



Creating A Single Global Electronic Market

1

OASIS/ebXML Registry Services Specification v2.2 **–Committee Working Draft**

OASIS/ebXML Registry Technical Committee

September 2002

2 *This page intentionally left blank.*

3 **1 Status of this Document**

4 This document is an OASIS Registry Technical Committee Working Draft - September 2002.

5 Distribution of this document is unlimited.

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

7 ***This version:***

8 <http://www.oasis-open.org/committees/regrep/documents/2.2/specs/ebrs.pdf>

9

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

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

12

13 ***Latest OASIS Approved Standard:***

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

15

16 **2 OASIS/ebXML Registry Technical Committee**

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

19 Zachary Alexander, Individual Member

20 John Bekisz, Software AG, Inc.

21 Kathryn Breininger, Boeing

22 Lisa Carnahan, NIST

23 Joseph M. Chiusano, LMI

24 Suresh Damodaran, Sterling Commerce

25 Fred Federlein, Sun Microsystems

26 Sally Fuger, Individual Member

27 Michael Kass, NIST

28 Kyu-Chul Lee, Individual Member

29 Matthew MacKenzie, XML Global

30 Komal Mangtani, BEA Systems

31 Monica Martin, Drake Certivo, Inc.

32 Farrukh Najmi, Sun Microsystems

33 Sanjay Patil, IONA

34 Nikola Stojanovic, Individual Member

35 Scott Zimmerman, Storagepoint

36 Contributors

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

39 Anne Fischer, Individual

40 Len Gallagher, NIST

41 Sekhar Vajjhala, Sun Microsystems

42

43	Table of Contents	
44	1 Status of this Document	3
45	2 OASIS/ebXML Registry Technical Committee	4
46	Table of Contents	5
47	Table of Figures	11
48	Table of Tables	13
49	3 Introduction	14
50	3.1 Summary of Contents of Document	14
51	3.2 General Conventions	14
52	3.3 Audience	14
53	4 Design Objectives	15
54	4.1 Goals	15
55	4.2 Caveats and Assumptions	15
56	5 System Overview	16
57	5.1 What The ebXML Registry Does	16
58	5.2 How The ebXML Registry Works	16
59	5.2.1 Schema Documents Are Submitted	16
60	5.2.2 Business Process Documents Are Submitted	16
61	5.2.3 Seller's Collaboration Protocol Profile Is Submitted	16
62	5.2.4 Buyer Discovers The Seller	16
63	5.2.5 CPA Is Established	17
64	5.3 Registry Users	17
65	5.4 Where the Registry Services May Be Implemented	18
66	5.5 Implementation Conformance	18
67	5.5.1 Conformance as an ebXML Registry	18
68	5.5.2 Conformance as an ebXML Registry Client	19
69	6 ebXML Registry Architecture	20
70	6.1 Registry Service Described	20
71	6.2 Abstract Registry Service	21
72	6.2.1 LifeCycleManager Interface	21
73	6.2.2 QueryManager Interface	22
74	6.3 Concrete Registry Services	22
75	6.4 SOAP Binding	23
76	6.4.1 WSDL Terminology Primer	23
77	6.4.2 Concrete Binding for SOAP	23
78	6.5 ebXML Message Service Binding	24
79	6.5.1 Service and Action Elements	24
80	6.5.2 Synchronous and Asynchronous Responses	24
81	6.5.3 ebXML Registry Collaboration Profiles and Agreements	25
82	6.6 REST Binding	25
83	6.6.1 Standard URI Parameters	26
84	6.6.2 QueryManager REST Interface	26
85	6.6.3 LifecycleManager REST Interface	28
86	6.6.4 Security Considerations	29
87	6.6.5 Exception Handling	29
88	6.7 Registry Clients	29

89	6.7.1	Registry Client Described	29
90	6.7.2	Registry Communication Bootstrapping	30
91	6.7.3	RegistryClient Interface	31
92	6.7.4	Registry Response	31
93	6.8	Interoperability Requirements	31
94	6.8.1	Client Interoperability.....	31
95	6.8.2	Inter-Registry Cooperation.....	32
96	7	Life Cycle Management Service	33
97	7.1	Life Cycle of a Repository Item	33
98	7.2	RegistryObject Attributes	33
99	7.3	The Submit Objects Protocol.....	34
100	7.3.1	SubmitObjectsRequest.....	34
101	7.3.2	RegistryResponse	35
102	7.3.3	Universally Unique ID Generation.....	36
103	7.3.4	ID Attribute And Object References	36
104	7.3.5	Audit Trail	37
105	7.3.6	Error Handling	37
106	7.3.7	Sample SubmitObjectsRequest.....	37
107	7.4	The Update Objects Protocol.....	40
108	7.4.1	Audit Trail	41
109	7.5	The Add Slots Protocol.....	41
110	7.5.1	AddSlotsRequest.....	42
111	7.6	The Remove Slots Protocol	43
112	7.6.1	RemoveSlotsRequest	43
113	7.7	The Approve Objects Protocol	44
114	7.7.1	ApproveObjectsRequest	45
115	7.7.2	Audit Trail	46
116	7.8	The Deprecate Objects Protocol.....	46
117	7.8.1	DeprecateObjectsRequest	46
118	7.8.2	Audit Trail	47
119	7.9	The Remove Objects Protocol.....	47
120	7.9.1	RemoveObjectsRequest.....	48
121	8	Query Management Service.....	50
122	8.1	Ad Hoc Query Request/Response	50
123	8.1.1	AdhocQueryRequest	51
124	8.1.2	AdhocQueryResponse	52
125	8.1.3	ReponseOption	53
126	8.1.4	Iterative Query Support	54
127	8.2	Filter Query Support	55
128	8.2.1	FilterQuery.....	56
129	8.2.2	RegistryObjectQuery	58
130	8.2.3	RegistryEntryQuery.....	71
131	8.2.4	AssociationQuery	74
132	8.2.5	AuditableEventQuery	76
133	8.2.6	ClassificationQuery	79
134	8.2.7	ClassificationNodeQuery.....	81
135	8.2.8	ClassificationSchemeQuery.....	86
136	8.2.9	RegistryPackageQuery	87

137	8.2.10	ExtrinsicObjectQuery	89
138	8.2.11	OrganizationQuery	90
139	8.2.12	ServiceQuery	94
140	8.2.13	Registry Filters.....	96
141	8.2.14	XML Clause Constraint Representation.....	100
142	8.3	SQL Query Support	104
143	8.3.1	SQL Query Syntax Binding To [ebRIM]	105
144	8.3.2	Semantic Constraints On Query Syntax.....	106
145	8.3.3	SQL Query Results	107
146	8.3.4	Simple Metadata Based Queries	107
147	8.3.5	RegistryObject Queries.....	107
148	8.3.6	RegistryEntry Queries	107
149	8.3.7	Classification Queries	108
150	8.3.8	Association Queries	109
151	8.3.9	Package Queries.....	110
152	8.3.10	ExternalLink Queries	110
153	8.3.11	Audit Trail Queries	110
154	8.4	Content Retrieval.....	110
155	8.4.1	Identification Of Content Payloads	111
156	8.4.2	GetContentResponse Message Structure	111
157	9	Content-based Discovery.....	113
158	9.1	Content-based Discovery: Use Cases	113
159	9.1.1	Find All CPPs Where Role is “Buyer”	113
160	9.1.2	Find All XML Schema’s That Use Specified Namespace	113
161	9.1.3	Find All WSDL Descriptions with a SOAP Binding	113
162	9.2	Content Indexing Service	113
163	9.2.1	Illustrative Example	114
164	9.3	Index Definition File.....	115
165	9.4	Indexable Content	115
166	9.5	Index Metadata	116
167	9.6	Content Indexing Protocol.....	116
168	9.6.1	IndexContentRequest.....	116
169	9.6.2	IndexContentResponse	117
170	9.7	Publishing a Content Indexing Service.....	118
171	9.7.1	Multiple Indexers and Index Definition Files.....	119
172	9.7.2	Restrictions On Publishing Content Indexing Services	120
173	9.8	Dynamic Content Indexing.....	120
174	9.8.1	Threading Model for Dynamic Content Indexing	120
175	9.8.2	Referential Integrity and Dynamic Content Indexing	120
176	9.8.3	Error Handling Model for Dynamic Content Indexing	120
177	9.8.4	Updates and Dynamic Content Indexing.....	121
178	9.8.5	Resolution Algorithm For Indexer and Index Definition File	121
179	9.9	Dynamic Content-based Discovery.....	121
180	9.10	Default XML Content Indexer.....	122
181	9.10.1	Publishing of Default XML Content Indexer	122
182	9.11	Canonical Index Definition Files	122
183	10	Event Notification.....	123
184	10.1	Use Cases	123

185	10.1.1	New Service is Offered.....	123
186	10.1.2	Monitor Download of Content.....	123
187	10.1.3	Monitor Price Changes.....	123
188	10.1.4	Keep Replicas Consistent With Source Object.....	123
189	10.2	Registry Events.....	123
190	10.3	Subscribing to Events.....	124
191	10.3.1	Event Selection.....	124
192	10.3.2	Notification Action.....	124
193	10.3.3	Subscription Authorization.....	124
194	10.3.4	Subscription Quotas.....	124
195	10.3.5	Subscription Expiration.....	124
196	10.4	Unsubscribing from Events.....	125
197	10.5	Notification of Events.....	125
198	10.6	Retrieval of Events.....	125
199	10.6.1	GetNotificationsRequest.....	125
200	10.6.2	GetNotificationsResponse.....	126
201	10.7	Event Management Policies.....	126
202	10.8	Notes	127
203	11	Cooperating Registries Support.....	128
204	11.1	Cooperating Registries Use Cases.....	128
205	11.1.1	Inter-registry Object References.....	128
206	11.1.2	Federated Queries.....	128
207	11.1.3	Local Caching of Data from Another Registry.....	128
208	11.1.4	Object Relocation.....	129
209	11.2	Registry Federations.....	129
210	11.2.1	Federation Metadata.....	129
211	11.2.2	Local Vs. Federated Queries.....	130
212	11.2.3	Federated Life Cycle Management Operations.....	130
213	11.2.4	Federations and Local Caching of Remote Data.....	131
214	11.2.5	Caching of Federation Metadata.....	131
215	11.2.6	Time Synchronization Between Registry Peers.....	131
216	11.2.7	Federations and Security.....	131
217	11.2.8	Federation Life Cycle Management Protocols.....	131
218	11.3	Object Replication.....	132
219	11.3.1	Use Cases for Object Replication.....	133
220	11.3.2	Queries And Replicas.....	133
221	11.3.3	Lifecycle Operations And Replicas.....	133
222	11.3.4	Object Replication and Federated Registries.....	134
223	11.3.5	Creating a Local Replica.....	134
224	11.3.6	Transactional Replication.....	134
225	11.3.7	Keeping Replicas Current.....	134
226	11.3.8	Write Operations on Local Replica.....	134
227	11.3.9	Tracking Location of a Replica.....	135
228	11.3.10	Remote Object References to a Replica.....	135
229	11.3.11	Removing a Local Replica.....	135
230	11.4	Object Relocation Protocol.....	135
231	11.4.1	RelocateObjectsRequest.....	138
232	11.4.2	AcceptObjectsRequest.....	139

233	11.4.3	Object Relocation and Remote ObjectRefs	139
234	11.4.4	Notification of Object Relocation.....	140
235	11.4.5	Object Relocation and Timeouts	140
236	12	Registry Security.....	141
237	12.1	Security Concerns	141
238	12.2	Integrity of Registry Content	141
239	12.2.1	Message Payload Signature	141
240	12.2.2	Payload Signature Requirements	142
241	12.3	Authentication.....	143
242	12.3.1	Message Header Signature	144
243	12.4	Key Distribution and KeyInfo Element	145
244	12.5	Confidentiality	146
245	12.5.1	On-the-wire Message Confidentiality.....	146
246	12.5.2	Confidentiality of Registry Content	146
247	12.6	Authorization.....	146
248	12.6.1	Actions	147
249	12.7	Access Control.....	147
250	Appendix A	Web Service Architecture	149
251	A.1	Registry Service Abstract Specification.....	149
252	A.2	Registry Service SOAP Binding.....	149
253	Appendix B	ebXML Registry Schema Definitions	150
254	B.1	RIM Schema	150
255	B.2	Query Schema.....	150
256	B.3	Registry Services Interface Schema	150
257	B.4	Examples of Instance Documents.....	150
258	Appendix C	Interpretation of UML Diagrams	151
259	C.1	UML Class Diagram.....	151
260	C.2	UML Sequence Diagram	151
261	Appendix D	SQL Query	152
262	D.1	SQL Query Syntax Specification.....	152
263	D.2	Non-Normative BNF for Query Syntax Grammar	152
264	D.3	Relational Schema For SQL Queries.....	154
265	Appendix E	Security Implementation Guideline	155
266	E.1	Security Concerns	155
267	E.2	Authentication.....	156
268	E.3	Authorization.....	156
269	E.4	Registry Bootstrap	156
270	E.5	Content Submission – Client Responsibility	156
271	E.6	Content Submission – Registry Responsibility	157
272	E.7	Content Delete/Deprecate – Client Responsibility.....	157
273	E.8	Content Delete/Deprecate – Registry Responsibility	157
274	E.9	Using ds:KeyInfo Field	157
275	Appendix F	Native Language Support (NLS)	159
276	F.1	Definitions	159
277	F.1.1	Coded Character Set (CCS):.....	159
278	F.1.2	Character Encoding Scheme (CES):	159
279	F.1.3	Character Set (charset):	159

280	F.2	NLS And Request / Response Messages	159
281	F.3	NLS And Storing of RegistryObject.....	159
282	F.3.1	Character Set of <i>LocalizedString</i>	160
283	F.3.2	Language Information of <i>LocalizedString</i>	160
284	F.4	NLS And Storing of Repository Items	160
285	F.4.1	Character Set of Repository Items.....	160
286	F.4.2	Language information of repository item.....	160
287	Appendix G	Registry Profile	161
288	13	References.....	162
289	14	Disclaimer.....	164
290	15	Contact Information.....	165
291	16	Copyright Statement	166
292	17	Notes.....	167
293			

294 **Table of Figures**

295	☞ Figure 1: Actor Relationships	18
296	☞ Figure 2: ebXML Registry Service Architecture	20
297	☞ Figure 3: The Abstract ebXML Registry Service	21
298	☞ Figure 4: A Concrete ebXML Registry Service	23
299	☞ Figure 5: Registry Architecture Supports Flexible Topologies	30
300	☞ Figure 6: Life Cycle of a Repository Item.....	33
301	☞ Figure 7: Submit Objects Sequence Diagram.....	34
302	☞ Figure 8: SubmitObjectsRequest Syntax	34
303	☞ Figure 9: RegistryResponse Syntax.....	35
304	☞ Figure 10: Update Objects Sequence Diagram.....	41
305	☞ Figure 11: Add Slots Sequence Diagram.....	42
306	☞ Figure 12: AddSlotsRequest Syntax.....	42
307	☞ Figure 13: Remove Slots Sequence Diagram	43
308	☞ Figure 14: RemoveSlotsRequest Syntax.....	44
309	☞ Figure 15: Approve Objects Sequence Diagram	45
310	☞ Figure 16: ApproveObjectsRequest Syntax.....	45
311	☞ Figure 17: Deprecate Objects Sequence Diagram	46
312	☞ Figure 18: DeprecateObjectsRequest Syntax.....	46
313	☞ Figure 19: Remove Objects Sequence Diagram	48
314	☞ Figure 20: RemovalObjectsRequest Syntax	48
315	☞ Figure 21: Submit Ad Hoc Query Sequence Diagram.....	51
316	☞ Figure 22: AdhocQueryRequest Syntax	51
317	☞ Figure 23: AdhocQueryResponse Syntax.....	52
318	☞ Figure 24: ResponseOption Syntax	53
319	☞ Figure 25: Example ebRIM Binding.....	55
320	☞ Figure 26: ebRIM Binding for RegistryObjectQuery.....	58
321	☞ Figure 27: ebRIM Binding for RegistryEntryQuery.....	71
322	☞ Figure 28: ebRIM Binding for AssociationQuery	74
323	☞ Figure 29: ebRIM Binding for AuditableEventQuery	76
324	☞ Figure 30: ebRIM Binding for ClassificationQuery	79
325	☞ Figure 31: ebRIM Binding for ClassificationNodeQuery	81
326	☞ Figure 32: ebRIM Binding for ClassificationSchemeQuery	86
327	☞ Figure 33: ebRIM Binding for RegistryPackageQuery	87
328	☞ Figure 34: ebRIM Binding for ExtrinsicObjectQuery	89
329	☞ Figure 35: ebRIM Binding for OrganizationQuery	91
330	☞ Figure 36: ebRIM Binding for ServiceQuery.....	95
331	☞ Figure 37: The Clause Structure	100
332	☞ Figure 38: Abstract Content Indexing Service: Inputs and Outputs	114
333	☞ Figure 39: Example of CPP indexing using Default XML Indexer.....	115
334	☞ Figure 40: Content Indexing Protocol.....	116

335	☞	Figure 41: IndexContentRequest Syntax	117
336	☞	Figure 42: IndexContentResponse Syntax.....	118
337	☞	Figure 43: Indexing Service Configuration	119
338	☞	Figure 44: GetNotificationsRequest Syntax	125
339	☞	Figure 45: GetNotificationsResponse Syntax.....	126
340	☞	Figure 46: Inter-registry Object References.....	128
341	☞	Figure 47: Registry Federations	129
342	☞	Figure 48: Object Replication.....	133
343		Figure 49: Object Relocation.....	135
344	☞	Figure 50: Relocate Objects Protocol	137
345	☞	Figure 51: RelocateObjectsRequest XML Schema	138
346			

347 **Table of Tables**

348	☞ Table 1: Registry Users.....	17
349	☞ Table 2: LifeCycle Manager Summary.....	21
350	☞ Table 3: Query Manager	22
351	☞ Table 4: Standard URI Parameters.....	26
352	☞ Table 5: QueryManager REST Interface	26
353	☞ Table 6: LifecycleManager REST Interface	28
354	☞ Table 7: RegistryClient Summary	31
355	☞ Table 8: Path Filter Expressions for Use Cases	84
356	☞ Table 9: Role to Permissions Mapping	147
357	☞ Table 10: Default Actor to Role Mappings.....	148
358		

359 **3 Introduction**

360 **3.1 Summary of Contents of Document**

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

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

366 **3.2 General Conventions**

367 The following conventions are used throughout this document:

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

370 The term “repository item” is used to refer to an object that has resides in a repository for storage
371 and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the
372 Registry by a RegistryObject instance.

