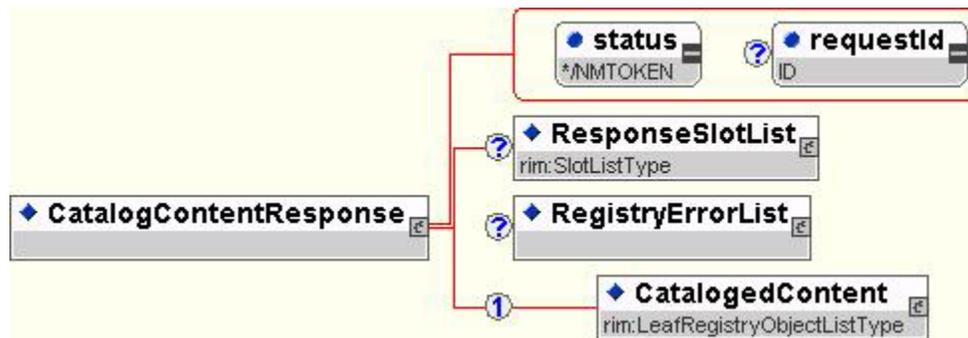
- 5013
- 5014 ▪ *CatalogingException*: signifies that an exception was encountered in the Cataloging algorithm for the service.

5015

5016 **9.8.2 CatalogContentResponse**

5017 The CatalogContentResponse is sent by the Content Cataloging Service as a response to a  
 5018 CatalogContentRequest.

5019

5020 **Syntax:**5021  
5022 **Figure 71: CatalogContentResponse Syntax**5023 **Parameters:**

5024 The following parameters are parameters that are either newly defined for this type or are  
 5025 inherited and have additional semantics beyond those defined in the base type description.

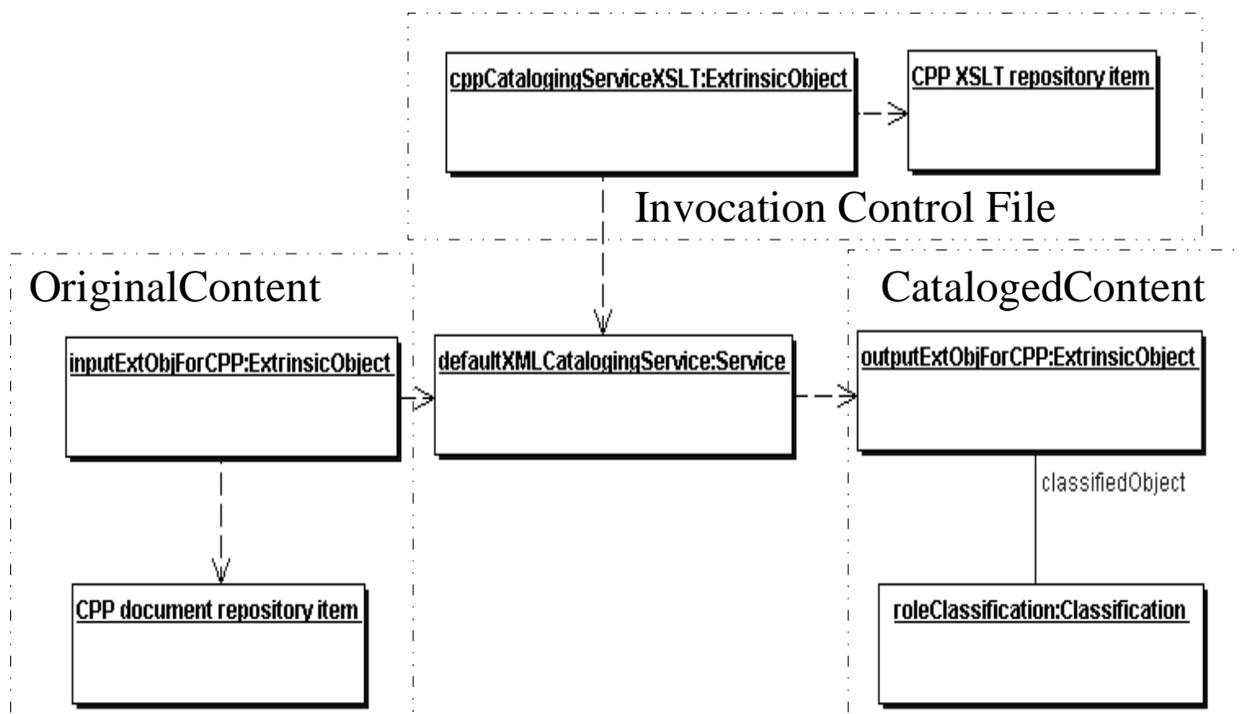
- 5026     ▪ *CatalogedContent*: This parameter specifies a collection of RegistryObject  
 5027 instances that were created or updated as a result of dynamic content cataloging  
 5028 by a content cataloging service. The Content Cataloging Service may add  
 5029 metadata such as Classifications, ExternalIdentifiers, name, description etc. to the  
 5030 CatalogedContent element. There *may* be an accompanying repository item as an  
 5031 attachment to this response message if the original repository item was modified  
 5032 by the request.

5033  
 5034

5035 **9.9 Illustrative Example: Default XML Cataloging Service**

5036 Figure 72 shows a UML instance diagram to illustrate how a Content Cataloging Service is used.  
 5037 This Content Cataloging Service is the normative Default XML Cataloging Service described in  
 5038 section 9.10.

- 5039     ○ In the center we see a Content Cataloging Service name defaultXMLCataloger Service.
- 5040     ○ On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP  
 5041 being input as Original Content to the defaultXMLCataloging Service.
- 5042     ○ On top we see an XSLT style sheet repository item and its ExtrinsicObject that is  
 5043 configured as an Invocation Control File for the defaultXMLCataloger Service.
- 5044     ○ On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for  
 5045 the CPP. We also see a Classification roleClassification, which classifies the CPP by the  
 5046 Role element within the CPP. These are the Cataloged Content generated as a result of  
 5047 the Cataloging Service cataloging the CPP.



5048  
5049 **Figure 72: Example of CPP cataloging using Default XML Cataloging Service**

5050

## 5051 9.10 Default XML Content Cataloging Service

5052 An ebXML Registry must provide the default XML Content Cataloging Service natively as a  
5053 built-in service with the following constraints:

- 5054 • There is exactly one Service instance for the Default XML Content Cataloging Service
- 5055 • The Service is an XSLT engine
- 5056 • The Service may be invoked with exactly one Invocation Control File
- 5057 • The Original Content for the Service must be XML document(s)
- 5058 • The Cataloged Content for the Service must be XML document(s)
- 5059 • The Invocation Control File must be an XSLT style sheet
- 5060 • Each invocation of the Service may be with different Invocation Control File (XSLT  
5061 style sheet) depending upon the objectType of the RegistryObject being cataloged. Each  
5062 objectType should have its own unique XSLT style sheet. For example, ebXML CPP  
5063 documents should have a specialized ebXML CPP Invocation Control XSLT style sheet.
- 5064 • The Service must have at least one input XML document that is a RegistryObject.  
5065 Typically this is an ExtrinsicObject or an ExternalLink.
- 5066 • The Service may have at most one additional input XML document that is the content  
5067 represented by the RegistryObject (e.g. a CPP document or an HL7 Conformance  
5068 Profile). The optional second input must be referenced within the XSLT Style sheet by a  
5069 using the “document” function with the document name specified by variable  
5070 “repositoryItem” as in “document(\$repositoryItem)”. A registry must define the variable  
5071 “repositoryItem” when invoking the default XML Cataloging Service.

- 5072       • The default XML Content Cataloging Service must apply the XSLT style sheet to the  
5073       input XML instance document(s) in an XSLT transformation to generate the Cataloged  
5074       Output.

5075       The Default XML Content Cataloging Service is a required normative feature of an ebXML  
5076       Registry.

### 5077       **9.10.1 Publishing of Default XML Content Cataloging Service**

5078       An ebXML Registry must provide the default XML Content Cataloging Service natively as a  
5079       built-in service. This built-in service must be published to the registry as part of the intrinsic  
5080       bootstrapping of required data within the registry.

## 5081 **10 Event Notification Service**

5082 This chapter defines the Event Notification feature of the OASIS ebXML Registry. The Event  
5083 Notification feature is an optional but normative feature of the ebXML Registry.

5084 Event Notification feature allows OASIS ebXML Registries to notify its users and / or other  
5085 registries about events of interest. It allows users to stay informed about registry events without  
5086 being forced to periodically poll the registry. It also allows a registry to propagate internal  
5087 changes to other registries whose content might be affected by those changes.

5088 ebXML registries support content-based Notification where interested parties express their  
5089 interest in form of a query. This is different from subject-based (sometimes referred to as topic-  
5090 based) notification, where information is categorized by subjects and interested parties express  
5091 their interests in those predefined subjects.

### 5092 **10.1 Use Cases**

5093 The following use cases illustrate different ways in which ebXML registries notify users or other  
5094 registries.

#### 5095 **10.1.1 CPP Has Changed**

5096 A user wishes to know when the CPP [ebCPP] of her partner is updated or superceded by  
5097 another CPP. When that happens she may wish to create a CPA [ebCPP] based upon the new  
5098 CPP.

#### 5099 **10.1.2 New Service is Offered**

5100 A user wishes to know when a new Plumbing service is offered in her town and be notified every  
5101 10 days. When that happens, she might try to learn more about that service and compare it with  
5102 her current Plumbing service provider's offering.

#### 5103 **10.1.3 Monitor Download of Content**

5104 User wishes to know whenever her CPP [ebCPP] is downloaded in order to evaluate on an  
5105 ongoing basis the success of her recent advertising campaign. She might also want to analyze  
5106 who the interested parties are.

#### 5107 **10.1.4 Monitor Price Changes**

5108 User wishes to know when the price of a product that she is interested in buying drops below a  
5109 certain amount. If she buys it she would also like to be notified when the product has been  
5110 shipped to her.

#### 5111 **10.1.5 Keep Replicas Consistent With Source Object**

5112 In order to improve performance and availability of accessing some registry objects, a local  
5113 registry may make replicas of certain objects that are hosted by another registry. The registry  
5114 would like to be notified when the source object for a replica is updated so that it can  
5115 synchronize the replica with the latest state of the source object.

## 5116 10.2 Registry Events

5117 Activities within a registry result in meaningful events. Typically, registry events are generated  
 5118 when a registry processes client requests. In addition, certain registry events may be caused by  
 5119 administrative actions performed by a registry operator. [ebRIM] defines the AuditableEvent  
 5120 class, instances of which represent registry events. When such an event occurs, an  
 5121 AuditableEvent instance is generated by the registry.

## 5122 10.3 Subscribing to Events

5123 A User may create a subscription with a registry if she wishes to receive notification for a  
 5124 specific type of event. A User creates a subscription by submitting a Subscription instance to a  
 5125 registry using the SubmitObjectsRequest. If a Subscription is submitted to a registry that does  
 5126 not support event notification then the registry must return an UnsupportedCapabilityException.  
 5127

5128 The listing below shows a sample Subscription that uses a pre-defined SQL query as its selector,  
 5129 that will result in an email notification to the user whenever a Service is created that is classified  
 5130 as a "Plumbing" service and located in "A Little Town".

5131

5132 The SQL query within the selector in plain English says the following:

5133 Find all Services that are Created AND classified by ClassificationNode  
 5134 where ClassificationNode's Path ends with string "Plumbing", AND classified  
 5135 by ClassificationNode where ClassificationNode's Code contains string "A  
 5136 Little Town".