373 The term "RegistryEntry" is used to refer to an object that provides metadata about a repository
374 item.

375 Capitalized Italic words are defined in the ebXML Glossary.

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

379 **3.3 Audience**

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

381 ?? Implementers of ebXML Registry Services

382 ?? Implementers of ebXML Registry Clients

383 **Related Documents**

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

385 a) *ebXML Registry Information Model* [ebRIM]

386 b) *ebXML Message Service Specification* [ebMS]

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

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

389 **4 Design Objectives**

390 **4.1 Goals**

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

392 ?? Communicate functionality of Registry services to software developers

393 ?? Specify the interface for Registry clients and the Registry

394 ?? Provide a basis for future support of more complete ebXML Registry requirements

395 ?? Be compatible with other ebXML specifications

396 **4.2 Caveats and Assumptions**

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

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

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

407 **5 System Overview**

408 **5.1 What The ebXML Registry Does**

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

413 **5.2 How The ebXML Registry Works**

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

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

423 **5.2.1 Schema Documents Are Submitted**

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

427 **5.2.2 Business Process Documents Are Submitted**

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

431 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

436 **5.2.4 Buyer Discovers The Seller**

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

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

442 5.2.5 CPA Is Established

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

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

449 5.3 Registry Users

450 We describe the actors who use the registry below. Some of the actors are defined in Section
 451 12.7. Note that the same entity may represent different actors. For example, a Registration
 452 Authority and Registry Administrator may have the same identity.

453  **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

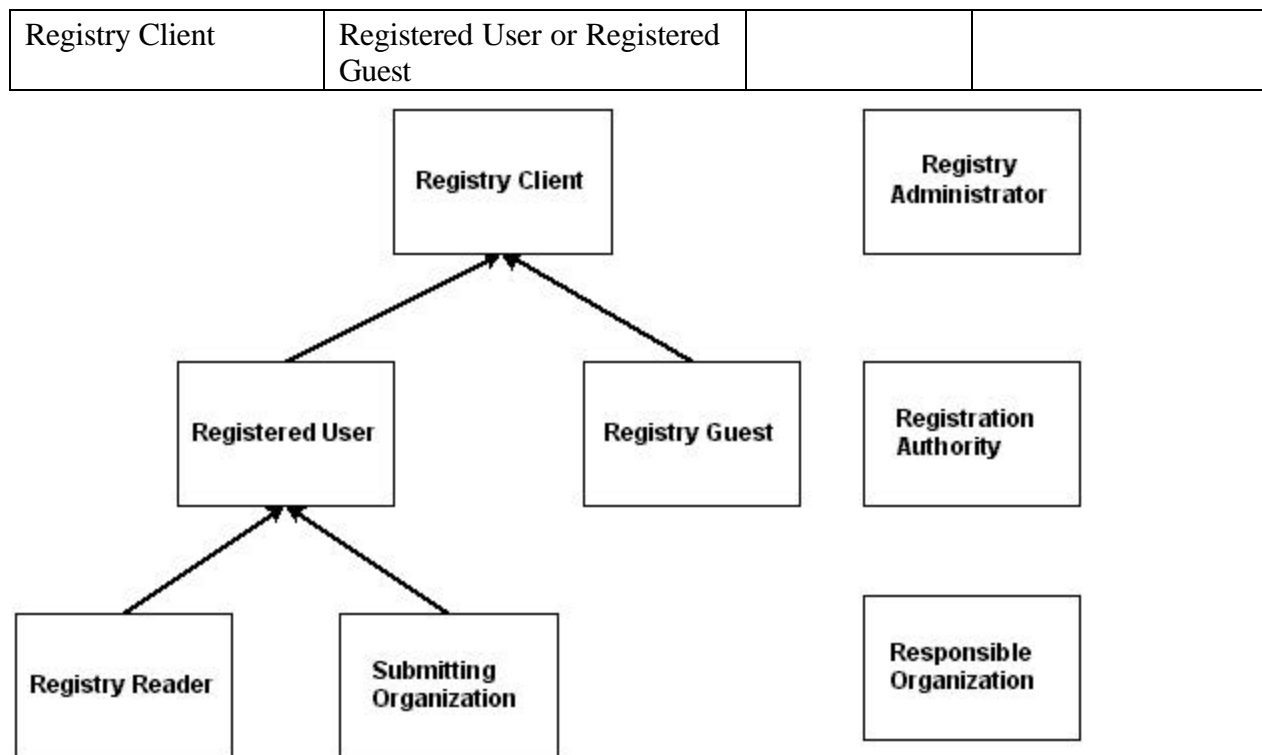


Figure 1: Actor Relationships

454
455

456 Note:
 457 In the current version of the specification the following are true.
 458 A Submitting Organization and a Responsible Organization are the same.
 459 Registration of a user happens out-of-band, i.e, by means not specified in this specification.
 460 A Registry Administrator and Registration Authority are the same.

461 **5.4 Where the Registry Services May Be Implemented**

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

464 **5.5 Implementation Conformance**

465 An implementation is a *conforming* ebXML Registry if the implementation meets the conditions
 466 in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the
 467 implementation meets the conditions in Section 5.5.2. An implementation is a conforming
 468 ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to
 469 the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming
 470 ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and
 471 Registry Client.

472 **5.5.1 Conformance as an ebXML Registry**

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

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

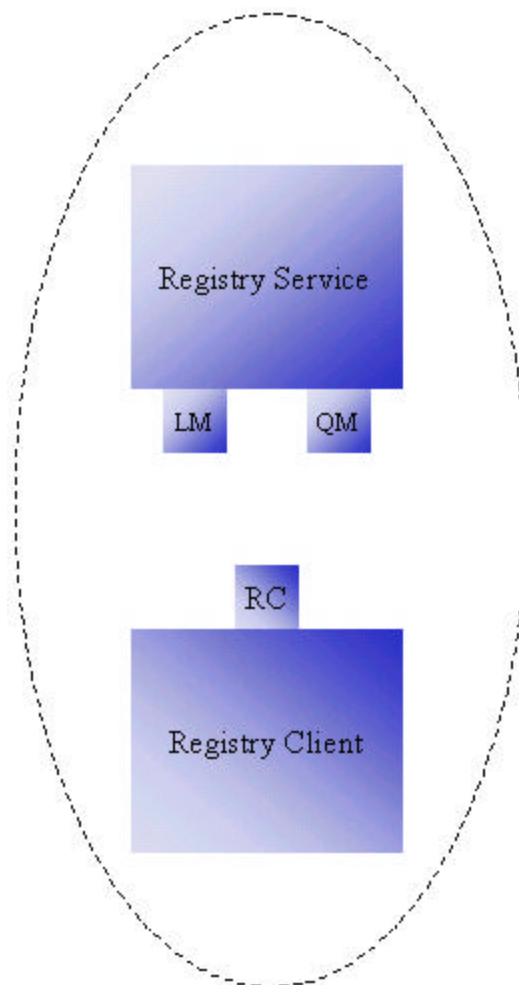
479 **5.5.2 Conformance as an ebXML Registry Client**

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

- 482 1. Supports the ebXML CPA and bootstrapping process.
483 2. Supports the syntax and the semantics of the Registry Client Interfaces.
484 3. Supports the defined ebXML Error Message DTD.
485 4. Supports the defined ebXML Registry Schema (Appendix B).
486

487 6 ebXML Registry Architecture

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



491
 492

☞☞ Figure 2: ebXML Registry Service Architecture

493 6.1 Registry Service Described

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

497 ?? A Life Cycle Management interface that provides a collection of methods for managing
 498 objects within the Registry.

499 ?? A Query Management Interface that controls the discovery and retrieval of information from
 500 the Registry.

501 A registry client program utilizes the services of the registry by invoking methods on one of the
 502 above interfaces defined by the Registry Service. This specification defines the interfaces
 503 exposed by the Registry Service (Sections **Error! Reference source not found.** and **Error!
 504 Reference source not found.**) as well as the interface for the Registry Client (Section 0).

505 6.2 Abstract Registry Service

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

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

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

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



516 **Figure 3: The Abstract ebXML Registry Service**

518 Appendix A provides hyperlinks to the abstract service definition in the Web Service Description
 519 Language (WSDL) syntax.

520 6.2.1 LifeCycleManager Interface

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

526 **Table 2: LifeCycle Manager Summary**

Method Summary of LifeCycleManager	
RegistryResponse	approveObjects (ApproveObjectsRequest req) Approves one or more previously submitted objects.
RegistryResponse	deprecateObjects (DeprecateObjectsRequest req) Deprecates one or more previously submitted objects.
RegistryResponse	removeObjects (RemoveObjectsRequest req) Removes one or more previously submitted objects from the Registry.

¹ Known as ObjectQueryManager in V1.0

² Known as ObjectManager in V1.0

RegistryResponse	submitObjects (SubmitObjectsRequest req) Submits one or more objects and possibly related metadata such as Associations and Classifications.
RegistryResponse	updateObjects (UpdateObjectsRequest req) Updates one or more previously submitted objects.
RegistryResponse	addSlots (AddSlotsRequest req) Add slots to one or more registry entries.
RegistryResponse	removeSlots (RemoveSlotsRequest req) Remove specified slots from one or more registry entries.

527 **6.2.2 QueryManager Interface**

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

531  **Table 3: Query Manager**

Method Summary of QueryManager	
RegistryResponse	submitAdhocQuery (AdhocQueryRequest req) Submit an ad hoc query request.
RegistryObject	getRegistryObject (String id) Submit a request to get the RegistryObject that matches the specified id.
ExtrinsicObject	getRepositoryItem (String id) Submit a request to get the repository item that matches the specified id.

532 **How to model RepositoryItem in getRepositoryItem??**

533 **Missing getContent. Can we remove getContent from this interface? Seems redundant??**

534 **6.3 Concrete Registry Services**

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

537 ?? Implementations of the interfaces defined by the abstract registry service.

538 ?? Bindings of these concrete interfaces to specific communication protocols.

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

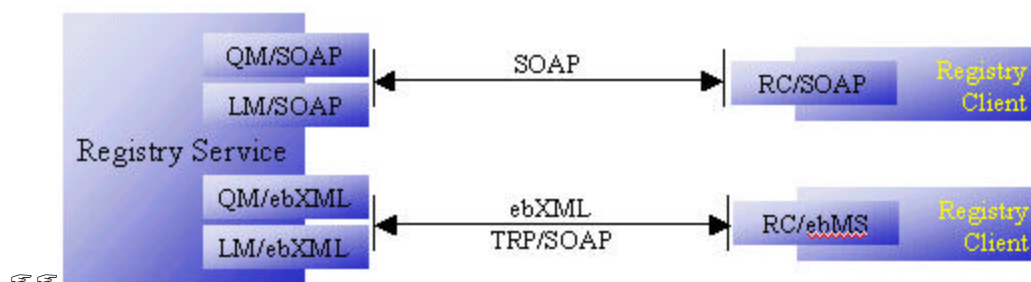
540 ?? A SOAP binding using the HTTP protocol

541 ?? An ebXML Messaging Service (ebMS) binding

542 ?? A REST binding

543 A registry must implement at least one of the SOAP and ebMS concrete bindings for the abstract
 544 registry service as shown in Figure 4. A registry must implement the REST binding for the
 545 abstract registry service as shown in Figure 4.

546



547

548

☞☞ **Figure 4: A Concrete ebXML Registry Service**

549 Figure 4 shows a concrete implementation of the abstract ebXML Registry (RegistryService) on
 550 the left side. The RegistryService provides the QueryManager and LifeCycleManager interfaces
 551 available with multiple protocol bindings (SOAP and ebMS).

552 Figure 4 also shows two different clients of the ebXML Registry on the right side. The top client
 553 uses SOAP interface to access the registry while the lower client uses ebMS interface. Clients
 554 use the appropriate concrete interface within the RegistryService service based upon their
 555 protocol preference.

556 6.4 SOAP Binding

557 6.4.1 WSDL Terminology Primer

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

565 An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract
 566 service may be used to define a concrete service by binding it to a specific protocol. This binding
 567 is done by providing a `binding definition` for each abstract port type that defines additional
 568 protocols specific details. Finally, a concrete `service definition` is defined as a collection of
 569 `ports`, where each port simply adds address information such as a URL for each concrete port.

570 6.4.2 Concrete Binding for SOAP

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

573 ?? A single service element with name “RegistryService” defines the concrete SOAP binding
 574 for the registry service.

575 ?? The service element includes two port definitions, where each port corresponds with one of
 576 the interfaces defined for the abstract registry service. Each port includes an HTTP URL for
 577 accessing that port.

578 ?? Each port definition also references a binding element, one for each interface defined in the
579 WSDL for the abstract registry service.

```
580 <service name = "RegistryService">
581   <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
582     <soap:address location = "http://your_URL_to_your_QueryManager"/>
583   </port>
584
585   <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
586     <soap:address location = "http://your_URL_to_your_QueryManager"/>
587   </port>
588 </service>
589
590
```

591 The complete WSDL description for the SOAP binding can be obtained via a hyperlink in
592 Appendix A.

593 6.5 ebXML Message Service Binding

594 6.5.1 Service and Action Elements

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

597 ?? The value of the Service element in the MessageHeader is an ebXML Registry Service
598 interface name (e.g., "LifeCycleManager"). The type attribute of the Service element should
599 have a value of "ebXMLRegistry".

600 ?? The value of the Action element in the MessageHeader is an ebXML Registry Service
601 method name (e.g., "submitObjects").

```
602
603 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
604 <eb:Action>submitObjects</eb:Action>
605
```

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

609 6.5.2 Synchronous and Asynchronous Responses

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

611 6.5.2.1 Asynchronous response

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

615 ?? MessageHeader;

616 ?? RegistryResponse element with empty content (e.g., **NO** AdHocQueryResponse);

617 o status attribute with value **Unavailable**.

618 The Registry delivers the actual Registry response element with non-empty content
619 asynchronously at a later time. The delivery is accomplished by the Registry invoking the
620 onResponse method on the RegistryClient interface as implemented by the registry client

621 application. The onResponse method includes a RegistryResponse element as shown below:

622 ?? MessageHeader;

623 ?? RegistryResponse element including;

624 ○ Status attribute (Success, Failure);

625 ○ Optional RegistryErrorList.

626 **6.5.2.2 Synchronous response**

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

629 ?? MessageHeader;

630 ?? RegistryResponse element including;

631 ○ Status attribute (Success, Failure);

632 ○ Optional RegistryErrorList.

633 **6.5.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

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

645 **6.6 REST Binding**

646 The ebXML Registry abstract interface defines a REST binding that enables access to the
647 registry over HTTP protocol.

648 REST [RESTThesis], which stands for Representational State Transfer, is an architectural style
649 for distributed hypermedia systems. The REST architectural style suggests that:

650 ○ A service be accessible over HTTP

651 ○ HTTP GET requests are preferred over other HTTP requests

652 ○ All access to the service capabilities and resources are via HTTP URLs

653 REST is more of a concept than a technology. It is easily implemented using standard facilities
654 found on a web server or development environment.

655 The REST binding maps the abstract registry interfaces to a REST styled HTTP interface. It
656 defines the URL parameters and their usage patterns that must be used to specify the interface,
657 method and invocation parameters in order to invoke a method on a registry interface such as the
658 QueryManager interface.

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

661 6.6.1 Standard URI Parameters

662 This section defines the normative URI parameters that must be supported by the REST
 663 Interface. A Registry may implement additional URI parameters in addition to these parameters.
 664

URL 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

665  **Table 4: Standard URI Parameters**


666

667 6.6.2 QueryManager REST Interface

668 The REST Interface to QueryManager must be supported by all registries.
 669 The REST Interface to QueryManager defines that the interface parameter must be
 670 "QueryManager". In addition the following method parameters are defined by the QueryManager
 671 REST Interface.