5137

5138 <?xml version="1.0" encoding="UTF-8"?>

5139

5140 <tns:Subscription xmlns:tns="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5"  
 5141 xmlns:query="urn:oasis:names:tc:ebxml-regrep:query:xsd:2.5"  
 5142 xmlns:rim="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5"  
 5143 xmlns:rs="urn:oasis:names:tc:ebxml-regrep:rs:xsd:2.5"  
 5144 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 5145 xsi:schemaLocation="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.5 ../../../../ebxmlrr-  
 5146 spec/misc/schema/v3/rim.xsd urn:oasis:names:tc:ebxml-regrep:query:xsd:2.5  
 5147 ../../../../ebxmlrr-spec/misc/schema/v3/query.xsd " id="e3373a7b-4958-4e55-8820-  
 5148 d03a191fb76a" notificationInterval="P10D" selector="urn:uuid:8ea2c0f5-b14a-4f46-  
 5149 88b2-f69e35405d86">

5150

5151 <!-- The selector is a reference to a query object that has the following query defined  
 5152 SELECT \* FROM Service s, AuditableEvent e, AffectectedObject ao,  
 5153 Classification c1, Classification c2  
 5154 ClassificationNode cn1, ClassificationNode cn2 WHERE  
 5155 e.eventType = 'Created' AND ao.id = s.id AND ao.parent=e.id AND  
 5156 c1.classifiedObject = s.id AND c1.classificationNode = cn1.id AND  
 5157 cn1.path LIKE '%Plumbing' AND  
 5158 c2.classifiedObject = s.id AND c2.classificationNode = cn2.id AND  
 5159 cn2.path LIKE '%A Little Town%'

5160 -->

```
5161     </Selector>
5162     <Action xsi:type="tns:NotifyActionType" notificationOption="Objects"
5163     endPoint="mailto:someone@littletown.us"/>
5164 </tns:Subscription>
```

### 5165 **10.3.1 Event Selection**

5166 In order for a User to only be notified of specific events of interest, she must specify a reference  
5167 to a stored AdHocQuery via the selector attribute within the Subscription instance. The query  
5168 determines whether an event qualifies for that Subscription or not. The query syntax is the  
5169 standard ad hoc query syntax described in chapter 8.

### 5170 **10.3.2 Notification Action**

5171 When creating a Subscription, a User may also specify Actions within the subscription that  
5172 specify what the registry must do when an event matching the Subscription (subscription event)  
5173 transpires.

5174 A user may omit specifying an Action within a Subscription if does not wish to be notified by the  
5175 registry. A user may periodically poll the registry and pull the pending Notifications.

5176 [ebRIM] defines two standard ways that a NotifyAction may be used:

- 5177 • Email NotifyAction that allows delivery of event notifications via email to a human user  
5178 or to an email end point for a software component or agent.
- 5179 • Service NotifyAction that allows delivery of event notifications via a programmatic  
5180 interface by invoking a specified listener web service.

5181 For each event that transpires in the registry, if the registry supports event notification, it must  
5182 check all registered and active Subscriptions and see if any Subscriptions match the event. If a  
5183 match is found then the registry performs the Notification Actions required for the Subscription.

### 5184 **10.3.3 Subscription Authorization**

5185 A registry may use registry specific policies to decide which User is authorized to create a  
5186 subscription and to what events. A Registry must return an AuthorizationException in the event  
5187 that an Unauthorized User submits a Subscription to a registry.

### 5188 **10.3.4 Subscription Quotas**

5189 A registry may use registry specific policies to decide an upper limit on the number of  
5190 Subscriptions a User is allowed to create. A Registry must return a QuotaExceededException in  
5191 the event that an Authorized User submits more Subscriptions than allowed by their registry  
5192 specific quota.

### 5193 **10.3.5 Subscription Expiration**

5194 Each subscription defines a startDate and and endDate attribute which determines the period  
5195 within which a Subscription is active. Outside the bounds of the active period, a Subscription may  
5196 exist in an expired state within the registry. A registry may remove an expired Subscription at  
5197 any time. In such cases the identity of a RegistryOperator User must be used for the request in  
5198 order to have sufficient authorization to remove a User's Subscription.

5199 A Registry must not consider expired Subscriptions when delivering notifications for an event to  
5200 its Subscriptions. An expired Subscription may be renewed by submitting a new Subscription.

### 5201 10.3.6 Subscription Rejection

5202 A Registry may reject a Subscription if it is too costly to support. For instance a Subscription that  
5203 wishes to be notified of any change in any object may be too costly for most registries. A  
5204 Registry MUST return a SubscriptionTooCostlyException in the event that an Authorized User  
5205 submits a Subscription that is too costly for the registry to process.

## 5206 10.4 Unsubscribing from Events

5207 A User may terminate a Subscription with a registry if she no longer wishes to be notified of  
5208 events related to that Subscription. A User terminates a Subscription by deleting the  
5209 corresponding Subscription object using the RemoveObjectsRequest to the registry.  
5210 Removal of a Subscription object follows the same rules as removal of any other object.

## 5211 10.5 Notification of Events

5212 A registry performs the Actions for a Subscription in order to actually deliver the events.  
5213 However, regardless of the specific delivery action, the registry must communicate the  
5214 Subscription events. The Subscription events are delivered within a Notification instance as  
5215 described by [ebRIM]. In case of Service NotifyAction, the Notification is delivered to a handler  
5216 service conformant to the RegistryClient interface described in section 6.7.3. In case of an Email  
5217 NotifyAction the notification is delivered an email address.

5218

5219 The listing below shows a sample Notification matching the subscription example in section  
5220 10.3:

5221  
5222  
5223  
5224  
5225  
5226  
5227  
5228  
5229  
5230  
5231  
5232  
5233  
5234  
5235  
5236  
5237  
5238  
5239  
5240  
5241  
5242  
5243  
5244

```
<?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:rims="urn:oasis:names:tc:ebxml-regrep:rims: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">
  <rims:LeafRegistryObjectList>
    <rims:Service id="f3373a7b-4958-4e55-8820-d03a191fb76a">
      <rims:Name>
        <rims:LocalizedString value="A Little Town
Plumbing"/>
      </rims:Name>
      <rims:Classification id="a3373a7b-4958-4e55-8820-
d03a191fb76a" classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
      <rims:Classification id="b3373a7b-4958-4e55-8820-
d03a191fb76a" classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
    </rims:Service>
  </rims:LeafRegistryObjectList>
</tns:Notification>
```

5245

5246 [ebRIM] defines an extensible description of Notifications, making it possible to allow for  
5247 registry or application specific Notifications. It defines several normative types of Notifications.

5248 A client may specify the type of Notification they wish to receive using the notificationOption  
 5249 attribute of the Action within the Subscription. The registry may override this notificationOption  
 5250 based upon registry specific operational policies.

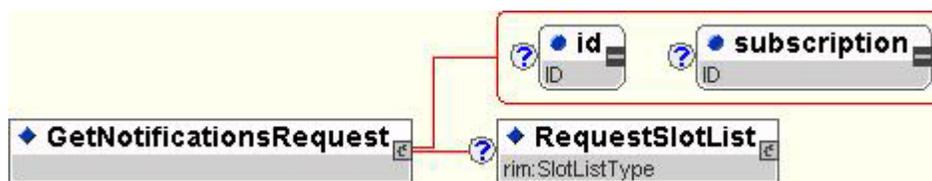
## 5251 10.6 Retrieval of Events

5252 The registry provides asynchronous PUSH style delivery of Notifications via notify Actions as  
 5253 described earlier. However, a client may also use a PULL style to retrieve any pending events for  
 5254 their Subscriptions. Pulling of events is done using the Get Notifications protocol.

### 5255 10.6.1 GetNotificationsRequest

5256 The GetNotificationsRequest is used by a client to retrieve any pending events for their  
 5257 Subscriptions.

5258 **Syntax:**



5259  
 5260

Figure 74: GetNotificationsRequest Syntax

5261 **Parameters:**

- 5262     ▪ *subscription*: This parameter specifies the id to a Subscription object which the  
 5263     client wishes to get Notifications.

5264

5265 **Returns:**

5266 This request returns a NotificationType. See section 0 for details.

5267 **Exceptions:**

5268 In addition to the exceptions common to all requests, the following exceptions may be returned:

- 5269     ▪ *ObjectNotFoundException*: signifies that the specified Subscription was not found  
 5270     in the registry.

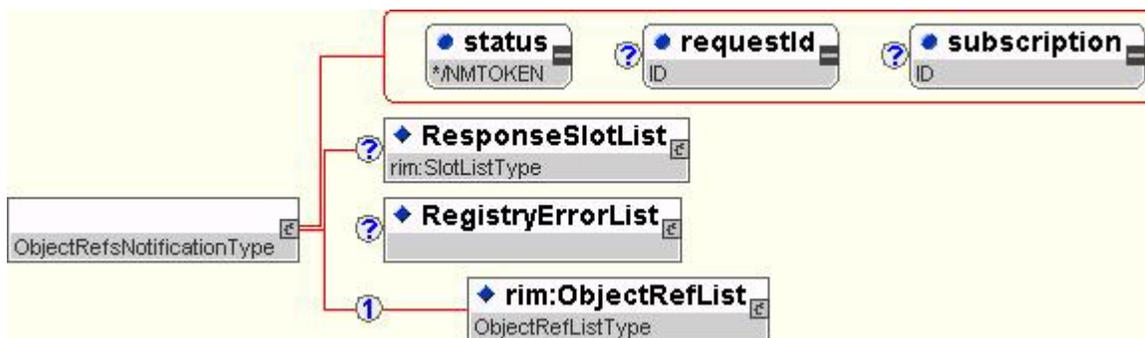
5271

### 5272 10.6.2 NotificationType

5273 NotificationType is the simplest form of notification. It is the base type for all types of  
 5274 Notifications.

5275

5276 **Syntax:**



5277  
5278

Figure 75: NotificationType Syntax

5279 **Parameters:**

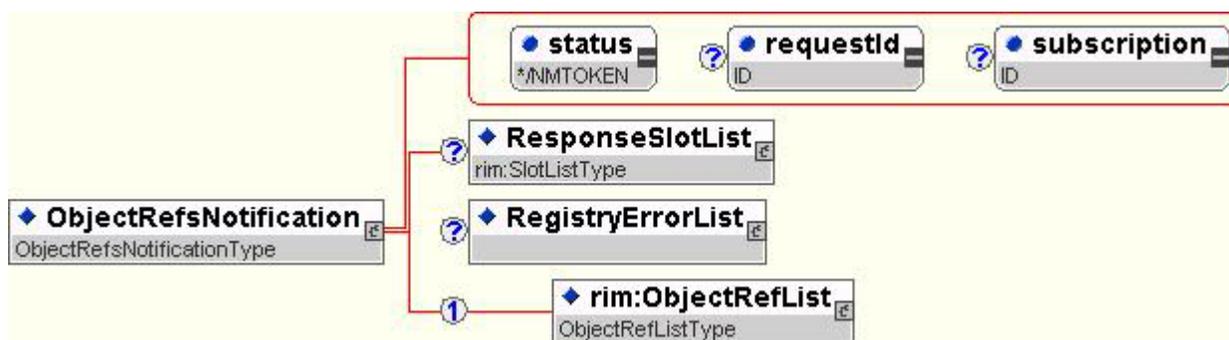
- 5280 *subscription*: This parameter specifies the id to a Subscription object for which  
5281 this is a Notification.

### 5282 10.6.3 ObjectRefsNotification

5283 ObjectRefsNotification is a concrete type of Notification that may be sent by the registry as a  
5284 response to GetNotificationsRequest. It extends NotificationType.