672

Method	Parameters	Return Value	HTTP Request Type
getRegistryObject	id	RegistryObject that matches the specified id.	GET
getRepositoryItem	id	The repository item that matches the specified id.	GET
submitAdhocQueryRequest	AdhocQueryRequest	RegistryResponse for the specified AdhocQueryRequest.	POST

673  **Table 5: QueryManager REST Interface**

674

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

677 6.6.2.1 Sample getRegistryObject Request

678

```
679 GET /rest?interface=QueryManager&method=getRegistryObject&param-id=
680 urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6 HTTP/1.0
```

681

682 **6.6.2.2 Sample getRegistryObject Response**

```
683
684 HTTP/1.1 200 OK
685 Content-Type: text/xml
686 Content-Length: 555
687
688 <?xml version="1.0"?>
689 <ExtrinsicObject id = "urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6 "
690   objectType="urn:uuid:32bbb291-0291-486d-a80d-cdd6cd625c57">
691   <Name>
692     <LocalizedString value = "Sample Object"/>
693   </Name>
694 </ExtrinsicObject>
695
```

696 **6.6.2.3 Sample getRepositoryItem Request**

```
697
698 GET /rest?interface=QueryManager&method=getRepositoryItem&param-id=
699 urn:uuid:a1137d00-091a-471e-8680-eb75b27b84b6 HTTP/1.0
700
```

701 **6.6.2.4 Sample getRepositoryItem Response**

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

```
704
705 HTTP/1.1 200 OK
706 Content-Type: text/xml
707 Content-Length: 555
708
709 <?xml version="1.0"?>
710 < CollaborationProtocolProfile>
711 ...
712 </CollaborationProtocolProfile>
713
```

714 **6.6.2.5 Sample submitAdhocQueryRequest Request**

715 The following example shows how an HTTP POST request is used to invoke the submit

```
716
717 POST /rest?interface=QueryManager&method=submitAdhocQueryRequest HTTP/1.0
718 User-Agent: Foo-ebXML/1.0
719 Host: www.registryserver.com
720 Content-Type: text/xml
721 Content-Length: 555
722
723 <?xml version="1.0"?>
724 <AdhocQueryRequest>
725 ...
726 </AdhocQueryRequest>
727
```

728 **6.6.2.6 Sample submitAdhocQueryRequest Response**

729

```

730 HTTP/1.1 200 OK
731 Content-Type: text/xml
732 Content-Length: 555
733
734 <?xml version="1.0"?>
735 <RegistryResponse />
736

```

737 6.6.3 LifecycleManager REST Interface

738 The REST Interface to QueryManager may optionally be supported by a registry.

739 The REST Interface to LifecycleManager defines that the interface parameter must be
 740 "LifecycleManager". In addition the following method parameters are defined by the
 741 LifecycleManager REST Interface.

742

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

743  Table 6: LifecycleManager REST Interface

744

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

747 6.6.3.1 Sample submitObjects Request

748 The following example shows how an HTTP POST request is used to invoke the submit

749

```

750 POST /rest?interface=QueryManager&method=submitObjects HTTP/1.0
751 User-Agent: Foo-ebXML/1.0
752 Host: www.registryserver.com
753 Content-Type: text/xml
754 Content-Length: 555
755
756 <?xml version="1.0"?>
757 <SubmitObjectsRequest>
758 ...
759 </SubmitObjectRequest>
760

```

761 6.6.3.2 Sample submitObjects Response

762

```

763 HTTP/1.1 200 OK

```

```
764 Content-Type: text/xml
765 Content-Length: 555
766
767 <?xml version="1.0"?>
768 <RegistryResponse>
769 ...
770 </RegistryResponse>
```

771 **How is a digital certificate provided to authenticate the HTTP request above??.**

772

773 **6.6.4 Security Considerations**

774 The REST interface supports the same mechanisms for data integrity and source integrity as are
775 mentioned in the Registry Services specification. Authentication may be performed by the
776 registry on a per message basis by verifying any digital signatures present, as well as at the
777 HTTP transport level using Basic or Digest authentication.

778 **6.6.5 Exception Handling**

779 Since the REST interface is merely an interface to various registry objects, exception handling
780 will take the same form as they do over other registry transports. Errors must be reported in a
781 `RegistryErrorList`, and sent back to the client on the same connection as the request.

782 When an error occurs, the HTTP status code and message should be appropriate to the error(s)
783 being reported in the `RegistryErrorList`. For example, if the `RegistryErrorList` is reporting
784 that an object wasn't found, therefore cannot be returned, an appropriate error code would be
785 404, with a message of "Object Not Found". A detailed list of HTTP status codes can be found in
786 [RFC2616].

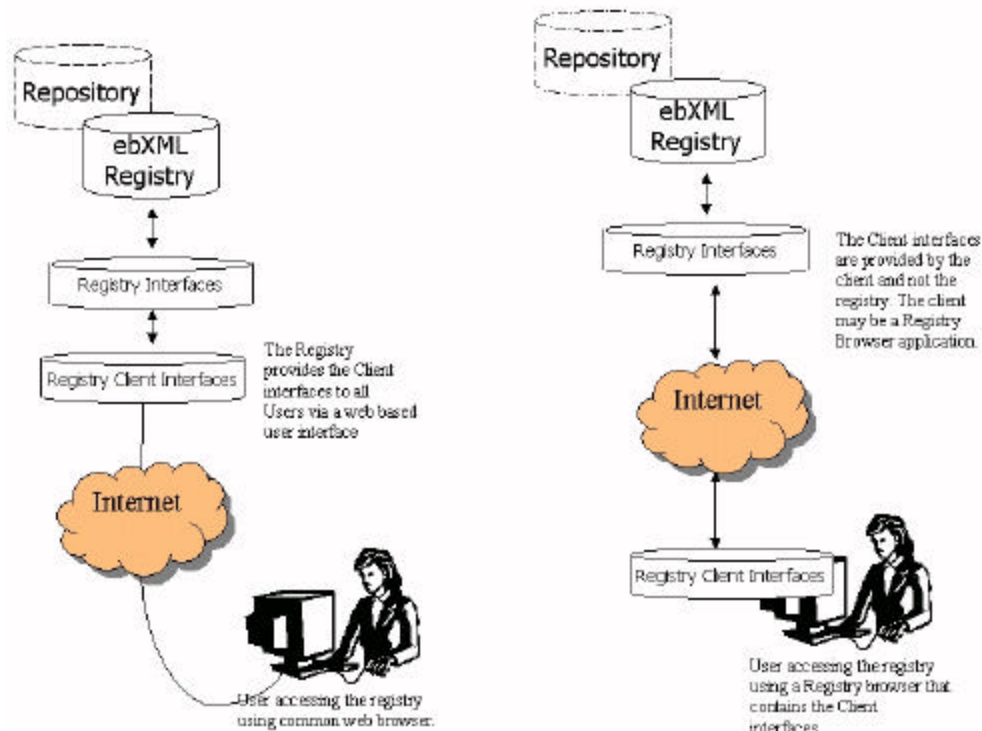
787

788 **6.7 Registry Clients**

789 **6.7.1 Registry Client Described**

790 The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts
791 the two possible topologies supported by the registry architecture with respect to the Registry
792 and Registry Clients. The picture on the left side shows the scenario where the Registry provides
793 a web based "thin client" application for accessing the Registry that is available to the user using
794 a common web browser. In this scenario the Registry Client interfaces reside across the Internet
795 and are local to the Registry from the user's view. The picture on the right side shows the
796 scenario where the user is using a "fat client" Registry Browser application to access the registry.
797 In this scenario the Registry Client interfaces reside within the Registry Browser tool and are
798 local to the Registry from the user's view. The Registry Client interfaces communicate with the
799 Registry over the Internet in this scenario.

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



805
806

☞ **Figure 5: Registry Architecture Supports Flexible Topologies**

807 **6.7.2 Registry Communication Bootstrapping**

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

814 ?? In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available
815 in a WSDL description for the registry. This WSDL conforms to the template WSDL
816 description in Appendix A.1. This WSDL description may be discovered in a public registry
817 such as UDDI.

818 ?? In case of ebMS binding, the information exchange between the client and the registry may
819 be accomplished in a registry specific manner, which may involve establishing a CPA
820 between the client and the registry. Once the information exchange has occurred the Registry
821 and the client will have addressing information (e.g. URLs) for the other party.

822 **6.7.2.1 Communication Bootstrapping for SOAP Binding**

823 Each ebXML Registry must provide a WSDL description for its RegistryService as defined by
824 Appendix A.1. A client uses the WSDL description to determine the address information of the
825 RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the
826 RegistryService may be accessed via a URL specified in the WSDL for the registry.

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

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

832 **6.7.2.2 Communication Bootstrapping for ebXML Message Service**

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

843 **6.7.3 RegistryClient Interface**

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

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

850  **Table 7: RegistryClient Summary**

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

851 **6.7.4 Registry Response**

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

854 **6.8 Interoperability Requirements**

855 **6.8.1 Client Interoperability**

856 The architecture requires that any ebXML compliant registry client can access any ebXML
 857 compliant registry service in an interoperable manner. An ebXML Registry may implement any
 858 number of protocol bindings from the set of normative bindings (currently ebMS and
 859 SOAP/HTTP) defined in this proposal. The support of additional protocol bindings is optional.

860 **6.8.2 Inter-Registry Cooperation**

861 This version of the specification does not preclude ebXML Registries from cooperating with
862 each other to share information, nor does it preclude owners of ebXML Registries from
863 registering their ebXML registries with other registry systems, catalogs, or directories.

864 Examples include:

865 ?? An ebXML Registry that serves as a registry of ebXML Registries.

866 ?? A non-ebXML Registry that serves as a registry of ebXML Registries.

867 ?? Cooperative ebXML Registries, where multiple ebXML registries register with each other in
868 order to form a federation.

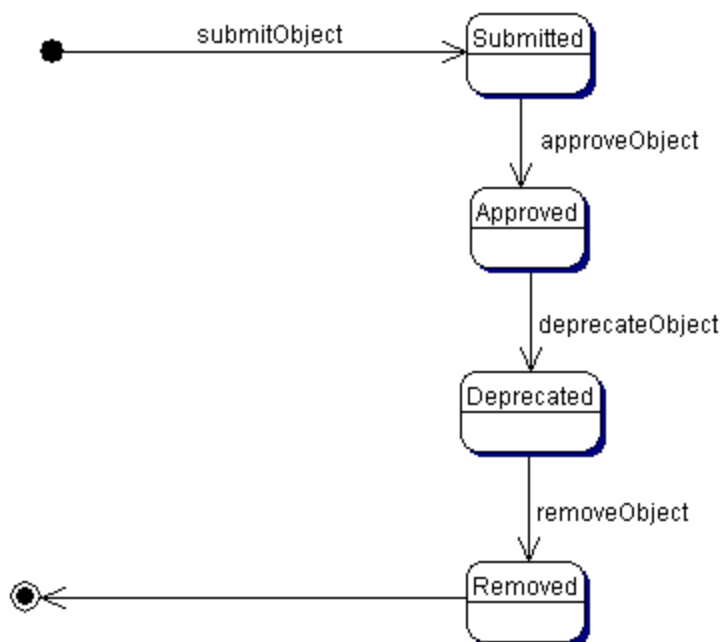
869 7 Life Cycle Management Service

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

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

879 7.1 Life Cycle of a Repository Item

880 The main purpose of the LifeCycleManagement service is to manage the life cycle of repository
 881 items. Figure 6 shows the typical life cycle of a repository item. Note that the current version of
 882 this specification does not support Object versioning. Object versioning will be added in a future
 883 version of this specification



884
 885

Figure 6: Life Cycle of a Repository Item

886 7.2 RegistryObject Attributes

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

892 **7.3 The Submit Objects Protocol**

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



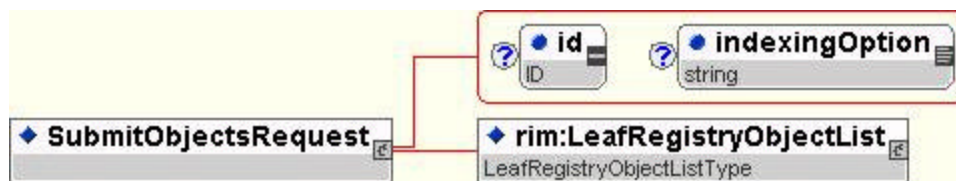
896
 897 **Figure 7: Submit Objects Sequence Diagram**

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

899 **7.3.1 SubmitObjectsRequest**

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

902 **7.3.1.1 Syntax:**



903
 904 **Figure 8: SubmitObjectsRequest Syntax**

905 **7.3.1.2 Parameters:**

906 *indexingOption*: This parameter specifies the submitter’s preference governing
 907 the indexing of the objects submitted via this request. Valid values are:

908 *NoIndexing*: This specifies that the registry must not index any of the objects
 909 submitted via this request.

910 *IndexModificationNotAllowed*: This specifies that the registry may index any of
 911 the objects submitted via this request as long as the original objects are not
 912 modified by the indexing operations.

913 *IndexModificationAllowed*: This specifies that the registry may index any of the
 914 objects submitted via this request even if the original objects are are modified by
 915 the indexing operations.

916 *LeafRegistryObjectsList*: This parameter specifies a collection of RegistryObject
 917 instances that are being submitted to the registry.

918

919 **7.3.1.3 Returns:**

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

921 **7.3.1.4 Exceptions:**

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

923 *AuthorizationException*: Indicates that the requestor attempted to perform an
 924 operation for which she was not authorized.

925 *ObjectNotFoundException*: Indicates that the requestor referenced an object
 926 within the request that was not found.

927 *InvalidRequestException*: Indicates that the requestor attempted to perform an
 928 operation which was semantically invalid.

929 *UnsupportedCapabilityException*: Indicates that the requestor attempted to
 930 submit some content that is not supported by the registry.

931 *QuotaExceededException*: Indicates that the requestor attempted to submit more
 932 content than the quota allowed for them by the registry.

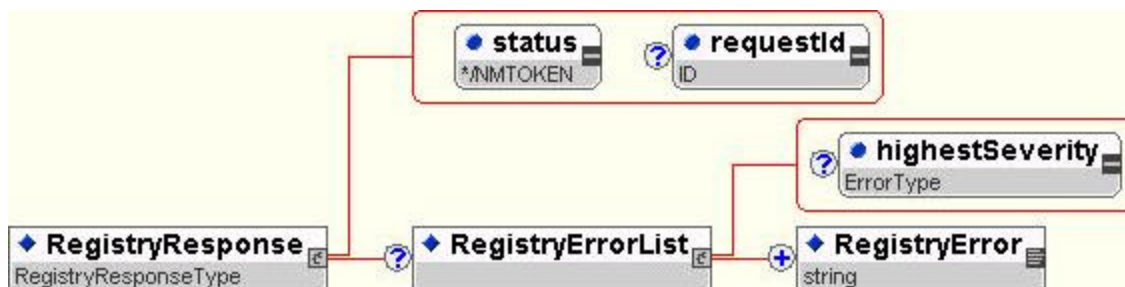
933

934 **7.3.2 RegistryResponse**

935 The RegistryResponse is sent by the registry as a response to several different requests. It is a
 936 simple response that can signal the status of a request and any errors or exceptions that may have
 937 occurred during the processing of that request.

938

939 **7.3.2.1 Syntax:**



940
 941

Figure 9: RegistryResponse Syntax

942 7.3.2.2 Parameters:

943 *requestId*: This parameter specifies the id of the request for which this is a
944 response of. It is used to correlate the response with its request.

945 *status*: This parameter specifies the status of the request. Valid values are as
946 follows:

947 *Success*: Request was processed successfully.

948 *Failure*: Errors were encountered during the processing of the request.

949 *Unavailable*: The results are unavailable. This is useful in asynchronous
950 responses.

951 *RegistryErrorList*: This parameter specifies a collection of RegistryErrors.
952 RegistryError is defined in ???. A RegistryErrorList includes a highestSeverity
953 attribute which logs the ErrorType for the most severe error that occurred.

954

955 7.3.3 Universally Unique ID Generation

956 As specified by [ebRIM], all objects in the registry have a unique id. The id must be a
957 Universally Unique Identifier (UUID) and must conform to the format of a URN that specifies a
958 DCE 128 bit UUID as specified in [UUID].

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

960 The registry usually generates this id. The client may optionally supply the id attribute for
961 submitted objects. If the client supplies the id and it conforms to the format of a URN that
962 specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the id
963 for the object. In this case, the registry must honour a client-supplied id and use it as the id
964 attribute of the object in the registry. If the id is found by the registry to not be globally unique,
965 the registry must raise the error condition: InvalidIdError.

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

969 7.3.4 ID Attribute And Object References

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

981 When an object in a SubmitObjectsRequest needs to reference an object that is already in the

982 registry, the request must contain an ObjectRef element whose id attribute is the id of the object
 983 in the registry. This id is by definition a proper uuid URN. An ObjectRef may be viewed as a
 984 proxy within the request for an object that is in the registry.

985 **7.3.5 Audit Trail**

986 The RS must create AuditableEvents object with eventType Created for each RegistryObject
 987 created via a SubmitObjects request.

988 **7.3.6 Error Handling**

989 **Need to move to a generic section on error handling??**

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

997 **7.3.7 Sample SubmitObjectsRequest**

998 The following example shows several different use cases in a single SubmitObjectsRequest. It
 999 does not show the complete SOAP or [ebMS] Message with the message header and additional
 1000 payloads in the message for the repository items.