5285

5286 **Syntax:**



5287  
5288

Figure 76: ObjectRefsNotification Syntax

5289 **Parameters:**

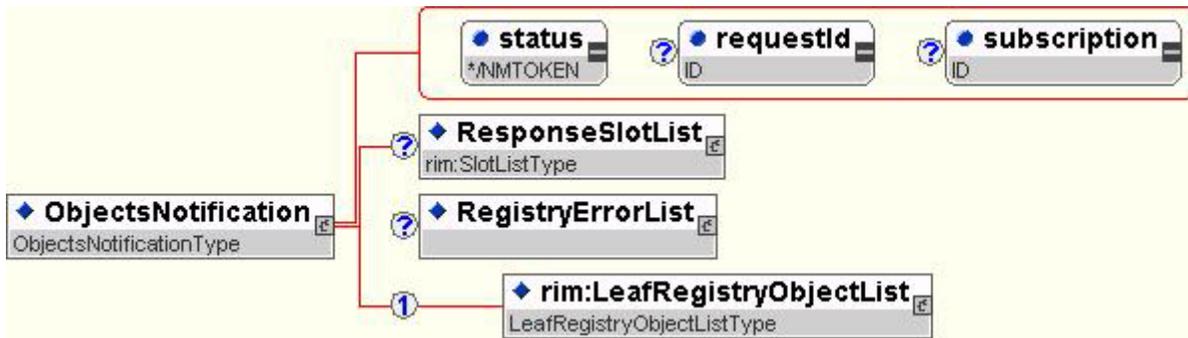
- 5290 *ObjectRefList*: This parameter specifies a Collection of ObjectRef instances  
5291 where each ObjectRef is to a RegistryObject that matches the Subscription for  
5292 which this is a Notification. The client must retrieve the actual RegistryObjects  
5293 separately using the ObjectRefs.

5294

### 5295 10.6.4 ObjectsNotification

5296 ObjectsNotification is a concrete type of Notification that may be sent by the registry as a  
5297 response to GetNotificationsRequest. It extends NotificationType.

5298

5299 **Syntax:**

5300

5301

Figure 77: ObjectsNotification Syntax

5302 **Parameters:**

5303

5304

5305

5306

- *LeafRegistryObjectList*: This parameter specifies a Collection of RegistryObject instances where each RegistryObject is one that matches the Subscription for which this is a Notification.

## 5307 10.7 Purging of Events

5308

5309

5310

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.

## 5311 11 Cooperating Registries Support

5312 This chapter describes the capabilities and protocols that enable multiple ebXML registries to  
5313 cooperate with each other to meet advanced use cases.

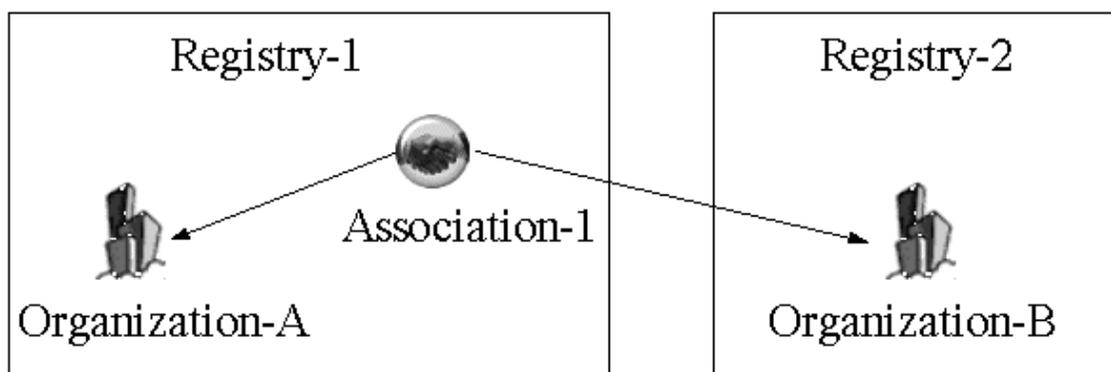
### 5314 11.1 Cooperating Registries Use Cases

5315 The following is a list of use cases that illustrate different ways that ebXML registries cooperate  
5316 with each other.

#### 5317 11.1.1 Inter-registry Object References

5318 A Submitting Organization wishes to submit a RegistryObject to a registry such that the  
5319 submitted object references a RegistryObject in another registry.

5320 An example might be where a RegistryObject in one registry is associated with a RegistryObject  
5321 in another registry.



5322  
5323 **Figure 78: Inter-registry Object References**

5324

#### 5325 11.1.2 Federated Queries

5326 A client wishes to issue a single query against multiple registries and get back a single response  
5327 that contains results based on all the data contained in all the registries. From the client's  
5328 perspective it is issuing its query against a single logical registry that has the union of all data  
5329 within all the physical registries.

#### 5330 11.1.3 Local Caching of Data from Another Registry

5331 A destination registry wishes to cache some or all the data of another source registry that is  
5332 willing to share its data. The shared dataset is copied from the source registry to the destination  
5333 registry and is visible to queries on the destination registry even when the source registry is not  
5334 available.

5335 Local caching of data may be desirable in order to improve performance and availability of  
5336 accessing that object.

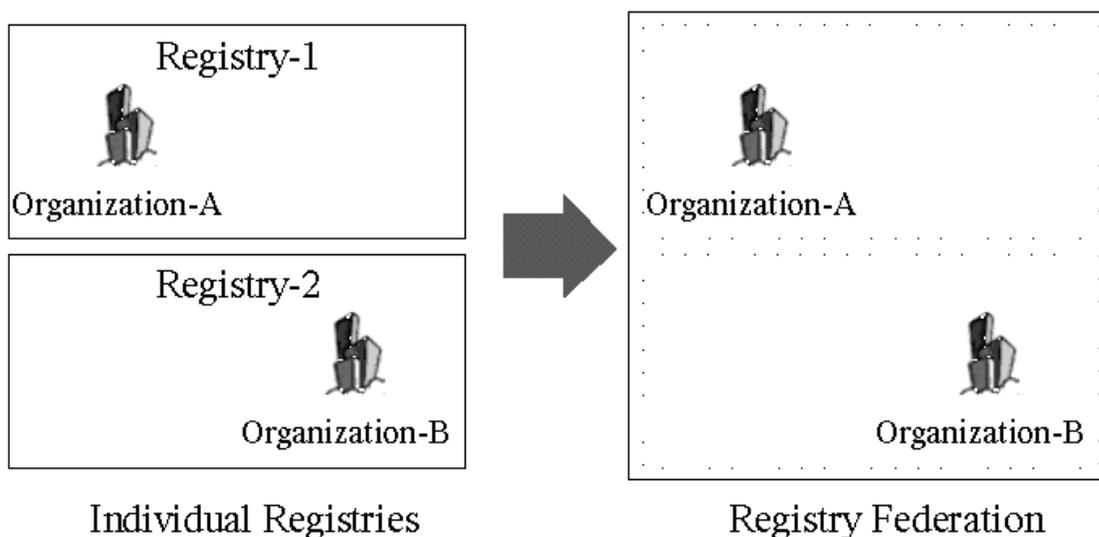
5337 An example might be where a RegistryObject in one registry is associated with a RegistryObject  
5338 in another registry, and the first registry caches the second RegistryObject locally.

### 5339 11.1.4 Object Relocation

5340 A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from  
5341 the registry where it was submitted to another registry.

## 5342 11.2 Registry Federations

5343 A registry federation is a group of registries that have voluntarily agreed to form a loosely  
5344 coupled union. Such a federation may be based on common business interests and specialties that  
5345 the registries may share. Registry federations appear as a single logical registry, to registry  
5346 clients.



5347  
5348

**Figure 79: Registry Federations**

5349 Registry federations are based on a peer-to-peer (P2P) model where all participating registries  
5350 are equal. Each participating registry is called a *registry peer*. There is no distinction between the  
5351 registry operator that created a federation and those registry operators that joined that Federation  
5352 later.

5353 Any registry operator may form a registry federation at any time. When a federation is created it  
5354 must have exactly one registry peer which is the registry operated by the registry operator that  
5355 created the federation.

5356 Any registry may choose to voluntarily join or leave a federation at any time.

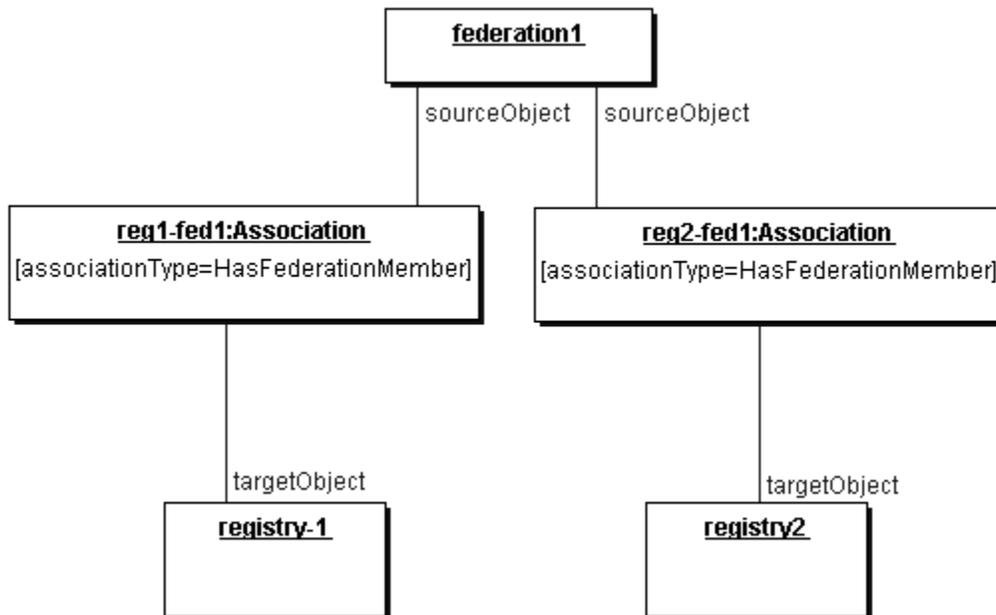
### 5357 11.2.1 Federation Metadata

5358 The Registry Information model defines the Registry and Federation classes, instances of these  
5359 classes and the associations between these instances describe a federation and its members. Such  
5360 instance data is referred to as Federation Metadata. The Registry and Federation classes are  
5361 described in detail in [ebRIM].

5362 The Federation information model is summarized here as follows:

- 5363 ○ A Federation instance represents a registry federation.
- 5364 ○ A Registry instance represents a registry that is a member of the Federation.

- 5365 ○ An Association instance with associationType of *HasFederationMember* represents  
 5366 membership of the registry in the federation. This Association links the Registry instance  
 5367 and the Federation instance.  
 5368



5369  
5370

Figure 80: Federation Metadata Example

### 5371 11.2.2 Local Vs. Federated Queries

5372 A federation appears to registry clients as a single unified logical registry. An  
 5373 AdhocQueryRequest sent by a client to a federation member may be local or federated. A new  
 5374 boolean attribute named *federated* is added to AdhocQueryRequest to indicate whether the query  
 5375 is federated or not.

#### 5376 Local Queries

5377 When the federated attribute of AdhocQueryRequest has the value of *false* then the query is a  
 5378 local query. In the absence of a *federated* attribute the default value of *federated* attribute is *false*.  
 5379 A local AdhocQueryRequest is only processed by the registry that receives the request. A local  
 5380 AdhocQueryRequest does not operate on data that belongs to other registries.

#### 5381 Federated Queries

5382 When the *federated* attribute of AdhocQueryRequest has the value of *true* then the query is a  
 5383 federated query.

5384 A federated query to any federation member must be routed by that member to all other  
 5385 federation member registries as parallel-distributed queries. A federated query operates on data  
 5386 that belongs to all members of the federation.

5387 When a client submits a federated query to a registry that is not a member of a federation, the  
 5388 registry must treat it as a local query.

### 5389 **Membership in Multiple Federations**

5390 A registry may be a member of multiple federations. In such cases if the *federated* attribute of  
5391 AdhocQueryRequest has the value of *true* then the registry must route the federated query to *all*  
5392 federations that it is a member of.

5393 Alternatively, the client may specify the id of a specific federation that the registry is a member  
5394 of, as the value of the *federation* parameter. The type of the federation parameter is anyURI and  
5395 identifies the “id” attribute of the desired Federation.

5396 In such cases the registry must route the federated query to the specified federation only.

### 5397 **11.2.3 Federated Lifecycle Management Operations**

5398 Details on how to create and delete federations and how to join and leave a federation are  
5399 described in 11.2.8.

5400 All lifecycle operations must be performed on a RegistryObject within its home registry using  
5401 the operations defined by the LifeCycleManager interface. Unlike query requests, lifecycle  
5402 management requests do not support any federated capabilities.

### 5403 **11.2.4 Federations and Local Caching of Remote Data**

5404 A federation member is not required to maintain a local cache of replicas of RegistryObjects and  
5405 repository items that belong to other members of the federation.

5406 A registry may choose to locally cache some or all data from any other registry whether that  
5407 registry is a federation member or not. Data caching is orthogonal to registry federation and is  
5408 described in section 11.3.

5409 Since by default there is minimal replication in the members of a federation, the federation  
5410 architecture scales well with respect to memory and disk utilization at each registry.

5411 Data replication is often necessary for performance, scalability and fault-tolerance reasons.

### 5412 **11.2.5 Caching of Federation Metadata**

5413 A special case for local caching is the caching of the Federation and Registry instances and  
5414 related Associations that define a federation and its members. Such data is referred to as  
5415 federation metadata. A federation member is required to locally cache the federation metadata,  
5416 from the federation home for each federation that it is a member of. The reason for this  
5417 requirement is consistent with a Peer-to-Peer (P2P) model and ensures fault –tolerance in case  
5418 the Federation home registry is unavailable.

5419 The federation member must keep the cached federation metadata synchronized with the master  
5420 copy in the Federation home, within the time period specified by the replicationSyncLatency  
5421 attribute of the Federation. Synchronization of cached Federation metadata may be done via  
5422 synchronous polling or asynchronous event notification using the event notification feature of the  
5423 registry.

### 5424 **11.2.6 Time Synchronization Between Registry Peers**

5425 Federation members are not required to synchronize their system clocks with each other.

5426 However, each Federation member SHOULD keep its clock synchronized with an atomic clock  
5427 server within the latency described by the replicationSyncLatency attribute of the Federation.

## 5428 **11.2.7 Federations and Security**

5429 Federation lifecycle management operations abide by the same security rules as standard  
5430 lifecycle management.

## 5431 **11.2.8 Federation Lifecycle Management Protocols**

5432 This section describes the various operations that manage the lifecycle of a federation and its  
5433 membership. A key design objective is to allow federation lifecycle operations to be done using  
5434 standard LifeCycleManager interface of the registry in a stylized manner.

### 5435 **Joining a Federation**

5436 The following rules govern how a registry joins a federation:

- 5437 • Each registry must have exactly one Registry instance within that registry for which it is  
5438 a home. The Registry instance is owned by the RegistryOperator and may be placed in  
5439 the registry using any operator specific means. The Registry instance must never change  
5440 its home registry.
- 5441 • A registry may request to join an existing federation by submitting an instance of an  
5442 Extramural Association that associates the Federation instance as sourceObject, to its  
5443 Registry instance as targetObject, using an associationType of *HasFederationMember*.  
5444 The home registry for the Association and the Federation objects must be the same.
- 5445 • The owner of the Federation instance must confirm the Extramural Association in order  
5446 for the registry to be accepted as a member of the federation.

### 5447 **Creating a Federation**

5448 The following rules govern how a federation is created:

- 5449 • A Federation is created by submitting a Federation instance to a registry using  
5450 SubmitObjectsRequest.
- 5451 • The registry where the Federation is submitted is referred to as the federation home.
- 5452 • The federation home may or may not be a member of that Federation.
- 5453 • A federation home may contain multiple Federation instances.

### 5454 **Leaving a Federation**

5455 The following rules govern how a registry leaves a federation:

5456 A registry may leave a federation at any time by removing its *HasFederationMember*  
5457 Association instance that links it with the Federation instance. This is done using the standard  
5458 RemoveObjectsRequest.

### 5459 **Dissolving a Federation**

5460 The following rules govern how a federation is dissolved:

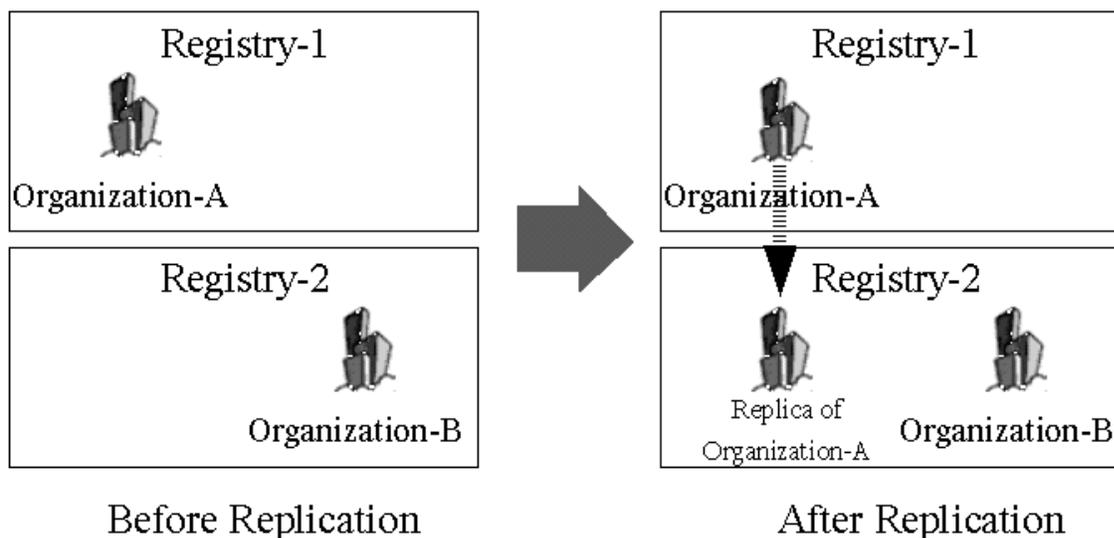
- 5461 • A federation is dissolved by sending a RemoveObjectsRequest to its home registry and  
5462 removing its Federation instance.
- 5463 • The removal of a Federation instance is controlled by the same Access Control Policies  
5464 that govern any RegistryObject.

- 5465
- 5466
- 5467
- 5468
- 5469
- 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.

## 5470 11.3 Object Replication

5471 RegistryObjects within a registry may be replicated in another registry. A replicated copy of a  
 5472 remote object is referred to as its replica. The remote object may be an original object or it may  
 5473 be a replica. A replica from an original is referred to as a first-generation replica. A replica of a  
 5474 replica is referred to as a second-generation replica (and so on).

5475 The registry that replicates a remote object locally is referred to as the destination registry for the  
 5476 replication. The registry that contains the remote object being replicated is referred to as the  
 5477 source registry for the replication.  
 5478



5479

5480

Figure 81: Object Replication

5481

### 5482 11.3.1 Use Cases for Object Replication

5483 A registry may create a local replica of a remote object for a variety of reasons. A few sample  
 5484 use cases follow:

- 5485
- 5486
- 5487
- 5488
- 5489
- 5490
- 5491
- 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.

### 5492 **11.3.2 Queries And Replicas**

5493 A registry must support client queries to consider a local replica of remote object as if it were a  
5494 local object. Local replicas are considered within the extent of the data set of a registry as far as  
5495 local queries are concerned.

5496 When a client submits a local query that retrieves a remote object by its id attribute, if the  
5497 registry contains a local replica of that object then the registry should return the state defined by  
5498 the local replica.

### 5499 **11.3.3 Lifecycle Operations And Replicas**

5500 LifeCycle operations on an original object must be performed at the home registry for that  
5501 object. LifeCycle operations on replicas of an original object only affect the replica and do not  
5502 have any impact on the original object.

### 5503 **11.3.4 Object Replication and Federated Registries**

5504 Object replication capability is orthogonal to the registry federation capability. Objects may be  
5505 replicated from any registry to any other registry without any requirement that the registries  
5506 belong to the same federation.

### 5507 **11.3.5 Creating a Local Replica**

5508 Any Submitting Organization can create a replica by using the standard SubmitObjectsRequest.  
5509 If a registry receives a SubmitObjectRequest which has an RegistryObjectList containing a  
5510 remote ObjectRef, then it must create a replica for that remote ObjectRef.

5511 In addition to Submitting Organizations, a registry itself may create a replica under specific  
5512 situations in a registry specific manner.

5513 Creating a local replica requires the destination registry to read the state of the remote object  
5514 from the source registry and then create a local replica of the remote object.

5515 A registry may use standard QueryManager interface to read the state of a remote object  
5516 (whether it is an original or a replica). No new APIs are needed to read the state of a remote  
5517 object. Since query functionality does not need prior registration, no prior registration or contract  
5518 is needed for a registry to read the state of a remote object.

5519 Once the state of the remote object has been read, a registry may use registry specific means to  
5520 create a local replica of the remote object. Such registry specific means may include the use of  
5521 the LifeCycleManager interface.

5522 A replica of a RegistryObject may be distinguished from an original since a replica must have its  
5523 home attribute point to the remote registry where the original for the replica resides.

### 5524 **11.3.6 Transactional Replication**

5525 Transactional replication enables a registry to replicate events in another registry in a  
5526 transactionally consistent manner. This is typically the case when entire registries are replicated  
5527 to another registry.

5528 This specification defines a more loosely coupled replication model as an alternative to  
5529 transactional replication for the following reasons:

- 5530 • Transactional replication requires a tight coupling between registries participating in the  
5531 replication
- 5532 • Transactional replication is not a typical use case for registries
- 5533 • Loosely coupled replication as defined by this specification typically suffices for most  
5534 use cases
- 5535 • Transaction replication is very complex and error prone

5536

5537

5538 Registry implementations are not required to implement transactional replication.

### 5539 **11.3.7 Keeping Replicas Current**

5540 A registry must keep its replicas current within the latency specified by the value of the  
5541 *replicationSyncLatency* attribute defined by the registry. This includes removal of the replica  
5542 when its original is removed from its home registry.

5543 Replicas may be kept current using the event notification feature of the registry or via periodic  
5544 polling.

### 5545 **11.3.8 Write Operations on Local Replica**

5546 Local Replicas are read-only objects. Lifecycle management operations of RegistryObjects are  
5547 not permitted on local replicas. All lifecycle management operation to RegistryObjects must be  
5548 performed in the home registry for the object.

### 5549 **11.3.9 Tracking Location of a Replica**

5550 A local replica of a remote RegistryObject instance must have exactly one ObjectRef instance  
5551 within the local registry. The home attribute of the ObjectRef associated with the replica tracks  
5552 its home location. A RegistryObject must have exactly one home. The home for a RegistryObject  
5553 may change via Object Relocation as described in section 11.4. It is optional for a registry to  
5554 track location changes for replicas within it.

### 5555 **11.3.10 Remote Object References to a Replica**

5556 It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another  
5557 RegistryObject. In such cases the home attribute of the ObjectRef contains the base URI to the  
5558 home registry for the replica.