1001 A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects
 1002 that are being submitted. It may also contain any number of ObjectRefs to link objects being
 1003 submitted to objects already within the registry.

```

1005 <?xml version = "1.0" encoding = "UTF-8"?>
1006 <SubmitObjectsRequest
1007   xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
1008   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
1009   xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
1010 spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
1011 file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
1012   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
1013   xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
1014   >
1015
1016   <rim:LeafRegistryObjectList>
1017
1018     <!--
1019     The following 3 objects package specified ExtrinsicObject in specified
1020     RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
1021     being submitted
1022     -->
1023
1024     <rim:RegistryPackage id = "acmePackage1" >
1025       <rim:Name>
1026         <rim:LocalizedString value = "RegistryPackage #1"/>
1027       </rim:Name>
1028       <rim:Description>
1029         <rim:LocalizedString value = "ACME's package #1"/>
1030       </rim:Description>
1031     </rim:RegistryPackage>
1032
1033     <rim:ExtrinsicObject id = "acmeCPP1" >
1034       <rim:Name>
  
```

```

1035     <rim:LocalizedString value = "Widget Profile" />
1036   </rim:Name>
1037   <rim:Description>
1038     <rim:LocalizedString value = "ACME's profile for selling widgets" />
1039   </rim:Description>
1040 </rim:ExtrinsicObject>
1041
1042   <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
1043 = "acmePackage1" targetObject = "acmeCPP1" />
1044
1045   <!--
1046     The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
1047     Where the RegistryPackage is being submitted and the ExtrinsicObject is
1048     already in registry
1049   -->
1050
1051   <rim:RegistryPackage id = "acmePackage2" >
1052     <rim:Name>
1053       <rim:LocalizedString value = "RegistryPackage #2"/>
1054     </rim:Name>
1055     <rim:Description>
1056       <rim:LocalizedString value = "ACME's package #2"/>
1057     </rim:Description>
1058   </rim:RegistryPackage>
1059
1060   <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
1061
1062   <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
1063 sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
1064
1065   <!--
1066     The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
1067     where the RegistryPackage and the ExtrinsicObject are already in registry
1068   -->
1069
1070   <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
1071   <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
1072
1073   <!-- id is unspecified implying that registry must create a uuid for this object -->
1074
1075   <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
1076 123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
1077
1078   <!--
1079     The following 3 objects externally link specified ExtrinsicObject using
1080     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
1081     are being submitted
1082   -->
1083
1084   <rim:ExternalLink id = "acmeLink1" >
1085     <rim:Name>
1086       <rim:LocalizedString value = "Link #1"/>
1087     </rim:Name>
1088     <rim:Description>
1089       <rim:LocalizedString value = "ACME's Link #1"/>
1090     </rim:Description>
1091   </rim:ExternalLink>
1092
1093   <rim:ExtrinsicObject id = "acmeCPP2" >
1094     <rim:Name>
1095       <rim:LocalizedString value = "Sprockets Profile" />
1096     </rim:Name>
1097     <rim:Description>
1098       <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
1099     </rim:Description>
1100   </rim:ExtrinsicObject>
1101
1102   <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
1103 sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
1104
1105   <!--
1106     The following 2 objects externally link specified ExtrinsicObject using specified
1107     ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject

```

```

1108     is already in registry. Note that the targetObject points to an ObjectRef in a
1109     previous line
1110     -->
1111
1112     <rim:ExternalLink id = "acmeLink2">
1113         <rim:Name>
1114             <rim:LocalizedString value = "Link #2"/>
1115         </rim:Name>
1116         <rim:Description>
1117             <rim:LocalizedString value = "ACME's Link #2"/>
1118         </rim:Description>
1119     </rim:ExternalLink>
1120
1121     <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
1122     "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
1123     123456789012"/>
1124
1125     <!--
1126     The following 3 objects externally identify specified ExtrinsicObject using specified
1127     ExternalIdentifier, where the ExternalIdentifier is being submitted and the
1128     ExtrinsicObject is already in registry. Note that the targetObject points to an
1129     ObjectRef in a previous line
1130     -->
1131
1132     <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
1133         <rim:Name>
1134             <rim:LocalizedString value = "DUNS"/>
1135         </rim:Name>
1136
1137         <rim:Description>
1138             <rim:LocalizedString value = "This is the DUNS scheme"/>
1139         </rim:Description>
1140     </rim:ClassificationScheme>
1141
1142     <rim:ExternalIdentifier id = "acmeDUNSID" identificationScheme="DUNS-id" value =
1143     "13456789012">
1144         <rim:Name>
1145             <rim:LocalizedString value = "DUNS" />
1146         </rim:Name>
1147         <rim:Description>
1148             <rim:LocalizedString value = "DUNS ID for ACME"/>
1149         </rim:Description>
1150     </rim:ExternalIdentifier>
1151
1152     <rim:Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc" associationType =
1153     "ExternallyIdentifies" sourceObject = "acmeDUNSID" targetObject = "urn:uuid:a2345678-1234-1234-
1154     123456789012"/>
1155
1156     <!--
1157     The following show submission of a brand new classification scheme in its entirety
1158     -->
1159     <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
1160         <rim:Name>
1161             <rim:LocalizedString value = "Geography"/>
1162         </rim:Name>
1163
1164         <rim:Description>
1165             <rim:LocalizedString value = "This is a sample Geography scheme"/>
1166         </rim:Description>
1167
1168         <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
1169     "NorthAmerica" >
1170             <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
1171     "UnitedStates" />
1172             <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
1173         </rim:ClassificationNode>
1174
1175         <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
1176             <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
1177                 <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
1178             </rim:ClassificationNode>
1179         </rim:ClassificationNode>
1180     </rim:ClassificationScheme>

```

```

1181
1182
1183 <!--
1184 The following show submission of a Automotive sub-tree of ClassificationNodes that
1185 gets added to an existing classification scheme named 'Industry'
1186 that is already in the registry
1187 -->
1188
1189 <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
1190 <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
1191 123456789012">
1192 <rim:Name>
1193 <rim:LocalizedString value = "Automotive" />
1194 </rim:Name>
1195 <rim:Description>
1196 <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
1197 </rim:Description>
1198 </rim:ClassificationNode>
1199
1200 <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
1201 <rim:Name>
1202 <rim:LocalizedString value = "Parts Supplier" />
1203 </rim:Name>
1204 <rim:Description>
1205 <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
1206 </rim:Description>
1207 </rim:ClassificationNode>
1208
1209 <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
1210 <rim:Name>
1211 <rim:LocalizedString value = "Engine Supplier" />
1212 </rim:Name>
1213 <rim:Description>
1214 <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
1215 </rim:Description>
1216 </rim:ClassificationNode>
1217
1218 <!--
1219 The following show submission of 2 Classifications of an object that is already in
1220 the registry using 2 ClassificationNodes. One ClassificationNode
1221 is being submitted in this request (Japan) while the other is already in the registry.
1222 -->
1223
1224 <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
1225 1234-123456789012" classificationNode = "Japan-id">
1226 <rim:Description>
1227 <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
1228 </rim:Description>
1229 </rim:Classification>
1230
1231 <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
1232 "urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
1233 123456789012">
1234 <rim:Description>
1235 <rim:LocalizedString value = "Classifies object using a node in the registry" />
1236 </rim:Description>
1237 </rim:Classification>
1238
1239 <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
1240 </rim:LeafRegistryObjectList>
1241 </SubmitObjectsRequest>
1242

```

1243 7.4 The Update Objects Protocol

1244 **There are issue with UpdateObjectsProtocol. We should consider getting rid of it??**

1245 This section describes the protocol of the Registry Service that allows a Registry Client to update
1246 one or more existing Registry Items in the registry on behalf of a Submitting Organization. It is
1247 expressed in UML notation as described in Appendix C.



1248
1249

☞☞ **Figure 10: Update Objects Sequence Diagram**

1250 For details on the schema for the Business documents shown in this process refer to Appendix B.
1251 The UpdateObjectsRequest message includes a LeafRegistryObjectList element. The
1252 LeafRegistryObjectList element specifies one or more RegistryObjects. Each object in the list
1253 must be a current RegistryObject. RegistryObjects must include all attributes, even those the
1254 user does not intend to change. A missing attribute is interpreted as a request to set that attribute
1255 to NULL.

1256 **7.4.1 Audit Trail**

1257 The RS must create AuditableEvents object with eventType Updated for each RegistryObject
1258 updated via an UpdateObjects request.

1259 **7.5 The Add Slots Protocol**

1260 This section describes the protocol of the Registry Service that allows a client to add slots to a
1261 previously submitted registry entry using the LifeCycleManager. Slots provide a dynamic
1262 mechanism for extending registry entries as defined by [ebRIM].



1263
1264

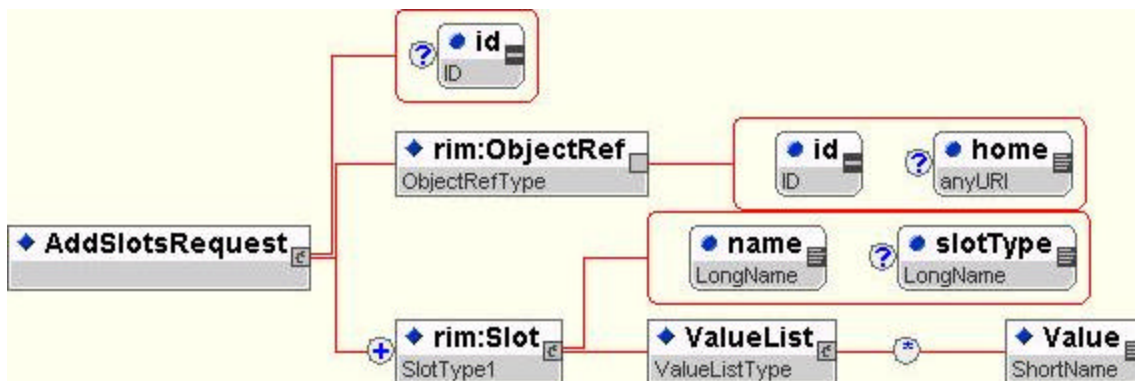
Figure 11: Add Slots Sequence Diagram

1265 In the event of success, the registry sends a RegistryResponse with a status of “success” back to
1266 the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure”
1267 back to the client.

1268 **7.5.1 AddSlotsRequest**

1269 The AddSlotsRequest is used by a client to add slots to an existing RegistryObject in the registry.

1270 **7.5.1.1 Syntax:**



1271
1272

Figure 12: AddSlotsRequest Syntax

1273 **7.5.1.2 Parameters:**

1274 *ObjectRef:* This parameter specifies a reference to a RegistryObject instance to
1275 which the requestor wishes to add slots via this request.

1276 *Slot:* This parameter specifies one or more Slot objects. Each Slot contains a
1277 ValueList with one or more Values. Each Slot also has a slot name and a slotType
1278 as described [ebRIM].

1279

1280 **7.5.1.3 Returns:**

1281 This request returns a RegistryResponse. See section 7.3.2 for details.

1282 **7.5.1.4 Exceptions:**

1283 In addition to the exceptions common to all requests, the following exceptions may be returned:

1284 *AuthorizationException*: Indicates that the requestor attempted to perform an
 1285 operation for which she was not authorized.

1286 *ObjectNotFoundException*: Indicates that the requestor referenced an object
 1287 within the request that was not found.

1288 *InvalidRequestException*: Indicates that the requestor attempted to perform an
 1289 operation which was semantically invalid.

1290

1291 **7.6 The Remove Slots Protocol**

1292 This section describes the protocol of the Registry Service that allows a client to remove slots to
 1293 a previously submitted registry entry using the LifeCycleManager.



1294
 1295

Figure 13: Remove Slots Sequence Diagram

1296 **7.6.1 RemoveSlotsRequest**

1297 The RemoveSlotsRequest is used by a client to remove slots from an existing RegistryObject in
 1298 the registry.

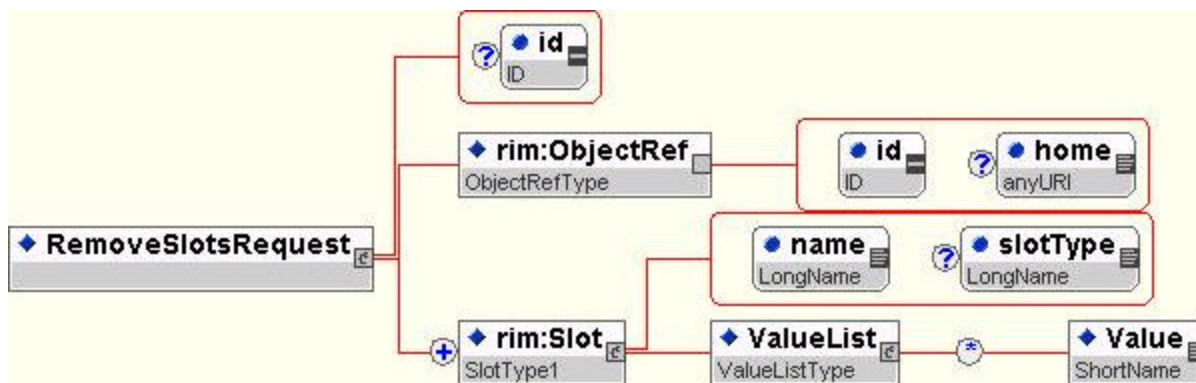
1299 **7.6.1.1 Syntax:**

Figure 14: RemoveSlotsRequest Syntax

1300
1301

1302 **7.6.1.2 Parameters:**

1303 *ObjectRef*: This parameter specifies a reference to a RegistryObject instance
1304 from which the requestor wishes to remove slots via this request.

1305 *Slot*: This parameter specifies one or more Slot objects. Each slot being removed
1306 is identified by its name attribute. Any Values specified with the ValueList for the
1307 Slot can be silently ignored.

1308

1309 **7.6.1.3 Returns:**

1310 This request returns a RegistryResponse. See section 7.3.2 for details.

1311 **7.6.1.4 Exceptions:**

1312 In addition to the exceptions common to all requests, the following exceptions may be returned:

1313 *AuthorizationException*: Indicates that the requestor attempted to perform an
1314 operation for which she was not authorized.

1315 *ObjectNotFoundException*: Indicates that the requestor referenced an object
1316 within the request that was not found.

1317 *InvalidRequestException*: Indicates that the requestor attempted to perform an
1318 operation which was semantically invalid.

1319

1320 **7.7 The Approve Objects Protocol**

1321 This section describes the protocol of the Registry Service that allows a client to approve one or
1322 more previously submitted repository items using the LifeCycleManager. Once a repository item
1323 is approved it will become available for use by business parties (e.g. during the assembly of new
1324 CPAs and Collaboration Protocol Profiles).



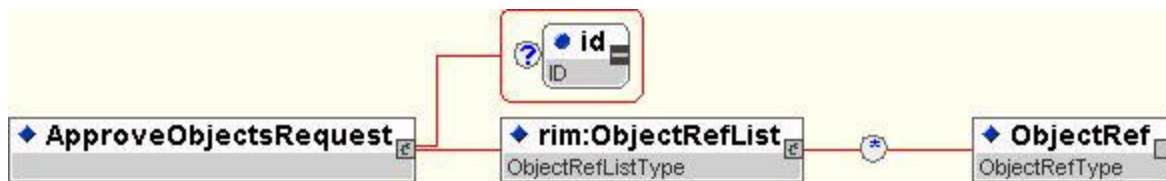
1325
1326

Figure 15: Approve Objects Sequence Diagram

1327 **7.7.1 ApproveObjectsRequest**

1328 The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject
1329 in the registry.

1330 **7.7.1.1 Syntax:**



1331
1332

Figure 16: ApproveObjectsRequest Syntax

1333 **7.7.1.2 Parameters:**

1334 *ObjectRefList*: This parameter specifies a collection of reference to existing
1335 RegistryObject instances in the registry. These are the objects that the requestor
1336 wishes to approve via this request.

1337

1338 **7.7.1.3 Returns:**

1339 This request returns a RegistryResponse. See section 7.3.2 for details.

1340 **7.7.1.4 Exceptions:**

1341 In addition to the exceptions common to all requests, the following exceptions may be returned:

1342 *AuthorizationException*: Indicates that the requestor attempted to perform an
1343 operation for which she was not authorized.

1344 *ObjectNotFoundException*: Indicates that the requestor referenced an object

1345 within the request that was not found.

1346 *InvalidRequestException*: Indicates that the requestor attempted to perform an
 1347 operation which was semantically invalid.

1348

1349 **7.7.2 Audit Trail**

1350 The RS must create AuditableEvents object with eventType Approved for each RegistryObject
 1351 approved via an Approve Objects request.

1352 **7.8 The Deprecate Objects Protocol**

1353 This section describes the protocol of the Registry Service that allows a client to deprecate one or
 1354 more previously submitted repository items using the LifeCycleManager. Once an object is
 1355 deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that
 1356 object can be submitted. However, existing references to a deprecated object continue to function
 1357 normally.



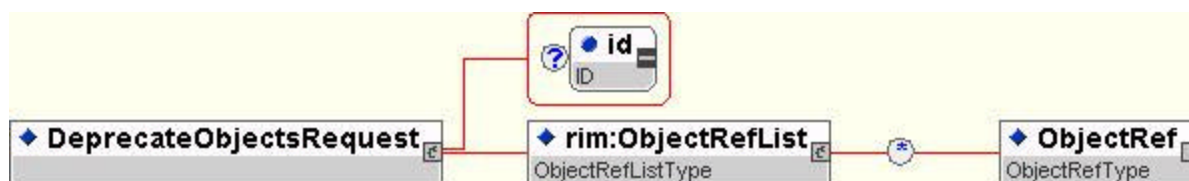
1358
 1359

Figure 17: Deprecate Objects Sequence Diagram

1360 **7.8.1 DeprecateObjectsRequest**

1361 The DeprecateObjectsRequest is used by a client to deprecate one or more existing
 1362 RegistryObject in the registry.

1363 **7.8.1.1 Syntax:**



1364
 1365

Figure 18: DeprecateObjectsRequest Syntax

1366 **7.8.1.2 Parameters:**

1367 ✎ *ObjectRefList*: This parameter specifies a collection of reference to existing
1368 RegistryObject instances in the registry. These are the objects that the requestor
1369 wishes to deprecate via this request.

1370

1371 **7.8.1.3 Returns:**

1372 This request returns a RegistryResponse. See section 7.3.2 for details.

1373 **7.8.1.4 Exceptions:**

1374 In addition to the exceptions common to all requests, the following exceptions may be returned:

1375 ✎ *AuthorizationException*: Indicates that the requestor attempted to perform an
1376 operation for which she was not authorized.

1377 ✎ *ObjectNotFoundException*: Indicates that the requestor referenced an object
1378 within the request that was not found.

1379 ✎ *InvalidRequestException*: Indicates that the requestor attempted to perform an
1380 operation which was semantically invalid.

1381

1382 **7.8.2 Audit Trail**

1383 The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject
1384 deprecated via a Deprecate Objects request.

1385 Global issue: We need check registryEntry Vs. Repository item term mis-use all over the RS
1386 specilay lifecycle chapter. Check error messages as well??.