### 5559 **11.3.11 Removing a Local Replica**

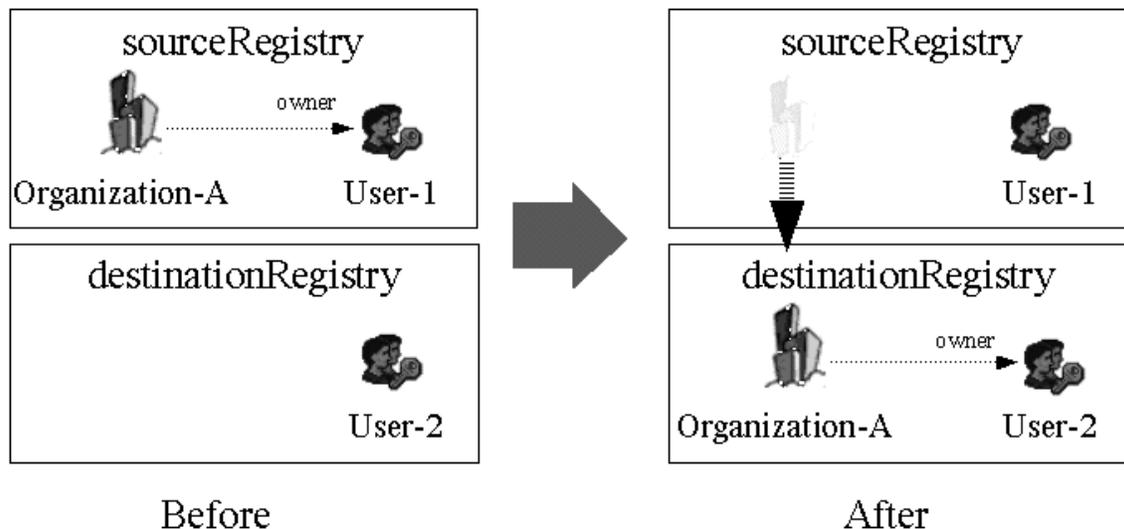
5560 A Submitting Organization can remove a replica by using the RemoveObjectsRequest. If a  
5561 registry receives a RemoveObjectsRequest that has an ObjectRefList containing a remote  
5562 ObjectRef, then it must remove the local replica for that remote ObjectRef.

## 5563 **11.4 Object Relocation Protocol**

5564 Every RegistryObject has a home registry and a User within the home registry that is the  
5565 publisher or owner of that object. Initially, the home registry is the where the object is originally  
5566 submitted. Initially, the owner is the User that submitted the object.

5567 A RegistryObject may be relocated from one home registry to another home registry using the  
5568 Object Relocation protocol.

5569 Within the Object Relocation protocol, the new home registry is referred to as the *destination*  
5570 registry while the previous home registry is called the *source* registry.



5571

Before

After

5572

Figure 82: Object Relocation

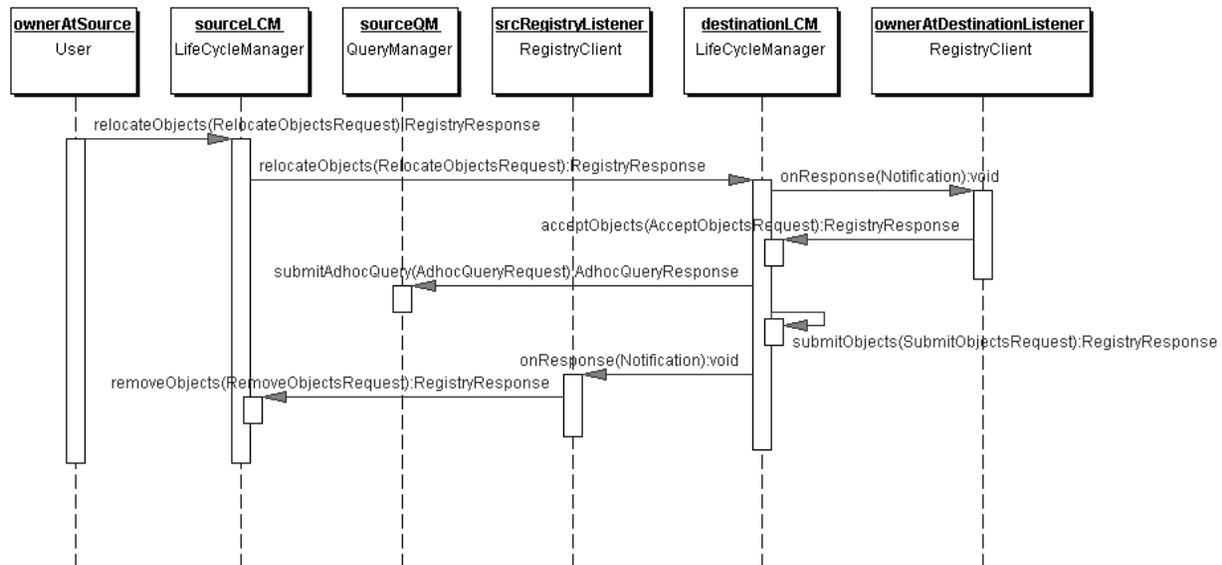
5573 The User at the source registry who owns the objects being relocated is referred to as the  
5574 *ownerAtSource*. The User at the destination registry, who is the new owner of the objects, is  
5575 referred to as the *ownerAtDestination*. While the *ownerAtSource* and the *ownerAtDestination*  
5576 may often be the same identity, the Object Relocation protocol treats them as two distinct  
5577 identities.

5578 A special case usage of the Object Relocation protocol is to transfer ownership of  
5579 RegistryObjects from one User to another within the same registry. In such cases the protocol is  
5580 the same except for the fact that the source and destination registries are the same.

5581 Following are some notable points regarding object relocation:

- 5582 • Object relocation does not require that the source and destination registries be in the same  
5583 federation or that either registry have a prior contract with the other.
- 5584 • Object relocation must preserve object id. While the home registry for a RegistryObject  
5585 may change due to object relocation, its id never changes.
- 5586 • ObjectRelocation must preserve referential integrity of RegistryObjects. Relocated  
5587 objects that have references to an object that did not get relocated must preserve their  
5588 reference. Similarly objects that have references to a relocated object must also preserve  
5589 their reference. Thus, relocating an object may result in making the value of a reference  
5590 attribute go from being a local reference to being a remote reference or vice versa.
- 5591 • AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque  
5592 transactionId identifying the relocateObjects transaction.
- 5593 • The requests defined by the Relocate Objects protocol must be sent to the source or  
5594 destination registry only.

- 5595
- 5596
- 5597
- 5598
- 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.



5599

5600

**Figure 83: Relocate Objects Protocol**

5601 Figure 83 illustrates the Relocate Objects Protocol. The participants in the protocol are the

5602 ownerAtSource and ownerAtDestination User instances as well as the LifeCycleManager

5603 interfaces of the sourceRegistry and destinationRegistry.

5604 The steps in the protocol are described next:

- 5605
- 5606
- 5607
- 5608
- 5609
- 5610
- 5611
- 5612
- 5613
- 5614
- 5615
- 5616
- 5617
- 5618
- 5619
- 5620
- 5621
- 5622
- 5623
- 5624
- 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.
  - Next, the sourceRegistry must relay the same RelocateObjectsRequest message to the LifeCycleManager interface of the destinationRegistry. This message enlists the destinationRegistry 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.
  - 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.
  - 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.

- 5625 5. The destinationRegistry sends an AdhocQueryRequest message to the sourceRegistry.  
 5626 The source registry returns the objects being relocated as an AdhocQueryResponse. In  
 5627 the event of a large number of objects this may involve multiple  
 5628 AdhocQueryRequest/responses as described by the iterative query feature described in  
 5629 section 8.1.4.
- 5630 6. The destinationRegistry submits the relocated data to itself assigning the identity of the  
 5631 ownerAtDestination as the owner. The relocated data may be submitted to the destination  
 5632 registry using any registry specific means or a SubmitObjectsRequest. However, the  
 5633 effect must be the same as if a SubmitObjectsRequest was used.
- 5634 7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been  
 5635 safely committed using the Event Notification feature of the registry as described in  
 5636 chapter 10.
- 5637 8. The sourceRegistry removes the relocated objects using any registry specific means and  
 5638 logging an AuditableEvent of type Relocated. This concludes the Object Relocation  
 5639 transaction.

#### 5640 11.4.1 RelocateObjectsRequest

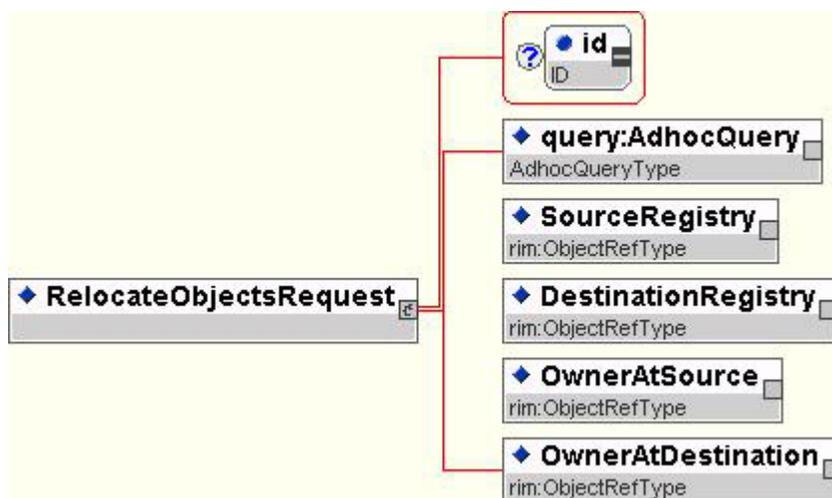


Figure 84: RelocateObjectsRequest XML Schema

5641  
 5642

#### 5643 Parameters:

- 5644
- 5645 ▪ *id*: the attribute *id* provides the transaction identifier for this instance of the protocol.
  - 5646 ▪ *AdhocQuery*: This element specifies an ad hoc query that selects the RegistryObjects that are being relocated.
  - 5647
  - 5648 ▪ *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.
  - 5649
  - 5650
  - 5651 ▪ *destinationRegistry*: This element specifies the ObjectRef to the destinationRegistry Registry instance.
  - 5652
  - 5653 ▪ *ownerAtSource*: This element specifies the ObjectRef to the ownerAtSource User instance.
  - 5654



5687 5. During this forwarding interval the person notifies interested parties of their change of  
5688 address.

5689 The registry must support a similar model for relocation of RegistryObjects. The following steps  
5690 describe the expected behavior when an object is relocated.

- 5691 1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other  
5692 RegistryObjects may have remote ObjectRefs to O1.
- 5693 2. The registry R1 must leave an AuditableEvent of type Relocated that includes the home  
5694 URI for the new registry R2.
- 5695 3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it  
5696 must forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the  
5697 client. The object O1 must include the home URI to R2 within the optional home  
5698 attribute of RegistryObject. Clients are advised to check the home attribute and update  
5699 the home attribute of their local ObjectRef to match the new home URI value for the  
5700 object.
- 5701 4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no  
5702 longer required to relay requests for O1 to R2 transparent to the client. Instead R1 must  
5703 return an ObjectNotFoundException.
- 5704 5. Clients that are interested in the relocation of O1 and being notified of its new address  
5705 may choose to be notified by having a prior subscription using the event notification  
5706 facility of the registry. For example a Registry that has a remote ObjectRefs to O1 may  
5707 create a subscription on relocation events for O1. This however, is not required behavior.

#### 5708 **11.4.4 Notification of Object Relocation To ownerAtDestination**

5709 This section describes how the destinationRegistry uses the event notification feature of the  
5710 registry to notify the ownerAtDestination of a Relocated event.

5711 The destinationRegistry must send a Notification with the following required characteristics:

- 5712 • The notification must be an instance of a Notification element.
- 5713 • The Notification instance must have at least one Slot.
- 5714 • The Slot must have the name eventNotification.correlationId
- 5715 • The Slot must have the correlationId for the Object Relocation transaction as the value of  
5716 the Slot.

5717

#### 5718 **11.4.5 Notification of Object Commit To sourceRegistry**

5719 This section describes how the destinationRegistry uses the event notification feature of the  
5720 registry to notify the sourceRegistry that it has completed committing the relocated objects.

5721 The destinationRegistry must send a Notification with the following required characteristics:

- 5722 • The notification must be an instance of a Notification element.
- 5723 • The Notification instance must have at least one Slot.
- 5724 • The Slot must have the name eventNotification.objectsCommitted
- 5725 • The Slot must have the value of *true*.