1387 **7.9 The Remove Objects Protocol**

1388 This section describes the protocol of the Registry Service that allows a client to remove one or
1389 more RegistryObject instances and/or repository items using the LifeCycleManager.

1390 The RemoveObjectsRequest message is sent by a client to remove RegistryObject instances
1391 and/or repository items.

1392 The remove object protocol is expressed in UML notation as described in Appendix C.



1393
1394

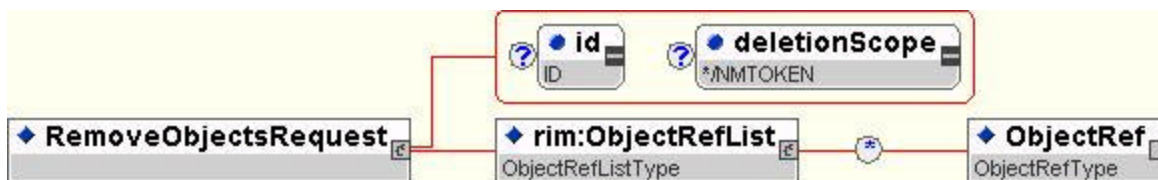
Figure 19: Remove Objects Sequence Diagram

1395 For details on the schema for the business documents shown in this process refer to Appendix B.

1396 **7.9.1 RemoveObjectsRequest**

1397 The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject
1398 from the registry.

1399 **7.9.1.1 Syntax:**



1400
1401

Figure 20: RemovalObjectsRequest Syntax

1402 **7.9.1.2 Parameters:**

1403 *deletionScope*: This parameter indicates the scope of impact of the
1404 RemoveObjectsRequest. Its valid values may be as follows:

1405 *DeleteRepositoryItemOnly*: This deletionScope specifies that the request should
1406 delete the repository items for the specified registry entries but not delete the
1407 specified registry entries. This is useful in keeping references to the registry
1408 entries valid.

1409 *DeleteAll*: This deletionScope specifies that the request should delete both the
1410 RegistryObject and the repository item for the specified registry entries. Only if
1411 all references (e.g. Associations, Classifications, ExternalLinks) to a
1412 RegistryObject have been removed, can that RegistryObject then be removed
1413 using a RemoveObjectsRequest with deletionScope DeleteAll.

1414 *ObjectRefList*: This parameter specifies a collection of reference to existing

1415 RegistryObject instances in the registry. These are the objects that the requestor
1416 wishes to remove via this request.

1417

1418 **7.9.1.3 Returns:**

1419 This request returns a RegistryResponse. See section 7.3.2 for details.

1420 **7.9.1.4 Exceptions:**

1421 In addition to the exceptions common to all requests, the following exceptions may be returned:

1422 *AuthorizationException*: Indicates that the requestor attempted to perform an
1423 operation for which she was not authorized.

1424 *ObjectNotFoundException*: Indicates that the requestor referenced an object
1425 within the request that was not found.

1426 *InvalidRequestException*: Indicates that the requestor attempted to perform an
1427 operation which was semantically invalid. Thrown when requestor attempts to
1428 remove a RegistryObject while it still has references.

1429

1430 **8 Query Management Service**

1431 This section describes the capabilities of the Registry Service that allow a client
1432 (QueryManagerClient) to search for or query different kind of registry objects in the ebXML
1433 Registry using the QueryManager interface of the Registry. The Registry supports the following
1434 query capabilities:

1435 ?? Filter Query

1436 ?? SQL Query

1437 The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry
1438 implementation. The SQL Query mechanism is an optional feature and MAY be provided by a
1439 registry implementation. However, if a vendor provides an SQL query capability to an ebXML
1440 Registry it SHALL conform to this document. As such this capability is a normative yet optional
1441 capability.

1442 In a future version of this specification, the W3C XQuery syntax may be considered as another
1443 query syntax.

1444 The Registry will hold a self-describing capability profile that identifies all supported
1445 AdhocQuery options. This profile is described in Appendix G.

1446 **8.1 Ad Hoc Query Request/Response**

1447 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
1448 AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry
1449 query mechanisms.

1450 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously
1451 back to the client. The AdhocQueryResponse returns a collection of objects whose element type
1452 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects
1453 representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate
1454 classes in [ebRIM] such as RegistryObject and RegistryEntry.

1455 Any errors in the query request messages are indicated in the corresponding query response
1456 message.



1457

1458

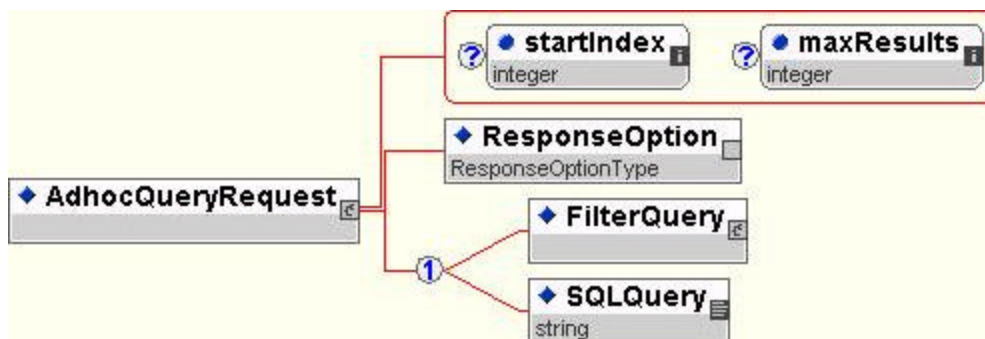
Figure 21: Submit Ad Hoc Query Sequence Diagram

1459 For details on the schema for the business documents shown in this process refer to Appendix
 1460 B.2.

1461 **8.1.1 AdhocQueryRequest**

1462 The AdhocQueryRequest is used to submit queries to the registry.

1463 **8.1.1.1 Syntax:**



1464
 1465

Figure 22: AdhocQueryRequest Syntax

1466 **8.1.1.2 Parameters:**

- 1467 *FilterQuery*: This parameter specifies a registry Filter Query.
- 1468 *maxResults*: This optional parameter specifies a limit on the maximum number of
- 1469 results (that are instances of the specified return type), the client wishes the query
- 1470 to return. If unspecified, the registry should return either all the results, or in case
- 1471 the result set size exceeds an registry operator specific limit, the registry should
- 1472 return a sub-set of results that are within the bounds of the registry operator
- 1473 specific limit.
- 1474 *ResponseOption*: This required parameter allows the client to control the format

1475 and content of the AdhocQueryResponse to this request. See section 8.1.3 for
 1476 details.

1477 *SQLQuery*: This parameter specifies a registry SQL Query.

1478 *startIndex*: This optional integer value is used to indicate which result set
 1479 SHOULD be returned first results set when iterating over a large result set. The
 1480 default value is 0, which returns result sets starting with index 0 (first result set).

1481

1482

1483 **8.1.1.3 Returns:**

1484 This request returns an AdhocQueryResponse. See section 9.6.2 for details.

1485 **8.1.1.4 Exceptions:**

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

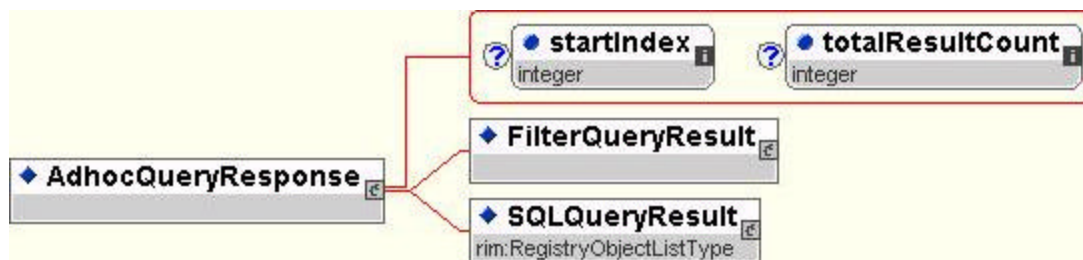
1487 *InvalidQueryException*: signifies that the query syntax was invalid. Client must
 1488 fix the query syntax and re-submit the query.

1489 **8.1.2 AdhocQueryResponse**

1490 The AdhocQueryRequest sent by the registry as a response to AdhocQueryRequest.

1491

1492 **8.1.2.1 Syntax:**



1493

1494

Figure 23: AdhocQueryResponse Syntax

1495 **8.1.2.2 Parameters:**

1496 *FilterQueryResult*: This parameter specifies the result of a registry Filter Query.

1497 *SQLQueryResult*: This parameter specifies the result of a registry SQL Query.

1498 *startIndex*: This optional integer value is used to indicate the index for the first
 1499 result in the result set returned by the query, within the complete result set
 1500 matching the query within the registry. By default, this value is 0.

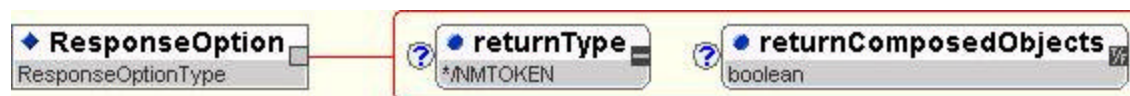
1501 *totalResultCount*: This optional parameter specifies the size of the complete
 1502 result set matching the query within the registry. When this value is unspecified,
 1503 the client should assume that value is the size of the result set contained within the
 1504 result.

1505

1506 **8.1.3 ResponseOption**

1507 A client specifies an ResponseOption structure within an AdhocQueryRequest to indicate the
1508 format of the results within the corresponding AdhocQueryResponse.

1509

1510 **8.1.3.1 Syntax:**

1511

1512

Figure 24: ResponseOption Syntax

1513

1514 **8.1.3.2 Parameters:**

1515 *returnComposedObjects*: This optional parameter specifies whether the
1516 RegistryObjects returned should include composed objects such as
1517 Classifications, ExternalIdentifiers etc. The default is to return all composed
1518 objects.

1519 *returnType*: This optional enumeration parameter specifies the type of
1520 RegistryObject to return within the response. Enumeration values for returnType
1521 are explained in section 8.1.3.3.

1522

1523 **8.1.3.3 Enumeration returnType**

1524 Enumeration values for returnType are as follows:

1525 ?? ObjectRef - This option specifies that the AdhocQueryResponse may contain a collection of
1526 ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return
1527 just the identifiers of the registry objects.

1528 ?? RegistryObject - This option specifies that the AdhocQueryResponse may contain a
1529 collection of RegistryObject XML elements as defined in [ebRIM Schema]. In this case all
1530 attributes of the registry objects are returned (objectType, name, description, ...) in addition
1531 to id attribute.

1532 ?? RegistryEntry - This option specifies that the AdhocQueryResponse may contain a collection
1533 of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which
1534 correspond to RegistryEntry or RegistryObject attributes.

1535 ?? LeafClass - This option specifies that the AdhocQueryResponse may contain a collection of
1536 XML elements that correspond to leaf classes as defined in [ebRIM Schema].

1537 ?? LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse may
1538 contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema]
1539 accompanied with their repository items or RegistryEntry or RegistryObject and their
1540 attributes. Linking of ExtrinsicObject and its repository item is accomplished using the
1541 technique explained in Section 8.4 -Content Retrieval.

1542 If “returnType” is higher then the RegistryObject option, then the highest option that satisfies the
 1543 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return
 1544 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass
 1545 option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then
 1546 RegistryObject metadata will be returned.

1547 **8.1.4 Iterative Query Support**

1548 The AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large
 1549 result set matching a logical query by allowing multiple AdhocQueryRequest requests to be
 1550 submitted such that each query requests a different sliding window within the result set. This
 1551 feature enables the registry to handle queries that match a very large result set, in a scalable
 1552 manner.

1553 The iterative queries feature is not a true Cursor capability as found in databases. The registry is
 1554 not required to maintain transactional consistency or state between iterations of a query. Thus it
 1555 is possible for new objects to be added or existing objects to be removed from the complete
 1556 result set in between iterations. As a consequence it is possible to have a result set element be
 1557 skipped or duplicated between iterations.

1558 Note that while it is not required, it may be possible for implementations to be smart and
 1559 implement a transactionally consistent iterative query feature. It is likely that a future version of
 1560 this specification will require a transactionally consistent iterative query capability.

1561 **8.1.4.1 Query Iteration Example**

1562 Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a
 1563 query that matches all 1007 Organizations. The user wishes to do the query iteratively such that
 1564 Organizations are retrieved in chunks of 100. The following table illustrates the parameters of
 1565 the AdhocQueryRequest and those of the AdhocQueryResponses for each iterative query in this
 1566 example.

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

1568

1569 8.2 Filter Query Support

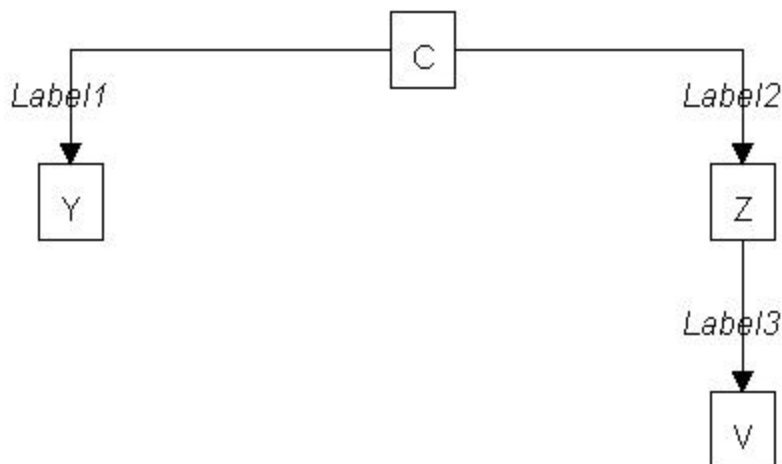
1570 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
 1571 conforming Registry implementation. Each query alternative is directed against a single class
 1572 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter
 1573 queries depending on which classes are queried on.

1574 ?? Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic
 1575 queries that might return different subclasses of the class that is queried on. The result of
 1576 such a query is a set of XML elements that correspond to instances of any class that satisfies
 1577 the responseOption defined previously in Section **Error! Reference source not found.** An
 1578 example might be that RegistryObjectQuery with responseOption LeafClass will return all
 1579 attributes of all instances that satisfy the query. This implies that response might return XML
 1580 elements that correspond to classes like ClassificationScheme, RegistryPackage,
 1581 Organization and Service.

1582 ?? Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact
 1583 traversals of these classes. Responses to these queries are accordingly constrained.

1584 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an
 1585 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified
 1586 herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified
 1587 in Section 8.1.

1588 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of
 1589 classes derived from a single class and its associations with other classes as defined by ebRIM.
 1590 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For
 1591 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a
 1592 class that is associated with Z. The ebRIM Binding for C might be as in Figure 25



1593  Figure 25: Example ebRIM Binding

1595 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and
 1596 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to
 1597 which ebRIM association is intended. The name of the query is determined by the root class, i.e.
 1598 this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances
 1599 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are
 1600 limited to instances that are linked to their parent node by the identified association.

1601 Each FilterQuery alternative depends upon one or more class filters, where a class filter is a
1602 restricted predicate clause over the attributes of a single class. Class methods that are defined in
1603 ebRIM and that return simple types constitute “visible attributes” that are valid choices for
1604 predicate clauses. Names of those attributes will be same as name of the corresponding method
1605 just without the prefix ‘get’. For example, in case of “getLevelNumber” method the
1606 corresponding visible attribute is “levelNumber”. The supported class filters are specified in
1607 Section 8.2.13 and the supported predicate clauses are defined in Section 8.2.14. A FilterQuery
1608 will be composed of elements that traverse the tree to determine which branches satisfy the
1609 designated class filters, and the query result will be the set of instances that support such a
1610 branch.

1611 In the above example, the CQuery element will have three subelements, one a CFilter on the C
1612 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on
1613 the Y class to eliminate branches from C to Y where the target of the association does not satisfy
1614 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third
1615 element is called a branch element because it allows class filters on each class along the path
1616 from C to V. In general, a branch element will have subelements that are themselves class filters,
1617 other branch elements, or a full-blown query on the class in the path.

1618 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one
1619 branch, filter or query element on Y is allowed. However, if the association is one-to-many, then
1620 multiple branch, filter or query elements are allowed. This allows one to specify that an instance
1621 of C must have associations with multiple instances of Y before the instance of C is said to
1622 satisfy the branch element.

1623 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be
1624 stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then
1625 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax
1626 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their
1627 respective superclass queries. Structures of XML elements that match the ebRIM classes are
1628 explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in
1629 ebRIM whenever possible.

1630 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual
1631 hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify
1632 the effect of that binding on query semantics.

1633 8.2.1 FilterQuery

1634 Purpose

1635 To identify a set of queries that traverse specific registry class. Each alternative assumes a
1636 specific binding to ebRIM. The status is a success indication or a collection of warnings and/or
1637 exceptions.

1638 Definition

```
1639 <element name="FilterQuery">  
1640   <complexType>  
1641     <choice minOccurs="1" maxOccurs="1">  
1642       <element ref="tns:RegistryObjectQuery" />  
1643       <element ref="tns:RegistryEntryQuery" />  
1644     </choice>  
1645   </complexType>  
1646 </element>
```



```

1645     <element ref="tns:AssociationQuery" />
1646     <element ref="tns:AuditableEventQuery" />
1647     <element ref="tns:ClassificationQuery" />
1648     <element ref="tns:ClassificationNodeQuery" />
1649     <element ref="tns:ClassificationSchemeQuery" />
1650     <element ref="tns:RegistryPackageQuery" />
1651     <element ref="tns:ExtrinsicObjectQuery" />
1652     <element ref="tns:OrganizationQuery" />
1653     <element ref="tns:ServiceQuery" />
1654   </choice>
1655 </complexType>
1656 </element>
1657
1658 <element name="FilterQueryResult">
1659   <complexType>
1660     <choice minOccurs="1" maxOccurs="1">
1661       <element ref="tns:RegistryObjectQueryResult" />
1662       <element ref="tns:RegistryEntryQueryResult" />
1663       <element ref="tns:AssociationQueryResult" />
1664       <element ref="tns:AuditableEventQueryResult" />
1665       <element ref="tns:ClassificationQueryResult" />
1666       <element ref="tns:ClassificationNodeQueryResult" />
1667       <element ref="tns:ClassificationSchemeQueryResult" />
1668       <element ref="tns:RegistryPackageQueryResult" />
1669       <element ref="tns:ExtrinsicObjectQueryResult" />
1670       <element ref="tns:OrganizationQueryResult" />
1671       <element ref="tns:ServiceQueryResult" />
1672     </choice>
1673   </complexType>
1674 </element>
1675

```

1676 Semantic Rules

- 1677 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 1678 2. Semantic rules specify the procedure for implementing the evaluation of Filter Queries.
1679 Implementations do not necessarily have to follow the same procedure provided that the
1680 same effect is achieved.
- 1681 3. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
1682 Each XML attribute carries a value derived from the value of an attribute specified in the
1683 Registry Information Model [eBRIM Schema].
- 1684 4. For each FilterQuery subelement there is only one corresponding FilterQueryResult
1685 subelement that must be returned as a response. Class name of the FilterQueryResult
1686 subelement has to match the class name of the FilterQuery subelement.
- 1687 5. If a Branch or Query element for a class has no sub-elements then every persistent instance
1688 of that class satisfies the Branch or Query.
- 1689 6. If an error condition is raised during any part of the execution of a FilterQuery, then the
1690 status attribute of the XML RegistryResult is set to "failure" and no AdHocQueryResult
1691 element is returned; instead, a RegistryErrorList element must be returned with its
1692 highestSeverity element set to "error". At least one of the RegistryError elements in the
1693 RegistryErrorList will have its severity attribute set to "error".

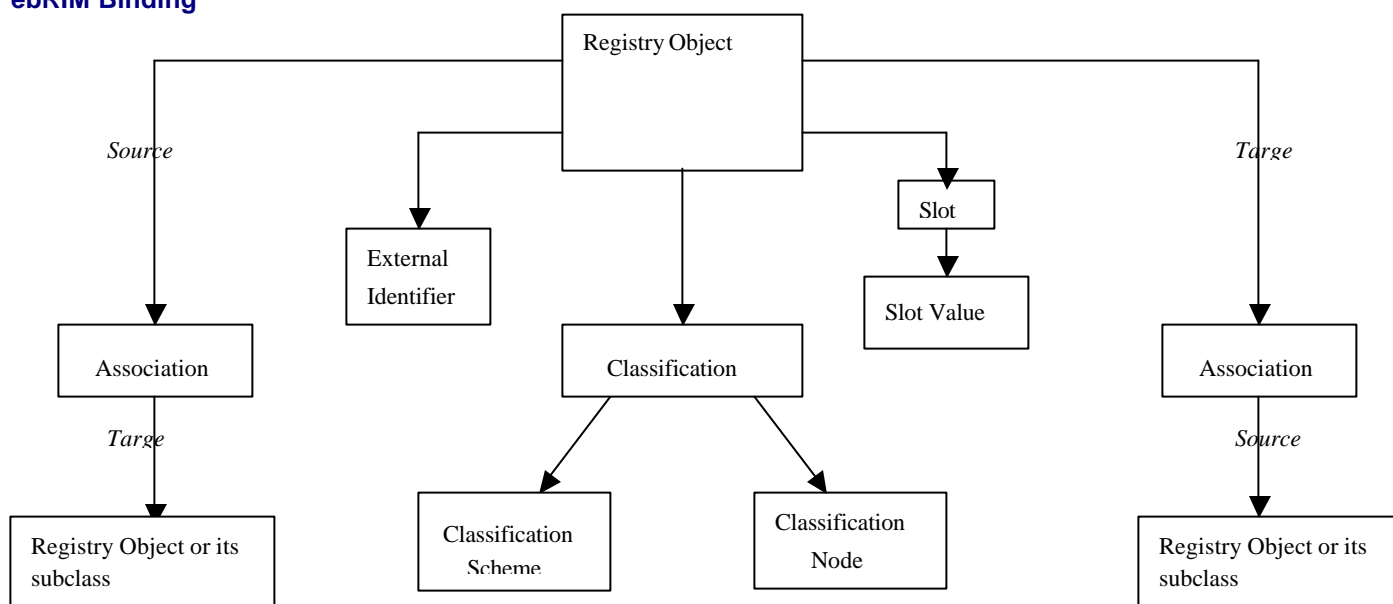
1694 7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of
 1695 the XML RegistryResult is set to “success” and an appropriate FilterQueryResult element
 1696 must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of
 1697 the RegistryErrorList is set to “warning” and the serverity attribute of each RegistryError is
 1698 set to “warning”.

1699 **8.2.2 RegistryObjectQuery**

1700 **Purpose**

1701 To identify a set of registry object instances as the result of a query over selected registry
 1702 metadata.

1703 **ebRIM Binding**



1704 **Figure 26: ebRIM Binding for RegistryObjectQuery**

1705 **Definition**

```

1706 <complexType name="RegistryObjectQueryType">
1707   <sequence>
1708     <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
1709     <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
1710     <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
1711     <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
1712     <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
1713     <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
1714     <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
1715     <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1716     <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1717   </sequence>
1718 </complexType>
1719 <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
1720
1721 <complexType name="LeafRegistryObjectListType">
1722   <choice minOccurs="0" maxOccurs="unbounded">
1723     <element ref="tns:ObjectRef" />
    
```

```

1724 <element ref="tns:Association" />
1725 <element ref="tns:AuditableEvent" />
1726 <element ref="tns:Classification" />
1727 <element ref="tns:ClassificationNode" />
1728 <element ref="tns:ClassificationScheme" />
1729 <element ref="tns:ExternalIdentifier" />
1730 <element ref="tns:ExternalLink" />
1731 <element ref="tns:ExtrinsicObject" />
1732 <element ref="tns:Organization" />
1733 <element ref="tns:RegistryPackage" />
1734 <element ref="tns:Service" />
1735 <element ref="tns:ServiceBinding" />
1736 <element ref="tns:SpecificationLink" />
1737 <element ref="tns:User" />
1738 </choice>
1739 </complexType>
1740
1741 <complexType name="RegistryObjectListType">
1742 <complexContent>
1743 <extension base="tns:LeafRegistryObjectListType">
1744 <choice minOccurs="0" maxOccurs="unbounded">
1745 <element ref="tns:RegistryEntry" />
1746 <element ref="tns:RegistryObject" />
1747 </choice>
1748 </extension>
1749 </complexContent>
1750 </complexType>
1751 <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1752
1753 <complexType name="InternationalStringBranchType">
1754 <sequence>
1755 <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1756 </sequence>
1757 </complexType>
1758
1759 <complexType name="AssociationBranchType">
1760 <sequence>
1761 <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1762 <choice minOccurs="0" maxOccurs="1">
1763 <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1764 <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1765 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1766 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1767 <element ref="tns:AssociationQuery" minOccurs="0" maxOccurs="1" />
1768 <element ref="tns:ClassificationQuery" minOccurs="0" maxOccurs="1" />
1769 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1770 <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1771 <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1772 <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1773 <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1774 <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1775 <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1776 <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
1777 <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1778 <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1779 </choice>
1780 </sequence>
1781 </complexType>

```

```
1782 <element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
1783 <element name="TargetAssociationBranch" type="tns:AssociationBranchType" />
1784
1785 <element name="ClassifiedByBranch">
1786   <complexType>
1787     <sequence>
1788       <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
1789       <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1790       <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1791     </sequence>
1792   </complexType>
1793 </element>
1794
1795 <element name="SlotBranch">
1796   <complexType>
1797     <sequence>
1798       <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
1799       <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
1800     </sequence>
1801   </complexType>
1802 </element>
1803
1804 <element name="UserBranch">
1805   <complexType>
1806     <sequence>
1807       <element ref="tns:UserFilter" minOccurs="0" maxOccurs="1" />
1808       <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
1809       <element ref="tns:TelephoneFilter" minOccurs="0" maxOccurs="unbounded" />
1810       <element ref="tns:EmailAddressFilter" minOccurs="0" maxOccurs="unbounded" />
1811       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1812     </sequence>
1813   </complexType>
1814 </element>
1815
1816 <complexType name="ServiceBindingBranchType">
1817   <sequence>
1818     <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
1819     <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
1820     <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
1821   </sequence>
1822 </complexType>
1823 <element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType" />
1824 <element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType" />
1825
1826 <element name="SpecificationLinkBranch">
1827   <complexType>
1828     <sequence>
1829       <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
1830       <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1831       <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1832     </sequence>
1833   </complexType>
1834 </element>
1835
```

1836 **Semantic Rules**

- 1837 1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The
1838 following steps will eliminate instances in RO that do not satisfy the conditions of the
1839 specified filters.
- 1840 a) If RO is empty then go to number 2 below.
- 1841 b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a
1842 registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from
1843 RO. If RO is empty then continue to the next numbered rule.
- 1844 c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise,
1845 let x be a remaining registry object in RO. If x is not linked to at least one
1846 ExternalIdentifier instance, then remove x from RO; otherwise, treat each
1847 ExternalIdentifierFilter element separately as follows: Let EI be the set of
1848 ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If
1849 EI is empty, then remove x from RO. If RO is empty then continue to the next numbered
1850 rule.
- 1851 d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a
1852 remaining registry object in RO. If x doesn't have an auditable event that satisfy
1853 AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is
1854 empty then continue to the next numbered rule.
- 1855 e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining
1856 registry object in RO. If x does not have a name then remove x from RO. If RO is empty
1857 then continue to the next numbered rule; otherwise treat NameBranch as follows: If any
1858 LocalizedStringFilter that is specified is not satisfied by all of the LocalizedStrings that
1859 constitute the name of the registry object then remove x from RO. If RO is empty then
1860 continue to the next numbered rule.
- 1861 f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a
1862 remaining registry object in RO. If x does not have a description then remove x from RO.
1863 If RO is empty then continue to the next numbered rule; otherwise treat
1864 DescriptionBranch as follows: If any LocalizedStringFilter that is specified is not
1865 satisfied by all of the LocalizedStrings that constitute the description of the registry
1866 object then remove x from RO. If RO is empty then continue to the next numbered rule.

- 1867 g) If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x
1868 be a remaining registry object in RO. If x is not the classifiedObject of at least one
1869 Classification instance, then remove x from RO; otherwise, treat each
1870 ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified
1871 within the ClassifiedByBranch, then let CL be the set of all Classification instances that
1872 have x as the classifiedObject; otherwise, let CL be the set of Classification instances that
1873 satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then
1874 remove x from RO and continue to the next numbered rule. Otherwise, if CL is not
1875 empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of
1876 remaining Classification instances in CL whose defining classification scheme satisfies
1877 the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and
1878 continue to the next numbered rule. Otherwise, if CL remains not empty, and if a
1879 ClassificationNodeQuery is specified, then replace CL by the set of remaining
1880 Classification instances in CL for which a classification node exists and for which that
1881 classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then
1882 remove x from RO. If RO is empty then continue to the next numbered rule.
- 1883 h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a
1884 remaining registry object in RO. If x is not linked to at least one Slot instance, then
1885 remove x from RO. If RO is empty then continue to the next numbered rule; otherwise,
1886 treat each SlotBranch element separately as follows: If a SlotFilter is not specified within
1887 the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the
1888 set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,
1889 then remove x from RO and continue to the next numbered rule. Otherwise, if SL
1890 remains not empty, and if a SlotValueFilter is specified, replace SL by the set of
1891 remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is
1892 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
- 1893 i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,
1894 let x be a remaining registry object in RO. If x is not the source object of at least one
1895 Association instance, then remove x from RO. If RO is empty then continue to the next
1896 numbered rule; otherwise, treat each SourceAssociationBranch element separately as
1897 follows:
1898 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be
1899 the set of all Association instances that have x as a source object; otherwise, let AF be the
1900 set of Association instances that satisfy the AssociationFilter and have x as the source
1901 object. If AF is empty, then remove x from RO.
1902
1903 If RO is empty then continue to the next numbered rule.
1904
1905 If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT
1906 be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target
1907 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1908 then continue to the next numbered rule.
1909

- 1910 If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let
1911 ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1912 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1913 RO is empty then continue to the next numbered rule.
1914
- 1915 If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT
1916 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1917 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1918 empty then continue to the next numbered rule.
1919
- 1920 If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT
1921 be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the
1922 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1923 empty then continue to the next numbered rule.
1924
- 1925 If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let
1926 ROT be the set of ClassificationScheme instances that satisfy the
1927 ClassificationSchemeQuery and are the target object of some element of AF. If ROT is
1928 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1929
- 1930 If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let
1931 ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1932 and are the target object of some element of AF. If ROT is empty, then remove x from
1933 RO. If RO is empty then continue to the next numbered rule.
1934
- 1935 If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT
1936 be the set of Organization instances that satisfy the OrganizationQuery and are the target
1937 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1938 then continue to the next numbered rule.
1939
- 1940 If an AuditableEventQuery is specified within the SourceAssociationBranch, then let
1941 ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1942 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1943 is empty then continue to the next numbered rule.
1944
- 1945 If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let
1946 ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1947 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1948 RO is empty then continue to the next numbered rule.
1949
- 1950 If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let
1951 ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1952 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1953 is empty then continue to the next numbered rule.
1954

1955 If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the
1956 set of Service instances that satisfy the ServiceQuery and are the target object of some
1957 element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue
1958 to the next numbered rule.
1959

1960 If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set
1961 of User instances that are the target object of some element of AF. If ROT is empty, then
1962 remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the
1963 member of ROT. If a UserFilter element is specified within the UserBranch, and if u does
1964 not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from
1965 RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter
1966 element is specified within the UserBranch, and if the postal address of u does not satisfy
1967 that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is
1968 empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are
1969 specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied
1970 by all of the telephone numbers of u then remove u from ROT. If ROT is empty, then
1971 remove x from RO. If RO is empty then continue to the next numbered rule. If an
1972 OrganizationQuery element is specified within the UserBranch, then let o be the
1973 Organization instance that is identified by the organization that u is affiliated with. If o
1974 doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROT.
1975 If ROT is empty, then remove x from RO. If RO is empty then continue to the next
1976 numbered rule.
1977

1978 If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT
1979 be the set of Classification instances that satisfy the ClassificationQuery and are the
1980 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1981 empty then continue to the next numbered rule (Rule 2).
1982

1983 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1984 ROT be the set of ServiceBinding instances that are the target object of some element of
1985 AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next
1986 numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is
1987 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then
1988 remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then
1989 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the
1990 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as
1991 follows:
1992

1993 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link
1994 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1995 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1996 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then
1997 remove x from RO. If RO is empty then continue to the next numbered rule. If a
1998 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1999 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:
2000 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
2001 not a specification link for at least one registry object in RO, then remove sl from SL. If

2002 SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO
2003 is empty then continue to the next numbered rule. If a RegistryEntryQuery element is
2004 specified within the SpecificationLinkBranch then let sl be a remaining specification link
2005 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the
2006 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least
2007 one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from
2008 ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next
2009 numbered rule. If a ServiceBindingTargetBranch is specified within the
2010 ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy
2011 the ServiceBindingTargetBranch and are the target service binding of some element of
2012 ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from
2013 RO. If RO is empty then continue to the next numbered rule.

2014
2015 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
2016 ROT be the set of SpecificationLink instances that are the target object of some element
2017 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the
2018 next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is
2019 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
2020 remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then
2021 continue to the next numbered rule. If a RegistryObjectQuery element is specified within
2022 the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat
2023 RegistryObjectQuery element as follows: Let RO be the result set of the
2024 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
2025 registry object in RO, then remove sl from ROT. If ROT is empty then remove x from
2026 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery
2027 element is specified within the SpecificationLinkBranch then let sl be a remaining
2028 specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the
2029 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification
2030 link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then
2031 remove x from RO. If RO is empty then continue to the next numbered rule.

2032
2033 If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be
2034 the set of Association instances that satisfy the AssociationQuery and are the target object
2035 of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then
2036 continue to the next numbered rule (Rule 2).

2037
2038 j) If a TargetAssociationBranch element is not specified then go to the next step; otherwise,
2039 let x be a remaining registry object in RO. If x is not the target object of some
2040 Association instance, then remove x from RO. If RO is empty then continue to the next
2041 numbered rule; otherwise, treat each TargetAssociationBranch element separately as
2042 follows:

2043

2044 If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be
2045 the set of all Association instances that have x as a target object; otherwise, let AF be the
2046 set of Association instances that satisfy the AssociationFilter and have x as the target
2047 object. If AF is empty, then remove x from RO. If RO is empty then continue to the next
2048 numbered rule.
2049

2050 If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be
2051 the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source
2052 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
2053 then continue to the next numbered rule.
2054

2055 If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let
2056 ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
2057 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
2058 RO is empty then continue to the next numbered rule.
2059

2060 If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS
2061 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
2062 source object of some element of AF. If ROS is empty, then remove x from RO. If RO is
2063 empty then continue to the next numbered rule.
2064

2065 If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS
2066 be the set of
2067 RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of
2068 some element of AF. If ROS is empty, then remove x from RO. If RO is empty then
2069 continue to the next numbered rule.
2070

2071 If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let
2072 ROS be the set of ClassificationScheme instances that satisfy the
2073 ClassificationSchemeQuery and are the source object of some element of AF. If ROS is
2074 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
2075

2076 If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let
2077 ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
2078 and are the source object of some element of AF. If ROS is empty, then remove x from
2079 RO. If RO is empty then continue to the next numbered rule.
2080

2081 If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS
2082 be the set of Organization instances that satisfy the OrganizationQuery and are the source
2083 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
2084 then continue to the next numbered rule.
2085

2086 If an AuditableEventQuery is specified within the TargetAssociationBranch, then let
2087 ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
2088 the source object of some element of AF. If ROS is empty, then remove x from RO. If
2089 RO is empty then continue to the next numbered rule.