5726

**5727 11.4.6 Object Relocation and Timeouts**

5728 No timeouts are specified for the Object Relocation protocol. Registry implementations may  
5729 cleanup incomplete Object Relocation transactions in a registry specific manner as an  
5730 administrative task using registry specific policies.

## 5731 **12 Registry Security**

5732 This chapter describes the security features of the ebXML Registry. It is assumed that the reader  
5733 is familiar with the security related classes in the Registry information model as described in  
5734 [ebRIM]. Security glossary terms can be referenced from RFC 2828.

### 5735 **12.1 Security Concerns**

5736 In the current version of this specification, we address data integrity and source integrity (item 1  
5737 in Appendix E.1). We have used a minimalist approach to address the access control concern as  
5738 in item 2 of Appendix E.1. By default, any Registered User identified by a User instance as  
5739 defined by [ebRIM] can publish content and anyone can view published content. In addition to  
5740 this default behaviour, the Registry Information Model [ebRIM] is designed to support more  
5741 sophisticated security policies in future versions of this specification.

### 5742 **12.2 Integrity of Registry Content**

5743 It is assumed that most registries do not have the resources to validate the veracity of the content  
5744 submitted to them. The mechanisms described in this section can be used to ensure that any  
5745 tampering with the content can be detected. Furthermore, these mechanisms support  
5746 unambiguous identification of a Registered User as the submitter for any registry content. The  
5747 Registered User has to sign the contents before submission – otherwise the content will be  
5748 rejected.

#### 5749 **12.2.1 Message Payload Signature**

5750 The integrity of the Registry content requires that all submitted content be signed by the  
5751 Registered User that submits the content. The signature on the submitted content ensures that:

- 5752 • Any tampering of the content can be detected.
- 5753 • The content's veracity can be ascertained by its association with a specific Registered User  
5754 and its indirect association with the Organization that the Registered User may be affiliated  
5755 with.

5756 This section specifies the requirements for generation, packaging and validation of payload  
5757 signatures. A payload signature is packaged with the payload. Therefore the requirements apply  
5758 regardless of whether the Registry Client and the Registration Authority communicate over  
5759 standard SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML  
5760 Messaging Service does not specify the generation, validation and packaging of payload  
5761 signatures. The specification of payload signatures is left up to the application and therefore  
5762 defined by this specification for ebXML Registry client applications. The requirements on the  
5763 payload signatures augment the [ebMS] specification.

#### 5764 **Use Case**

5765 This Use Case illustrates the use of header and payload signatures (we discuss header signatures  
5766 later).

- 5767 • RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the  
5768 content along with the payload signature to the Registry.

- 5769
- RC2 (Registry Client 2) retrieves RC1's content from the Registry.
- 5770
- RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
- 5771 the content, the response from the Registration Authority to RC2 contains the following:
- 5772
- Payload containing the content that has been published by RC1.
- 5773
- RC1's payload signature (represented by a ds:Signature element) over RC1's published
- 5774 content.
- 5775
- The public key for validating RC1's payload signature in ds:Signature element ( using the
- 5776 KeyInfo element as specified in [XMLDSIG] ) so RC2 can obtain the public key for
- 5777 signature (e.g. retrieve a certificate containing the public key for RC1).
- 5778
- A ds:Signature element containing the header signature. Note that the Registration
- 5779 Authority (not RC1) generates this signature.

## 5780 12.2.2 Payload Signature Requirements

### 5781 Payload Signature Packaging Requirements

5782 A payload signature is represented by a ds:Signature element. The payload signature must be

5783 packaged with the payload as specified here. This packaging assumes that the payload is always

5784 signed.

- 5785
- The payload and its signature must be enclosed in a MIME multipart message with a
- 5786 Content-Type of multipart/related.
- 5787
- The first body part must contain the XML signature as specified in Section 0, "Payload
- 5788 Signature Generation Requirements".
- 5789
- The second body part must be the content.

5790 The packaging of the payload signature with two payloads is as shown in the example in Section

5791 8.4.4.

### 5792 Payload Signature Generation Requirements

5793 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in

5794 this section. Note: the "ds" name space reference is to <http://www.w3.org/2000/09/xmlsig#>

- 5795
- ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
- 5796 using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a
- 5797 client may use any of these attributes. However, signing using the following Algorithm
- 5798 attribute: <http://www.w3.org/2000/09/xmlsig#dsa-sha1> will allow interoperability with all
- 5799 XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this
- 5800 algorithm.

5801 The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The following

5802 Canonicalization algorithm (specified in [XMLDSIG]) must be supported

5803 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

- 5804
- One ds:Reference element to reference each of the payloads that needs to be signed must be
- 5805 created. The ds:Reference element:
- 5806
- Must identify the payload to be signed using the URI attribute of the ds:Reference
- 5807 element.
- 5808
- Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support
- 5809 the following digest algorithm:

- 5810 <http://www.w3.org/2000/09/xmlsig#sha1>
- 5811 ○ Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].
- 5812 The ds:SignatureValue must be generated as specified in [XMLDSIG].
- 5813 The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject
- 5814 to the requirements stated in Section 12.4, “KeyDistrbution and KeyInfo element”.

### 5815 **Message Payload Signature Validation**

- 5816 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

### 5817 **Payload Signature Example**

- 5818 The following example shows the format of the payload signature:

```

5819
5820 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
5821 <ds:SignedInfo>
5822   <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmlsig#dsa-sha1"/>
5823   <ds:CanonicalizationMethod>
5824     Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
5825   </ds:CanonicalizationMethod>
5826   <ds:Reference URI=#Payload1>
5827     <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmlsig#sha1">
5828     <ds:DigestValue> ... </ds:DigestValue>
5829   </ds:Reference>
5830 </ds:SignedInfo>
5831 <ds:SignatureValue> ... </ds:SignatureValue>
5832 </ds:Signature>
5833

```

## 5834 **12.3 Authentication**

- 5835 The Registry must be able to authenticate the identity of the User associated with client requests
- 5836 and also the owner of each repository item.

- 5837 The User can be identified by verifying the message header signature with the certificate of the
- 5838 User. The certificate may be in the message itself or provided to the registry through means
- 5839 unspecified in this specification. If not provided in the message, this specification does not
- 5840 specify how the Registry correlates a specific message with a certificate. Each payload must also
- 5841 be signed to ensure the integrity of that payload and to determine its owner. Authentication is
- 5842 also required in order to identify the actions a User is authorized to perform with respect to
- 5843 specific RegistryObject resources in the Registry.

- 5844 The Registry must perform authentication on a per message basis. From a security point of view,
- 5845 all messages are independent and there is no concept of a session encompassing multiple
- 5846 messages or conversations. Session support may be added as an optimization feature in future
- 5847 versions of this specification.

- 5848 It is important to note that the message header signature can only guarantee data integrity. It does
- 5849 not guarantee safety from “replay” attacks. True support for authentication requires timestamps
- 5850 or nonce (nonrecurring series of numbers to identify each message) that are signed.

### 5851 12.3.1 Message Header Signature

5852 Message headers are signed to provide data integrity while the message is in transit. Note that the  
5853 signature within the message header also signs the digests of the payloads.

#### 5854 Header Signature Requirements

5855 Message headers may be signed and are referred to as a header signature. When a Registered  
5856 User sends a request, the Registration Authority may use a pre-established contract or a default  
5857 policy to determine whether the response should contain a header signature. When a Registry  
5858 Guest sends a request, the Registration Authority may use a default policy to determine whether  
5859 the response contains a header signature.

5860 This section specifies the requirements for generation, packaging and validation of a header  
5861 signature. These requirements apply when the Registry Client and Registration Authority  
5862 communicate using standard SOAP with Attachments. When ebXML MS is used for  
5863 communication, then the message handler (i.e. [ebMS]) specifies the generation, packaging and  
5864 validation of XML signatures in the SOAP header. Therefore the header signature requirements  
5865 do not apply when the ebXML MS is used for communication. However, payload signature  
5866 generation requirements (0) do apply whether standard SOAP with Attachments or ebXML MS  
5867 is used for communication.

#### 5868 Packaging Requirements

5869 A header signature is represented by a ds:Signature element. The ds:Signature element generated  
5870 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature  
5871 element in the SOAP header field is shown in Section 8.4.4.

#### 5872 Header Signature Generation Requirements

5873 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in  
5874 this section. A ds:Signature element contains:

- 5875 • ds:SignedInfo
- 5876 • ds:SignatureValue
- 5877 • ds:KeyInfo

5878 The ds:SignedInfo element must be generated as follows:

- 5879 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified  
5880 using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,  
5881 a client must be capable of signing using only the following Algorithm attribute:  
5882 <http://www.w3.org/2000/09/xmlsig#dsa-sha1>. All XMLDSIG implementations conforming to the  
5883 [XMLDSIG] specification support this algorithm.
- 5884 2. The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The  
5885 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:  
5886 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- 5887 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.  
5888 This signs the entire ds:Reference element and:  
5889 ○ Must include the ds:Transform  
5890 <http://www.w3.org/2000/09/xmlsig#enveloped-signature>

- 5891 This ensures that the signature (which is embedded in the <SOAP-ENV:Header>  
5892 element) is not included in the signature calculation.
- 5893 ○ Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the  
5894 ds:Reference element. The URI attribute is optional in the [XMLDSIG] specification.  
5895 The URI attribute must be an empty string (“”).
  - 5896 ○ Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support  
5897 the digest algorithm: <http://www.w3.org/2000/09/xmlsig#sha1>
  - 5898 ○ Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].
- 5899 The ds:SignatureValue must be generated as specified in [XMLDSIG].
- 5900 The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in  
5901 Section 12.4, “KeyDistribution and KeyInfo element”.

### 5902 Header Signature Validation Requirements

5903 The ds:Signature element for the ebXML message header must be validated by the recipient as  
5904 specified by [XMLDSIG].

### 5905 Header Signature Example

5906 The following example shows the format of a header signature:

```
5907
5908 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
5909   <ds:SignedInfo>
5910     <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmlsig#dsa-sha1/>
5911     <ds:CanonicalizationMethod>
5912       Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
5913     </ds:CanonicalizationMethod>
5914     <ds:Reference URI= "" >
5915       <ds:Transform>
5916         http://www.w3.org/2000/09/xmlsig#enveloped-signature
5917       </ds:Transform>
5918       <ds:DigestMethod DigestAlgorithm="./xmlsig#sha1">
5919       <ds:DigestValue> ... </ds:DigestValue>
5920     </ds:Reference>
5921   </ds:SignedInfo>
5922   <ds:SignatureValue> ... </ds:SignatureValue>
5923 </ds:Signature>
5924
```

## 5925 12.4 Key Distribution and KeyInfo Element

5926 To validate a signature, the recipient of the signature needs the public key corresponding to the  
5927 signer’s private key. The participants may use the KeyInfo field of ds:Signature, or distribute the  
5928 public keys in an implementation specific manner. In this section we consider the case when the  
5929 public key is sent in the KeyInfo field. The following use cases need to be addressed:

- 5930 • Registration Authority needs the public key of the Registry Client to validate the signature

- 5931 • Registry Client needs the public key of the Registration Authority to validate the Registry's  
5932 signature.
- 5933 • Registry Client RC1 needs the public key of Registry Client RC2 to validate the content  
5934 signed by RC1.
- 5935 • [XMLDSIG] provides an optional *ds:KeyInfo* element that can be used to pass information  
5936 to the recipient for retrieving the public key. This field together with the procedures outlined  
5937 in this section is used to securely pass the public key to a recipient. If the KeyInfo field is  
5938 present, it must contain a X509 Certificate as specified in [XMLDSIG].

5939 The following assumptions are also made:

- 5940 1. A Certificate is associated both with the Registration Authority and a Registry Client.
- 5941 2. A Registry Client registers its certificate with the Registration Authority. The mechanism  
5942 used for this is not specified here.
- 5943 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local  
5944 key store. The mechanism is not specified here.