2090
2091 If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let
2092 ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
2093 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
2094 RO is empty then continue to the next numbered rule.

2095
2096 If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let
2097 ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
2098 the source object of some element of AF. If ROS is empty, then remove x from RO. If
2099 RO is empty then continue to the next numbered rule.

2100
2101 If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the
2102 set of Service instances that satisfy the ServiceQuery and are the source object of some
2103 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue
2104 to the next numbered rule.

2105
2106 If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set
2107 of User instances that are the source object of some element of AF. If ROS is empty, then
2108 remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the
2109 member of ROS. If a UserFilter element is specified within the UserBranch, and if u does
2110 not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from
2111 RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter
2112 element is specified within the UserBranch, and if the postal address of u does not satisfy
2113 that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is
2114 empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are
2115 specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied
2116 by all of the telephone numbers of u then remove u from ROS. If ROS is empty, then
2117 remove x from RO. If RO is empty then continue to the next numbered rule. If an
2118 OrganizationQuery element is specified within the UserBranch, then let o be the
2119 Organization instance that is identified by the organization that u is affiliated with. If o
2120 doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROS.
2121 If ROS is empty, then remove x from RO. If RO is empty then continue to the next
2122 numbered rule.

2123
2124 If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be
2125 the set of Classification instances that satisfy the ClassificationQuery and are the source
2126 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
2127 then continue to the next numbered rule (Rule 2).

2128

2129 If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS
2130 be the set of ServiceBinding instances that are the source object of some element of AF.
2131 If ROS is empty, then remove x from RO. If RO is empty then continue to the next
2132 numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is
2133 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then
2134 remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then
2135 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the
2136 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as
2137 follows:
2138 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link
2139 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
2140 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
2141 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then
2142 remove x from RO. If RO is empty then continue to the next numbered rule. If a
2143 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
2144 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:
2145 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
2146 not a specification link for some registry object in RO, then remove sl from SL. If SL is
2147 empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is
2148 empty then continue to the next numbered rule. If a RegistryEntryQuery element is
2149 specified within the SpecificationLinkBranch then let sl be a remaining specification link
2150 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the
2151 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
2152 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.
2153 If ROS is empty then remove x from RO. If RO is empty then continue to the next
2154 numbered rule.
2155

2156 If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let
 2157 ROS be the set of SpecificationLink instances that are the source object of some element
 2158 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the
 2159 next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is
 2160 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
 2161 remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then
 2162 continue to the next numbered rule. If a RegistryObjectQuery element is specified within
 2163 the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat
 2164 RegistryObjectQuery element as follows: Let RO be the result set of the
 2165 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
 2166 registry object in RO, then remove sl from ROS. If ROS is empty then remove x from
 2167 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery
 2168 element is specified within the SpecificationLinkBranch then let sl be a remaining
 2169 specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the
 2170 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification
 2171 link for some registry entry in RE, then remove sl from ROS. If ROS is empty then
 2172 remove x from RO. If RO is empty then continue to the next numbered rule. If a
 2173 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT
 2174 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and
 2175 are the target service binding of some element of ROT. If SBT is empty then remove sb
 2176 from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the
 2177 next numbered rule.

2178
 2179 If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be
 2180 the set of Association instances that satisfy the AssociationQuery and are the source
 2181 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
 2182 then continue to the next numbered rule (Rule 2).

- 2183 2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set
 2184 RO to be the result of the RegistryObjectQuery.
- 2185 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2186 within the RegistryResponse.

2187 Examples

2188 A client application needs all items that are classified by two different classification schemes,
 2189 one based on "Industry" and another based on "Geography". Both schemes have been defined by
 2190 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",
 2191 respectively. The following query identifies registry entries for all registered items that are
 2192 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of
 2193 "Asia/Japan".

```
2194
2195 <AdhocQueryRequest>
2196   <ResponseOption returnType = "RegistryEntry"/>
2197   <FilterQuery>
2198     <RegistryObjectQuery>
2199       <ClassifiedByBranch>
2200         <ClassificationFilter>
2201           <Clause>
2202             <SimpleClause leftArgument = "path">
```

```

2203     <StringClause stringPredicate = "Equal">//Automotive</StringClause>
2204     </SimpleClause>
2205     </Clause>
2206     </ClassificationFilter>
2207     <ClassificationSchemeQuery>
2208     <NameBranch>
2209     <LocalizedStringFilter>
2210     <Clause>
2211     <SimpleClause leftArgument = "value">
2212     <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
2213     </SimpleClause>
2214     </Clause>
2215     </LocalizedStringFilter>
2216     </NameBranch>
2217     </ClassificationSchemeQuery>
2218     </ClassifiedByBranch>
2219     <ClassifiedByBranch>
2220     <ClassificationFilter>
2221     <Clause>
2222     <SimpleClause leftArgument = "path">
2223     <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
2224     </SimpleClause>
2225     </Clause>
2226     </ClassificationFilter>
2227     <ClassificationSchemeQuery>
2228     <NameBranch>
2229     <LocalizedStringFilter>
2230     <Clause>
2231     <SimpleClause leftArgument = "value">
2232     <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
2233     </SimpleClause>
2234     </Clause>
2235     </LocalizedStringFilter>
2236     </NameBranch>
2237     </ClassificationSchemeQuery>
2238     </ClassifiedByBranch>
2239     </RegistryObjectQuery>
2240     </FilterQuery>
2241 </AdhocQueryRequest>
2242

```

2243 A client application wishes to identify all RegistryObject instances that are classified by some
2244 internal classification scheme and have some given keyword as part of the description of one of
2245 the classification nodes of that classification scheme. The following query identifies all such
2246 RegistryObject instances. The query takes advantage of the knowledge that the classification
2247 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode
2248 instances.

```

2249 <AdhocQueryRequest>
2250 <ResponseOption returnType = "RegistryObject"/>
2251 <FilterQuery>
2252 <RegistryObjectQuery>
2253 <ClassifiedByBranch>
2254 <ClassificationNodeQuery>
2255 <DescriptionBranch>
2256 <LocalizedStringFilter>
2257 <Clause>
2258

```

```

2259     <SimpleClause leftArgument = "value">
2260         <StringClause stringPredicate = "Equal">transistor</StringClause>
2261     </SimpleClause>
2262 </Clause>
2263 </LocalizedStringFilter>
2264 </DescriptionBranch>
2265 </ClassificationNodeQuery>
2266 </ClassifiedByBranch>
2267 </RegistryObjectQuery>
2268 </FilterQuery>
2269 </AdhocQueryRequest>
2270

```

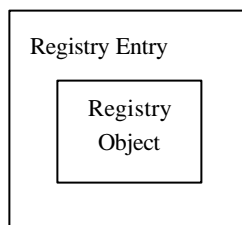
2271 8.2.3 RegistryEntryQuery

2272 Purpose

2273 To identify a set of registry entry instances as the result of a query over selected registry
 2274 metadata.

2275 ebRIM Binding

2276



2277  **Figure 27: ebRIM Binding for RegistryEntryQuery**

2278 Definition

```

2279 <complexType name="RegistryEntryQueryType">
2280     <complexContent>
2281         <extension base="tns:RegistryObjectQueryType">
2282             <sequence>
2283                 <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
2284             </sequence>
2285         </extension>
2286     </complexContent>
2287 </complexType>
2288 <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
2289
2290 <element name="RegistryEntryQueryResult">
2291     <complexType>
2292         <choice minOccurs="0" maxOccurs="unbounded">
2293             <element ref="rim:ObjectRef" />
2294             <element ref="rim:ClassificationScheme" />
2295             <element ref="rim:ExtrinsicObject" />
2296             <element ref="rim:RegistryEntry" />
2297             <element ref="rim:RegistryObject" />
2298             <element ref="rim:RegistryPackage" />
2299         </choice>
2300     </complexType>
2301 </element>
2302

```

2303

2304 **Semantic Rules**

- 2305 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
2306 steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
- 2307 a) If RE is empty then continue to the next numbered rule.
- 2308 b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
2309 registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
2310 If RE is empty then continue to the next numbered rule.
- 2311 c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
2312 RegistryObjectQuery over RE as explained in Section 8.2.2.
- 2313 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, set RE
2314 to be the result of the RegistryEntryQuery.
- 2315 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2316 within the RegistryResponse.

2317 **Examples**

2318 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if
2319 they have registered any of their business documents in the Registry. The following query
2320 returns a set of registry entry identifiers for currently registered items submitted by any
2321 organization whose name includes the string "XYZ". It does not return any registry entry
2322 identifiers for superseded, replaced, deprecated, or withdrawn items.

```

2323 <AdhocQueryRequest>
2324   <ResponseOption returnType = "ObjectRef"/>
2325   <FilterQuery>
2326     <RegistryEntryQuery>
2327       <TargetAssociationBranch>
2328         <AssociationFilter>
2329           <Clause>
2330             <SimpleClause leftArgument = "associationType">
2331               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2332             </SimpleClause>
2333           </Clause>
2334         </AssociationFilter>
2335       <OrganizationQuery>
2336         <NameBranch>
2337           <LocalizedStringFilter>
2338             <Clause>
2339               <SimpleClause leftArgument = "value">
2340                 <StringClause stringPredicate = "Contains">XYZ</StringClause>
2341               </SimpleClause>
2342             </Clause>
2343           </LocalizedStringFilter>
2344         </NameBranch>
2345       </OrganizationQuery>
2346     </TargetAssociationBranch>
2347   </RegistryEntryFilter>
2348   <Clause>
2349     <SimpleClause leftArgument = "status">
2350

```



```

2351     <StringClause stringPredicate = "Equal">Approved</StringClause>
2352     </SimpleClause>
2353   </Clause>
2354 </RegistryEntryFilter>
2355 </RegistryEntryQuery>
2356 </FilterQuery>
2357 </AdhocQueryRequest>
2358

```

2359 A client is using the United Nations Standard Product and Services Classification (UNSPSC)
 2360 scheme and wants to identify all companies that deal with products classified as "Integrated
 2361 circuit components", i.e. UNSPSC code "321118". The client knows that companies have
 2362 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
 2363 such profile has been classified by UNSPSC according to the products the company deals with.
 2364 However, the client does not know if the UNSPSC classification scheme is internal or external to
 2365 this registry. The following query returns a set of approved registry entry instances for CPP's of
 2366 companies that deal with integrated circuit components.

```

2367 <AdhocQueryRequest>
2368 <ResponseOption returnType = "RegistryEntry"/>
2369 <FilterQuery>
2370 <RegistryEntryQuery>
2371 <ClassifiedByBranch>
2372 <ClassificationFilter>
2373 <Clause>
2374 <SimpleClause leftArgument = "code">
2375 <StringClause stringPredicate = "Equal">321118</StringClause>
2376 </SimpleClause>
2377 </Clause>
2378 </ClassificationFilter>
2379 <ClassificationSchemeQuery>
2380 <NameBranch>
2381 <LocalizedStringFilter>
2382 <Clause>
2383 <SimpleClause leftArgument = "value">
2384 <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
2385 </SimpleClause>
2386 </Clause>
2387 </LocalizedStringFilter>
2388 </NameBranch>
2389 </ClassificationSchemeQuery>
2390 </ClassifiedByBranch>
2391 <RegistryEntryFilter>
2392 <Clause>
2393 <CompoundClause connectivePredicate = "And">
2394 <Clause>
2395 <SimpleClause leftArgument = "objectType">
2396 <StringClause stringPredicate = "Equal">CPP</StringClause>
2397 </SimpleClause>
2398 </Clause>
2399 <Clause>
2400 <SimpleClause leftArgument = "status">
2401 <StringClause stringPredicate = "Equal">Approved</StringClause>
2402 </SimpleClause>
2403 </Clause>
2404 </CompoundClause>
2405

```

```

2406     </Clause>
2407     </RegistryEntryFilter>
2408     </RegistryEntryQuery>
2409     </FilterQuery>
2410 </AdhocQueryRequest>
2411

```

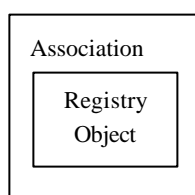
2412 8.2.4 AssociationQuery

2413 Purpose

2414 To identify a set of association instances as the result of a query over selected registry metadata.

2415

2416 ebRIM Binding



2417  **Figure 28: ebRIM Binding for AssociationQuery**

2418 Definition

```

2419
2420 <complexType name = "AssociationQueryType">
2421   <complexContent>
2422     <extension base = "tns:RegistryObjectQueryType">
2423       <sequence>
2424         <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
2425       </sequence>
2426     </extension>
2427   </complexContent>
2428 </complexType>
2429 <element name = "AssociationQuery" type = "tns:AssociationQueryType"/>
2430
2431 <element name="AssociationQueryResult">
2432   <complexType>
2433     <choice minOccurs="0" maxOccurs="unbounded">
2434       <element ref="rim:ObjectRef" />
2435       <element ref="rim:RegistryObject" />
2436       <element ref="rim:Association" />
2437     </choice>
2438   </complexType>
2439 </element>
2440

```

2441 Semantic Rules

- 2442 1. Let A denote the set of all persistent Association instances in the Registry. The following
- 2443 steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
- 2444 a) If A is empty then continue to the next numbered rule.

- 2445 b) If an AssociationFilter element is not directly contained in the AssociationQuery element,
 2446 then go to the next step; otherwise let x be an association instance in A. If x does not
 2447 satisfy the AssociationFilter then remove x from A. If A is empty then continue to the
 2448 next numbered rule.
- 2449 c) Let A be the set of remaining Association instances. Evaluate inherited
 2450 RegistryObjectQuery over A as explained in Section 8.2.2.
- 2451 2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to
 2452 be the result of the AssociationQuery.
- 2453 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2454 within the RegistryResponse.

2455 **Examples**

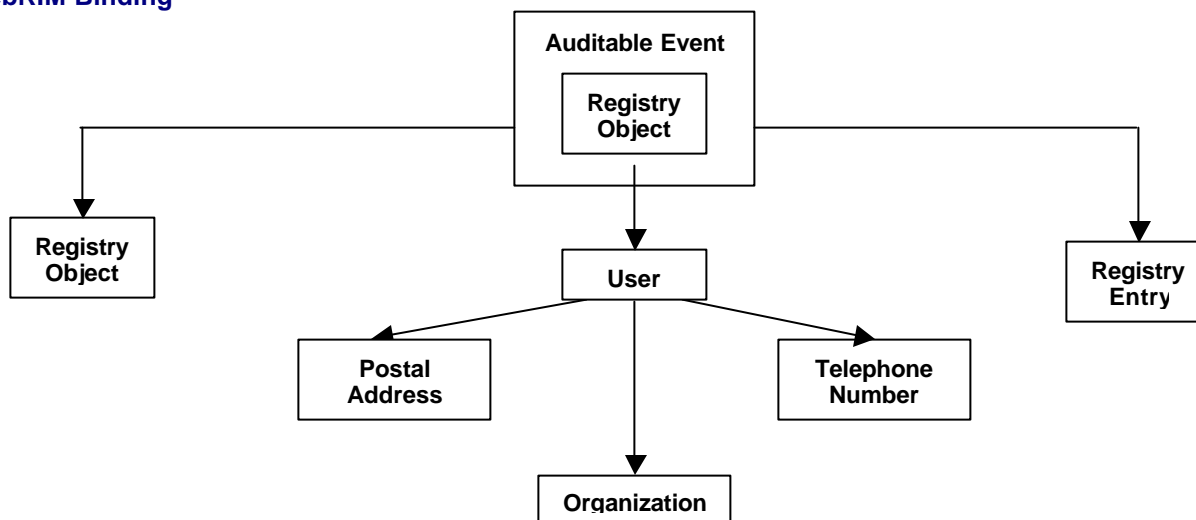
2456 A client application wishes to identify a set of associations that are 'equivalentTo' a set of other
 2457 associations.

```

2458
2459 <AdhocQueryRequest">
2460   <ResponseOption returnType="LeafClass" />
2461   <FilterQuery>
2462     <AssociationQuery>
2463       <SourceAssociationBranch>
2464         <AssociationFilter>
2465           <Clause>
2466             <SimpleClause leftArgument="associationType">
2467               <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
2468             </SimpleClause>
2469           </Clause>
2470         </AssociationFilter>
2471       </AssociationQuery>
2472     <AssociationFilter>
2473       <Clause>
2474         <SimpleClause leftArgument="associationType">
2475           <StringClause stringPredicate="StartsWith">Sin</StringClause>
2476         </SimpleClause>
2477       </Clause>
2478     </AssociationFilter>
2479   </AssociationQuery>
2480 </SourceAssociationBranch>
2481 <AssociationFilter>
2482   <Clause>
2483     <SimpleClause leftArgument="associationType">
2484       <StringClause stringPredicate="StartsWith">Son</StringClause>
2485     </SimpleClause>
2486   </Clause>
2487 </AssociationFilter>
2488 </AssociationQuery>
2489 </FilterQuery>
2490 </AdhocQueryRequest>
2491
  
```

2492 **8.2.5 AuditableEventQuery**2493 **Purpose**

2494 To identify a set of auditable event instances as the result of a query over selected registry
2495 metadata.

2496 **ebRIM Binding**

2497 **Figure 29: ebRIM Binding for AuditableEventQuery**

2498 **Definition**

```

2499 <complexType name="AuditableEventQueryType">
2500   <complexContent>
2501     <extension base="tns:RegistryObjectQueryType">
2502       <sequence>
2503         <element ref="tns:AuditableEventFilter" minOccurs="0" />
2504         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2505         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2506         <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2507       </sequence>
2508     </extension>
2509   </complexContent>
2510 </complexType>
2511 <element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />
2512 <element name="AuditableEventQueryResult">
2513   <complexType>
2514     <choice minOccurs="0" maxOccurs="unbounded">
2515       <element ref="rim:ObjectRef" />
2516       <element ref="rim:RegistryObject" />
2517       <element ref="rim:AuditableEvent" />
2518     </choice>
2519   </complexType>
2520 </element>
2521 </element>
2522 </element>
2523 </element>

```

2524 Semantic Rules

- 2525 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
2526 following steps will eliminate instances in AE that do not satisfy the conditions of the
2527 specified filters.
 - 2528 a) If AE is empty then continue to the next numbered rule.
 - 2529 b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an
2530 auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from
2531 AE. If AE is empty then continue to the next numbered rule.
 - 2532 c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
2533 x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:
2534 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is
2535 not an auditable event for some registry object in RO, then remove x from AE. If AE is
2536 empty then continue to the next numbered rule.
 - 2537 d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
2538 be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let
2539 RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an
2540 auditable event for some registry entry in RE, then remove x from AE. If AE is empty
2541 then continue to the next numbered rule.
 - 2542 e) If a UserBranch element is not specified then go to the next step; otherwise, let x be a
2543 remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter
2544 element is specified within the UserBranch, and if u does not satisfy that filter, then
2545 remove x from AE. If a PostalAddressFilter element is specified within the UserBranch,
2546 and if the postal address of u does not satisfy that filter, then remove x from AE. If
2547 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
2548 TelephoneNumberFilters isn't satisfied by all of the telephone numbers of u then remove
2549 x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if any of
2550 the EmailAddressFilters isn't satisfied by all of the email addresses of u then remove x
2551 from AE. If an OrganizationQuery element is specified within the UserBranch, then let o
2552 be the Organization instance that is identified by the organization that u is affiliated with.
2553 If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x from
2554 AE. If AE is empty then continue to the next numbered rule.
 - 2555 f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited
2556 RegistryObjectQuery over AE as explained in Section 8.2.2.
- 2557 2. If AE is empty, then raise the warning: *auditable event query result is empty*; otherwise set
2558 AE to be the result of the AuditableEventQuery.
- 2559 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2560 within the RegistryResponse.