5945 Appendix F.8 contains a few scenarios on the use of KeyInfo field.

## 5946 **12.5 Confidentiality**

### 5947 **12.5.1 On-the-wire Message Confidentiality**

5948 It is suggested but not required that message payloads exchanged between Registry Clients and  
5949 the Registry be encrypted during transmission. This specification does not specify how payload  
5950 encryption is to be done.

### 5951 **12.5.2 Confidentiality of Registry Content**

5952 In the current version of this specification, there are no provisions for confidentiality of Registry  
5953 content. All content submitted to the Registry may be discovered and read by any client. This  
5954 implies that the Registry and the client need to have an a priori agreement regarding encryption  
5955 algorithm, key exchange agreements, etc.

## 5956 **12.6 Access Control and Authorization**

5957 The Registry must provide an access control and authorization mechanism based on the Access  
5958 Control Information Model defined in [eBRIM]. This model defines a default access control  
5959 policy that must be supported by the registry. In addition it also defines a binding to [XACML]  
5960 that allows fine-grained access control policies to be defined.

### 5961 **12.6.1 Actors / Role Mapping**

5962 The following table shows the mapping of actors listed in Section 5.3 and their default roles.

5963

<b>Actor</b>	<b>Role</b>
Registered User	ContentOwner
Registry Administrator Registration Authority	RegistryAdministrator
Registry Guest	RegistryGuest
Registry Reader	RegistryGuest

**Table 9: Default Actor to Role Mappings**

5964

5965

5966

## 5967 **Appendix A Web Service Architecture**

### 5968 **A.1 Registry Service Abstract Specification**

5969 The normative definition of the Abstract Registry Service in WSDL is defined at the following  
5970 location on the web:

5971 <http://www.oasis-open.org/committees/regrep/documents/2.5/services/Registry.wsdl>

### 5972 **A.2 Registry Service SOAP Binding**

5973 The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined  
5974 at the following location on the web:

5975 <http://www.oasis-open.org/committees/regrep/documents/2.5/services/RegistrySOAPBinding.wsdl>

5976

## 5977 **Appendix B ebXML Registry Schema Definitions**

### 5978 **B.1 RIM Schema**

5979 The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the  
5980 following location on the web:

5981 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rim.xsd>

### 5982 **B.2 Registry Services Interface Base Schema**

5983 The normative XML Schema definition that defines the XML requests and responses supported  
5984 by the registry service interfaces in this document can be found at the following location on the  
5985 web:

5986 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rs.xsd>

### 5987 **B.3 QueryManager Service Schema**

5988 The normative XML Schema definition for the XML syntax for the QueryManager service  
5989 interface can be found at the following location on the web:

5990 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/query.xsd>

### 5991 **B.4 LifecycleManager Service Schema**

5992 The normative XML Schema definition for the XML syntax for the LifecycleManager service  
5993 interface can be found at the following location on the web:

5994 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/lcm.xsd>

### 5995 **B.5 Content Management Service Schema**

5996 The normative XML Schema definition for the XML syntax for the Content Management  
5997 Services interface can be found at the following location on the web:

5998 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/cms.xsd>

### 5999 **B.6 Examples of Instance Documents**

6000 A growing number of non-normative XML instance documents that conform to the normative  
6001 Schema definitions described earlier may be found at the following location on the web:

6002 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/>

6003

## 6004 **Appendix C Interpretation of UML Diagrams**

6005 This section describes in *abstract terms* the conventions used to define ebXML business process  
6006 description in UML.

### 6007 **C.1 UML Class Diagram**

6008 A UML class diagram is used to describe the Service Interfaces required to implement an  
6009 ebXML Registry Services and clients. The UML class diagram contains:

6010

- 6011 1. A collection of UML interfaces where each interface represents a Registry Service  
6012 interface.
- 6013 2. Tabular description of methods on each interface where each method represents an  
6014 Action (as defined by [ebCPP]) within the Service Interface.
- 6015 3. One or more parameters for each method. The type of each parameter represents the  
6016 ebXML message type that is exchanged as part of the Action corresponding to the  
6017 method. Multiple arguments imply multiple payload documents within the body of the  
6018 corresponding ebXML message.

### 6019 **C.2 UML Sequence Diagram**

6020 A UML sequence diagram is used to specify the business protocol representing the interactions  
6021 between the UML interfaces for a Registry specific ebXML business process. A UML sequence  
6022 diagram provides the necessary information to determine the sequencing of messages and request  
6023 to response association as well as request to error response association.

6024 Each sequence diagram shows the sequence for a specific conversation protocol as method calls  
6025 from the requestor to the responder. Method invocation may be synchronous or asynchronous  
6026 based on the UML notation used on the arrowhead for the link. A half arrowhead represents  
6027 asynchronous communication. A full arrowhead represents synchronous communication.

6028 Each method invocation may be followed by a response method invocation from the responder to  
6029 the requestor to indicate the ResponseName for the previous Request. Possible error response is  
6030 indicated by a conditional response method invocation from the responder to the requestor. See  
6031 Figure 15 on page 39 for an example.

## 6032 Appendix D SQL Query

### 6033 D.1 SQL Query Syntax Specification

6034 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The  
6035 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax  
6036 conforms to the <query specification> as well as the additional restrictions identified below:

- 6037 1. A <select list> may contain at most one <select sublist>.
- 6038 2. The <select list> must be is a single column whose data type is UUID, from the table in the  
6039 <from clause>.
- 6040 3. A <derived column> may not have an <as clause>.
- 6041 4. A <table expression> does not contain the optional <group by clause> and <having  
6042 clause> clauses.
- 6043 5. A <table reference> can only consist of <table name> and <correlation name>.
- 6044 6. A <table reference> does not have the optional AS between <table name> and  
6045 <correlation name>.
- 6046 7. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows  
6047 for the right hand side of the <in predicate> to be limited to a restricted <query  
6048 specification> as defined above.
- 6049 8. The SQL query syntax allows for the use of <sql invoked routines> invocation from  
6050 [SQL/PSM] as the RHS of the <in predicate>.

### 6051 D.2 Non-Normative BNF for Query Syntax Grammar

6052 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as  
6053 an aid to implementers. Since this BNF is not based directly on [SQL] it is provided as non-  
6054 normative syntax. For the normative syntax rules see Appendix D.1.

6055

```

sqlQuery ::= SQLSelect ( <SEMICOLON> )? <EOF>
SQLSelect ::= <SELECT> SQLSelectCols <FROM> SQLTableList (
    SQLWhere )? ( SQLOrderBy )?
SQLSelectCols ::= ( <ALL> | <DISTINCT> )* ( ( "*" | SQLvalueTerm ) )
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> "(" SQLValueListOrProcedureCall ")"  
 SQLValueListOrProcedureCall ::= ( ProcedureCall | SQLValueList )  
 ProcedureCall ::= <ID> "(" <STRING\_LITERAL> ")"  
 SQLValueList ::= SQLValueElement ( "," SQLValueElement )  
 SQLValueElement ::= ( <NULL> | SQLSumExpr | SQLSelect )  
 SQLIsClause ::= SQLColRef <IS> ( <NOT> )? <NULL>  
 SQLLikeClause ::= ( <NOT> )? <LIKE> SQLPattern  
 SQLPattern ::= ( <STRING\_LITERAL> )  
 SQLColRef ::= SQLvalue  
 SQLvalue ::= ( SQLvalueTerm )  
 SQLvalueTerm ::= <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> )

6056

### 6057 D.3 Relational Schema For SQL Queries

6058 The normative Relational Schema definition for SQL queries can be found at the following  
 6059 location on the web:

6060 <http://www.oasis-open.org/committees/regrep/documents/2.5/sql/database.sql>

6061

6062 The stored procedures that must be supported by the SQL query feature are defined at the following  
 6063 location on the web:

6064 <http://www.oasis-open.org/committees/regrep/documents/2.5/sql/storedProcedures.sql>

## 6065 **Appendix E Security Implementation Guideline**

6066 This section provides a suggested blueprint for how security processing may be implemented in  
6067 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have  
6068 different implementations as long as they support the default security roles and authorization  
6069 rules described in this document.

### 6070 **E.1 Security Concerns**

6071 The security risks broadly stem from the following concerns. After a description of these  
6072 concerns and potential solutions, we identify the concerns that we address in the current  
6073 specification

- 6074 1. Is the content of the registry (data) trustworthy?
  - 6075 a) How to make sure “what is in the registry” is “what is put there” by the ContentOwner?  
6076 This concern can be addressed by ensuring that the publisher is authenticated using  
6077 digital signature (Source Integrity), message is not corrupted during transfer using digital  
6078 signature (Data Integrity), and the data is not altered by unauthorized subjects based on  
6079 access control policy (Authorization)
  - 6080 b) How to protect data while in transmission?  
6081 Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data  
6082 Confidentiality that can be addressed by encrypting the data in transmission. How to  
6083 protect against a replay attack?
  - 6084 c) Is the content up to date? The versioning as well as any time stamp processing, when  
6085 done securely will ensure the “latest content” is guaranteed to be the latest content.
  - 6086 d) How to ensure only bona fide responsible organizations add contents to registry?  
6087 Ensuring Source Integrity (as in 1a).
  - 6088 e) How to ensure that bona fide publishers add contents to registry only at authorized  
6089 locations? (System Integrity)
  - 6090 f) What if the publishers deny modifying certain content after-the-fact? To prevent this  
6091 (Nonrepudiation) audit trails may be kept which contain signed message digests.
  - 6092 g) What if the reader denies getting information from the registry?
- 6093 2. How to provide selective access to registry content? The broad answer is, by using an access  
6094 control policy – applies to (a), (b), and (c) directly.
  - 6095 a) How does a ContentOwner restrict access to the content to only specific registry readers?
  - 6096 b) How can a ContentOwner allow some “partners” (fellow publishers) to modify content?
  - 6097 c) How to provide selective access to partners the registry usage data?
  - 6098 d) How to prevent accidental access to data by unauthorized users? Especially with  
6099 hardware or software failure of the registry security components? The solution to this  
6100 problem is by having System Integrity.
  - 6101 e) Data confidentiality of RegistryObject

- 6102 3. How do we make “who can see what” policy itself visible to limited parties, even excluding  
6103 the administrator (self & confidential maintenance of access control policy). By making sure  
6104 there is an access control policy for accessing the policies themselves.
- 6105 4. How to transfer credentials? The broad solution is to use credentials assertion (such as being  
6106 worked on in Security Assertions Markup Language (SAML)). Currently, Registry does not  
6107 support the notion of a session. Therefore, some of these concerns are not relevant to the  
6108 current specification.
- 6109 a) How to transfer credentials (authorization/authentication) to federated registries?  
6110 b) How do aggregators get credentials (authorization/authentication) transferred to them?  
6111 c) How to store credentials through a session?

## 6112 **E.2 Authentication**

- 6113 1. As soon as a message is received, a User object is created.
- 6114 2. If the message is signed, it is verified (including the validity of the certificate) and the DN of  
6115 the certificate becomes the identity of the User.
- 6116 3. If the message is not signed, a User instance is created with the role RegistryGuest. This step  
6117 is suggested for symmetry and to decouple the rest of the processing.
- 6118 4. The message is then processed for the Action and the objects it will act on.

## 6119 **E.3 Authorization**

6120 For every RegistryObject resource, the Policy Decision Point as defined by [XACML] within the  
6121 registry will process the AccessControlPolicy object associated with the RegistryObject resource  
6122 to verify that the requestor subject (e.g. User) is permitted to perform the requested action (e.g.  
6123 create, update, delete) on the specified resource.

## 6124 **E.4 Registry Bootstrap**

6125 When a Registry is newly created, a default User object should be created with the identity of the  
6126 Registry Administrator’s certificate DN with a role RegistryAdministrator. This way, any  
6127 message signed by the RegistryAdministrator will get all the privileges.

6128 When a Registry is newly created, an instance of AccessControlPolicy is created as the default  
6129 AccessControlPolicy.

## 6130 **E.5 Content Submission – Client Responsibility**

6131 The Registered User must sign the contents before submission – otherwise the content will be  
6132 rejected.

## 6133 **E.6 Content Submission – Registry Responsibility**

- 6134 1. As with any other request, the Registry Client will first be authenticated. In this case, the  
6135 User object will get the DN from the certificate.
- 6136 2. As per the request in the message, the RegistryObject will be created.

- 6137 3. The RegistryObject is assigned the default AccessControlPolicy.  
6138 4. If the Registry Client is not previously registered, the registry may either reject the request or  
6139 accept it and implicitly register the Registry Client.

6140

## 6141 **E.7 Content Remove/Deprecate – Client Responsibility**

6142 The Registry client must sign the header before submission, for authentication purposes;  
6143 otherwise, the request will be rejected

## 6144 **E.8 Content Remove/Deprecate – Registry Responsibility**

- 6145 1. As with any other request, the Registry Client will first be authenticated. In this case, the  
6146 User object will get the DN from the certificate.
- 6147 2. As per the request in the message (remove or deprecate), the appropriate method in the  
6148 RegistryObject class will be accessed.
- 6149 3. The access controller performs the authorization by iterating through the Permission objects  
6150 associated with this object via the default AccessControlPolicy.
- 6151 4. If authorization succeeds then the action will be permitted. Otherwise an error response is  
6152 sent back with a suitable AuthorizationException error message.

## 6153 **E.9 Using ds:KeyInfo Field**

6154 Two typical usage scenarios for ds:KeyInfo are described below.

### 6155 **Scenario 1**

- 6156 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.  
6157 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.  
6158 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.  
6159 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature.  
6160 5. Registration Authority validates the header signature using the public key from the  
6161 certificate.
- 6162 6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the  
6163 certificate from the KeyInfo field of the payload signature. Note that this step is not an  
6164 essential one if the onus of validation is that of the eventual user, another Registry Client, of  
6165 the content.

### 6166 **Scenario 2**

- 6167 1. RC1 signs the payload and SOAP envelope using its private key and publishes to the  
6168 Registry.
- 6169 2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.

- 6170 3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature.  
6171 This step is required in addition to step 2 because when RC2 retrieves content, it should see  
6172 RC1's signature with the payload.
- 6173 4. RC2 retrieves content from the Registry.
- 6174 5. Registration Authority signs the SOAP envelope using its private key. Registration Authority  
6175 sends RC1's content and the RC1's signature (signed by RC1).
- 6176 6. Registration Authority need not send its certificate in the KeyInfo field since RC2 is assumed  
6177 to have obtained the Registration Authority's certificate in an implementation-specific  
6178 manner and installed it in its local key store.
- 6179 7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the  
6180 Registration Authority's signature.
- 6181 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates  
6182 the signature on the payload.

## 6183 **Appendix F Native Language Support (NLS)**

### 6184 **F.1 Definitions**

6185 Although this section discusses only character set and language, the following terms have to be  
6186 defined clearly.

#### 6187 **F.1.1 Coded Character Set (CCS):**

6188 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of  
6189 CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

#### 6190 **F.1.2 Character Encoding Scheme (CES):**

6191 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are  
6192 ISO-2022, UTF-8.

#### 6193 **F.1.3 Character Set (charset):**

- 6194 • charset is a set of rules for mapping from a sequence of octets to a sequence of  
6195 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- 6196 • A list of registered character sets can be found at [IANA].

### 6197 **F.2 NLS And Request / Response Messages**

6198 For the accurate processing of data in both registry client and registry services, it is essential to  
6199 know which character set is used. Although the body part of the transaction may contain the  
6200 charset in xml encoding declaration, registry client and registry services shall specify charset  
6201 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a  
6202 text/xml entity is received with the charset parameter omitted, MIME processors and XML  
6203 processors MUST use the default charset value of "us-ascii". For example:

```
6204 Content-Type: text/xml; charset=ISO-2022-JP  
6205  
6206
```

6207 Also, when an application/xml entity is used, the charset parameter is optional, and registry  
6208 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which  
6209 directly address this contingency.

6210 If another Content-Type is used, then usage of charset must follow [RFC 3023].

### 6211 **F.3 NLS And Storing of RegistryObject**

6212 This section provides NLS guidelines on how a registry should store RegistryObject instances.

6213 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple  
6214 locales. Thus there is no language or character set associated with a specific RegistryObject  
6215 instance.

6216 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows.  
6217 Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in

6218 RegistryObject class) as defined by [eBRIM], may have multiple locale specific values expressed  
6219 as LocalizedString sub-elements within the XML element representing the I18N capable  
6220 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a  
6221 specific locale. Each LocalizedString element has a charset and lang attribute as well as a value  
6222 attribute of type string.

### 6223 **F.3.1 Character Set of *LocalizedString***

6224 The character set used by a locale specific String (*LocalizedString*) is defined by the charset  
6225 attribute. It is highly recommended to use UTF-8 or UTF-16 for maximum interoperability.

### 6226 **F.3.2 Language Information of *LocalizedString***

6227 The language may be specified in `xml:lang` attribute (Section 2.12 [REC-XML]).

## 6228 **F.4 NLS And Storing of Repository Items**

6229 This section provides NLS guidelines on how a registry should store repository items.  
6230 While a single instance of an *ExtrinsicObject* is capable of supporting multiple locales, it is  
6231 always associated with a single repository item. The repository item may be in a single locale or  
6232 may be in multiple locales. This specification does not specify the repository item.

### 6233 **F.4.1 Character Set of Repository Items**

6234 The MIME *Content-Type* mime header for the mime multipart containing the repository item  
6235 MAY contain a *charset* attribute that specifies the character set used by the repository item. For  
6236 example:

```
6237 Content-Type: text/xml; charset="UTF-8"
```

6240 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset  
6241 of a repository item must be preserved as it is originally specified in the transaction.

### 6242 **F.4.2 Language information of repository item**

6243 The Content-language mime header for the mime bodypart containing the repository item may  
6244 specify the language for a locale specific repository item. The value of the Content-language  
6245 mime header property must conform to [RFC 1766].

6246 This document currently specifies only the method of sending the information of character set  
6247 and language, and how it is stored in a registry. However, the language information may be used  
6248 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a  
6249 language negotiation procedure, like registry client is asking a favorite language for messages  
6250 from registry services, could be another functionality for the future revision of this document.

6251 **13 References**

- 6252 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 6253 [ebRIM] ebXML Registry Information Model version 2.5
- 6254 <http://www.oasis-open.org/committees/regrep/documents/2.5/specs/ebRIM.pdf>
- 6255 [ebRIM Schema] ebXML Registry Information Model Schema
- 6256 <http://www.oasis-open.org/committees/regrep/documents/2.5/schema/rim.xsd>
- 6257 [ebBPSS] ebXML Business Process Specification Schema
- 6258 <http://www.ebxml.org/specs>
- 6259 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 6260 <http://www.ebxml.org/specs/>
- 6261 [ebMS] ebXML Messaging Service Specification, Version 1.0
- 6262 <http://www.ebxml.org/specs/>
- 6263 [XPT] XML Path Language (XPath) Version 1.0
- 6264 <http://www.w3.org/TR/xpath>
- 6265 [SQL] Structured Query Language (FIPS PUB 127-2)
- 6266 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 6267 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 6268 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 6269 [IANA] IANA (Internet Assigned Numbers Authority).
- 6270 Official Names for Character Sets, ed. Keld Simonsen et al.
- 6271 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 6272 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
- 6273 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
- 6274 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 6275 [RFC 2119] IETF (Internet Engineering Task Force). RFC 2119
- 6276 [RFC 2130] IETF (Internet Engineering Task Force). RFC 2130
- 6277 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
- 6278 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
- 6279 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 6280 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
- 6281 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
- 6282 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 6283 [RFC2616] RFC 2616:
- 6284 Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1*. 1999.
- 6285 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 6286 [RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
- 6287 Internet Security Glossary, ed. R. Shirey. May 2000.
- 6288 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html>
- 6289 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
- 6290 XML Media Types, ed. M. Murata. 2001.
- 6291 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
- 6292 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
- 6293 <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](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
6296 <http://www.opengroup.org/publications/catalog/c706.htm><http://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-xmlsig-core-20010820/>

## 6305 **14 Disclaimer**

6306 The views and specification expressed in this document are those of the authors and are not  
6307 necessarily those of their employers. The authors and their employers specifically disclaim  
6308 responsibility for any problems arising from correct or incorrect implementation or use of this  
6309 design.

**6310 15 Contact Information****6311 Team Leader**

6312 Name: Kathryn R. Breininger  
6313 Company: The Boeing Company  
6314 Street: P.O. Box 3707 MC 62-LC  
6315 City, State, Postal Code: Seattle, WA 98124-2207  
6316 Country: USA  
6317 Phone: 425-965-0182  
6318 Email: [kathryn.r.breininger@boeing.com](mailto:kathryn.r.breininger@boeing.com)

6319

**6320 Editor**

6321 Name: Sally Fuger  
6322 Company: Automotive Industry Action Group  
6323 Street: 26200 Lahser Road, Suite 200  
6324 City, State, Postal Code: Southfield, MI 48034  
6325 Country: USA  
6326 Phone: (248) 358-9744  
6327 Email: [sfuger@aiag.org](mailto:sfuger@aiag.org)

6328

**6329 Technical Editor**

6330 Name: Farrukh S. Najmi  
6331 Company: Sun Microsystems  
6332 Street: 1 Network Dr., MS BUR02-302  
6333 City, State, Postal Code: Burlington, MA, 01803-0902  
6334 Country: USA  
6335 Phone: (781) 442-9017  
6336 Email: [farrukh.najmi@sun.com](mailto:farrukh.najmi@sun.com)

6337

**6338 Technical Editor**

6339 Name: Nikola Stojanovic  
6340 Company: Metaspaces Consulting  
6341 Street: 101 Pineview Terrace  
6342 City, State, Postal Code: Ithaca, NY, 14850  
6343 Country: USA  
6344 Phone: (607) 273-2224  
6345 Email: [nikola.stojanovic@acm.org](mailto:nikola.stojanovic@acm.org)

6346

## 6347 **16 Copyright Statement**

6348 Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

6349 **Copyright (C) The Organization for the Advancement of Structured Information**  
6350 **Standards [OASIS], 2002. All Rights Reserved.**

6351 This document and translations of it may be copied and furnished to others, and derivative works  
6352 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
6353 published and distributed, in whole or in part, without restriction of any kind, provided that the  
6354 above copyright notice and this paragraph are included on all such copies and derivative works.  
6355 However, this document itself may not be modified in any way, such as by removing the  
6356 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
6357 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
6358 Property Rights document must be followed, or as required to translate it into languages other  
6359 than English.

6360 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
6361 successors or assigns.

6362 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
6363 **DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT**  
6364 **LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN**  
6365 **WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF**  
6366 **MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

6367 OASIS takes no position regarding the validity or scope of any intellectual property or other  
6368 rights that might be claimed to pertain to the implementation or use of the technology described  
6369 in this document or the extent to which any license under such rights might or might not be  
6370 available; neither does it represent that it has made any effort to identify any such rights.  
6371 Information on OASIS's procedures with respect to rights in OASIS specifications can be found  
6372 at the OASIS website. Copies of claims of rights made available for publication and any  
6373 assurances of licenses to be made available, or the result of an attempt made to obtain a general  
6374 license or permission for the use of such proprietary rights by implementers or users of this  
6375 specification, can be obtained from the OASIS Executive Director.

6376 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
6377 applications, or other proprietary rights which may cover technology that may be required to  
6378 implement this specification. Please address the information to the OASIS Executive Director.