2561 Examples

2562 A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The
2563 client is now interested in all events since the beginning of the year that have impacted that item.
2564 The following query will return a set of AuditableEvent instances for all such events.

```
2565 <AdhocQueryRequest>  
2566
```

```

2567 <ResponseOption returnType = "LeafClass"/>
2568 <FilterQuery>
2569   <AuditableEventQuery>
2570     <AuditableEventFilter>
2571       <Clause>
2572         <SimpleClause leftArgument = "timestamp">
2573           <RationalClause logicalPredicate = "GE">
2574             DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2575           </RationalClause>
2576         </Simple Clause>
2577       </Clause>
2578     </AuditableEventFilter>
2579   <RegistryEntryQuery>
2580     <NameBranch>
2581       <LocalizedStringFilter>
2582         <Clause>
2583           <SimpleClause leftArgument = "value">
2584             <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
2585           </SimpleClause>
2586         </Clause>
2587       </LocalizedStringFilter>
2588     </NameBranch>
2589   </RegistryEntryQuery>
2590 </AuditableEventQuery>
2591 </FilterQuery>
2592 </AdhocQueryRequest
2593

```

2594 A client company has many registered objects in the Registry. The Registry allows events
 2595 submitted by other organizations to have an impact on your registered items, e.g. new
 2596 classifications and new associations. The following query will return a set of identifiers for all
 2597 auditable events, invoked by some other party, that had an impact on an item submitted by
 2598 “myorg”.

```

2600 <AdhocQueryRequest>
2601   <ResponseOption returnType = "LeafClass"/>
2602   <FilterQuery>
2603     <AuditableEventQuery>
2604       <RegistryEntryQuery>
2605         <TargetAssociationBranch>
2606           <AssociationFilter>
2607             <Clause>
2608               <SimpleClause leftArgument = "associationType">
2609                 <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2610               </SimpleClause>
2611             </Clause>
2612           </AssociationFilter>
2613         <OrganizationQuery>
2614           <NameBranch>
2615             <LocalizedStringFilter>
2616               <Clause>
2617                 <SimpleClause leftArgument = "value">
2618                   <StringClause stringPredicate = "Equal">myorg</StringClause>
2619                 </SimpleClause>
2620               </Clause>
2621             </LocalizedStringFilter>
2622           </NameBranch>

```

```

2623     </OrganizationQuery>
2624     </TargetAssociationBranch>
2625 </RegistryEntryQuery>
2626 <UserBranch>
2627   <OrganizationQuery>
2628     <NameBranch>
2629       <LocalizedStringFilter>
2630         <Clause>
2631           <SimpleClause leftArgument = "value">
2632             <StringClause stringPredicate = "-Equal">myorg</StringClause>
2633           </SimpleClause>
2634         </Clause>
2635       </LocalizedStringFilter>
2636     </NameBranch>
2637   </OrganizationQuery>
2638 </UserBranch>
2639 </AuditableEventQuery>
2640 </FilterQuery>
2641 </AdhocQueryRequest>
2642

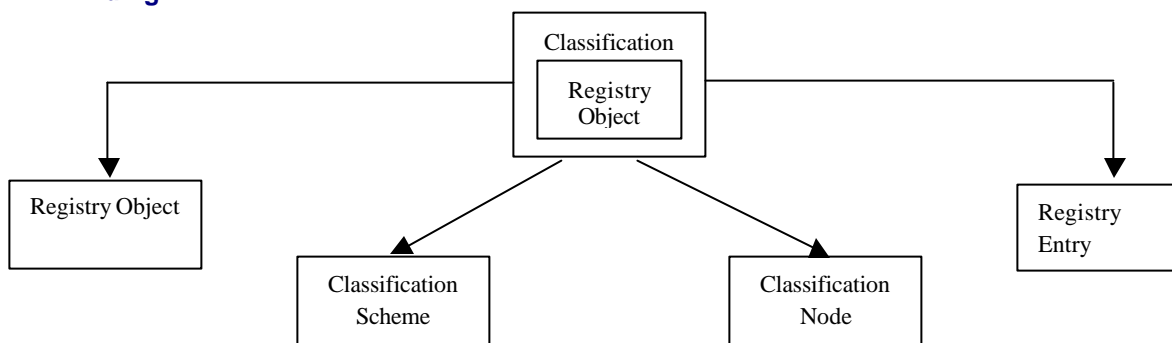
```

2643 8.2.6 ClassificationQuery

2644 Purpose

2645 To identify a set of classification instances as the result of a query over selected registry
 2646 metadata.

2647 ebRIM Binding



2648 **Figure 30: ebRIM Binding for ClassificationQuery**

2649 Definition

```

2650
2651 <complexType name = "ClassificationQueryType">
2652   <complexContent>
2653     <extension base = "tns:RegistryObjectQueryType">
2654       <sequence>
2655         <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
2656         <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
2657         <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
2658         <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
2659         <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
2660       </sequence>
2661     </extension>
2662   </complexContent>

```

```

2663 </complexType>
2664 <element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>
2665
2666 <element name=" ClassificationQueryResult ">
2667   <complexType>
2668     <choice minOccurs="0" maxOccurs="unbounded">
2669       <element ref="rim:ObjectRef" />
2670       <element ref="rim:RegistryObject" />
2671       <element ref="rim:Classification" />
2672     </choice>
2673   </complexType>
2674 </element>
2675

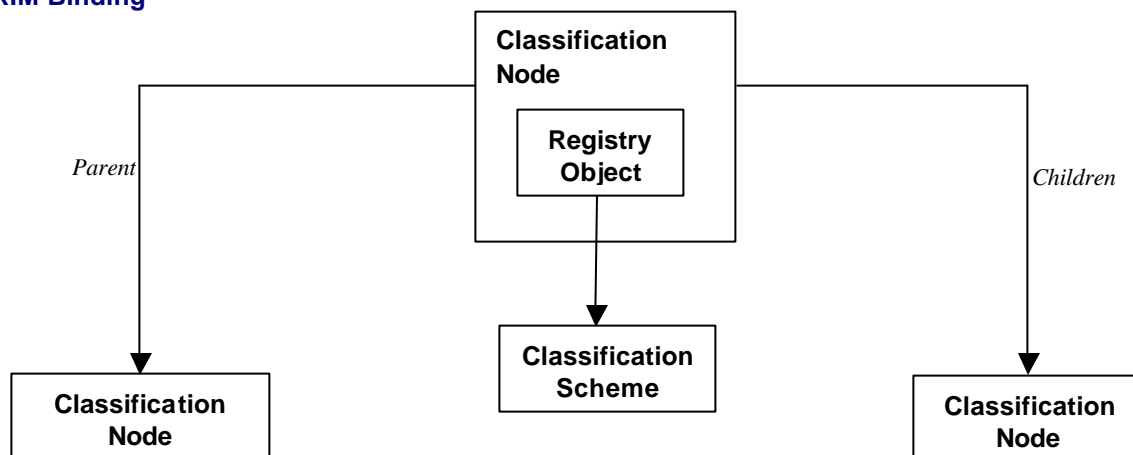
```

2676 Semantic Rules

- 2677 1. Let C denote the set of all persistent Classification instances in the Registry. The following
 2678 steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
 2679
 - 2680 a) If C is empty then continue to the next numbered rule.
 - 2681 b) If a ClassificationFilter element is not directly contained in the ClassificationQuery
 2682 element, then go to the next step; otherwise let x be an classification instance in C. If x
 2683 does not satisfy the ClassificationFilter then remove x from C. If C is empty then
 2684 continue to the next numbered rule.
 - 2685 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
 2686 be a remaining classification in C. If the defining classification scheme of x does not
 2687 satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C.
 2688 If C is empty then continue to the next numbered rule.
 - 2689 d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be
 2690 a remaining classification in C. If the classification node of x does not satisfy the
 2691 ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is
 2692 empty then continue to the next numbered rule.
 - 2693 e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
 2694 x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let
 2695 RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a
 2696 classification of at least one registry object in RO, then remove x from C. If C is empty
 2697 then continue to the next numbered rule.
 - 2698 f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
 2699 be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE
 2700 be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a
 2701 classification of at least one registry entry in RE, then remove x from C. If C is empty
 2702 then continue to the next numbered rule.
- 2702 2. If C is empty, then raise the warning: *classification query result is empty*; otherwise
 2703 otherwise, set C to be the result of the ClassificationQuery.
- 2704 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2705 within the RegistryResponse.

2706 **8.2.7 ClassificationNodeQuery**2707 **Purpose**

2708 To identify a set of classification node instances as the result of a query over selected registry
2709 metadata.

2710 **ebRIM Binding**

2711 **Figure 31: ebRIM Binding for ClassificationNodeQuery**

2712 **Definition**

```

2713 <complexType name="ClassificationNodeQueryType">
2714   <complexContent>
2715     <extension base="tns:RegistryObjectQueryType">
2716       <sequence>
2717         <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
2718         <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
2719         <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
2720           maxOccurs="1" />
2721         <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
2722           minOccurs="0" maxOccurs="unbounded" />
2723       </sequence>
2724     </extension>
2725   </complexContent>
2726 </complexType>
2727 <element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />
2728
2729 <element name="ClassificationNodeQueryResult">
2730   <complexType>
2731     <choice minOccurs="0" maxOccurs="unbounded">
2732       <element ref="rim:ObjectRef" />
2733       <element ref="rim:RegistryObject" />
2734       <element ref="rim:ClassificationNode" />
2735     </choice>
2736   </complexType>
2737 </element>
2738
2739
  
```

2740 **Semantic Rules**

- 2741 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The
2742 following steps will eliminate instances in CN that do not satisfy the conditions of the
2743 specified filters.
- 2744 a) If CN is empty then continue to the next numbered rule.
- 2745 b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a
2746 classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove
2747 x from CN. If CN is empty then continue to the next numbered rule.
- 2748 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
2749 be a remaining classification node in CN. If the defining classification scheme of x does
2750 not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x
2751 from CN. If CN is empty then continue to the next numbered rule.
- 2752 d) If a ClassificationNodeParentBranch element is not specified, then go to the next step;
2753 otherwise, let x be a remaining classification node in CN and execute the following
2754 paragraph with $n=x$.
- 2755 Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base
2756 level node), then remove x from CN and go to the next step; otherwise, let p be the parent
2757 node of n. If a ClassificationNodeFilter element is directly contained in the
2758 ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,
2759 then remove x from CN. If CN is empty then continue to the next numbered rule. If a
2760 ClassificationSchemeQuery element is directly contained in the
2761 ClassificationNodeParentBranch and if defining classification scheme of p does not
2762 satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then
2763 continue to the next numbered rule.
- 2764 If another ClassificationNodeParentBranch element is directly contained within this
2765 ClassificationNodeParentBranch element, then repeat the previous paragraph with $n=p$.
- 2766 e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next
2767 numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the
2768 parent node of some ClassificationNode instance, then remove x from CN and if CN is
2769 empty continue to the next numbered rule; otherwise, treat each
2770 ClassificationNodeChildrenBranch element separately and execute the following
2771 paragraph with $n = x$.
- 2772 Let n be a classification node instance. If a ClassificationNodeFilter element is not
2773 specified within the ClassificationNodeChildrenBranch element then let CNC be the set
2774 of all classification nodes that have n as their parent node; otherwise, let CNC be the set
2775 of all classification nodes that satisfy the ClassificationNodeFilter and have n as their
2776 parent node. If CNC is empty, then remove x from CN and if CN is empty continue to the
2777 next numbered rule; otherwise, let c be any member of CNC. If a
2778 ClassificationSchemeQuery element is directly contained in the
2779 ClassificationNodeChildrenBranch and if the defining classification scheme of c does not
2780 satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then
2781 remove x from CN. If CN is empty then continue to the next numbered rule; otherwise,
2782 let y be an element of CNC and continue with the next paragraph.

2783 If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly
 2784 contain another ClassificationNodeChildrenBranch element, then continue to the next
 2785 numbered rule; otherwise, repeat the previous paragraph with the new
 2786 ClassificationNodeChildrenBranch element and with $n = y$.

2787 f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited
 2788 RegistryObjectQuery over CN as explained in Section 8.2.2.

2789 2. If CN is empty, then raise the warning: *classification node query result is empty*; otherwise
 2790 set CN to be the result of the ClassificationNodeQuery.

2791 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2792 within the RegistryResponse.

2793 Path Filter Expression usage in ClassificationNodeFilter

2794 The path filter expression is used to match classification nodes in ClassificationNodeFilter
 2795 elements involving the path attribute of the ClassificationNode class as defined by the getPath
 2796 method in [ebRIM].

2797 The path filter expressions are based on a very small and proper sub-set of location path syntax
 2798 of XPath.

2799 The path filter expression syntax includes support for matching multiple nodes by using wild
 2800 card syntax as follows:

2801 ?? Use of '*' as a wildcard in place of any path element in the pathFilter

2802 ?? Use of '/' syntax to denote any descendent of a node in the pathFilter

2803 It is defined by the following BNF grammar:

```

2804 pathFilter ::= '/' schemeId nodePath
2805 nodePath ::= slashes nodeCode
2806           | slashes '*'
2807           | slashes nodeCode ( nodePath )?
2808 Slashes   ::= '/' | '/'
2809
2810
```

2811 In the above grammar, schemeId is the id attribute of the ClassificationScheme instance. In the
 2812 above grammar nodeCode is defined by NCName production as defined by
 2813 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

2814 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a
 2815 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause
 2816 is a PATH Filter expression.

2817 This is illustrated in the following example that matches all second level nodes in
 2818 ClassificationScheme with id 'Geography-id' and with code 'Japan':

```

2819 <ClassificationNodeQuery>
2820   <ClassificationNodeFilter>
2821     <Clause>
2822       <SimpleClause leftArgument = "path">
2823         <StringClause stringPredicate = "Equal">//Geography-id/*//Japan</StringClause>
2824       </SimpleClause>
2825     </Clause>
2826   </ClassificationNodeFilter>
2827 </ClassificationNodeQuery>
2828
2829
```

2830 **Use Cases and Examples of Path Filter Expressions**

2831 The following table lists various use cases and examples using the sample Geography scheme
 2832 below:

```

2833 <ClassificationScheme id='Geography-id' name="Geography" />
2834
2835 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
2836 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
2837
2838 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
2839 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
2840 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
2841
2842
    
```

2843  **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'

2844 **Examples**

2845 A client application wishes to identify all of the classification nodes in the first three levels of a
 2846 classification scheme hierarchy. The client knows that the name of the underlying classification
 2847 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three
 2848 levels.

```

2849
2850 <AdhocQueryRequest>
2851   <ResponseOption returnType = "LeafClass"/>
2852   <FilterQuery>
2853     <ClassificationNodeQuery>
2854       <ClassificationNodeFilter>
2855         <Clause>
2856           <SimpleClause leftArgument = "levelNumber">
2857             <RationalClause logicalPredicate = "LE">
2858               <IntClause>3</IntClause>
    
```

```

2859     </RationalClause>
2860     </SimpleClause>
2861   </Clause>
2862 </ClassificationNodeFilter>
2863 <ClassificationSchemeQuery>
2864   <NameBranch>
2865     <LocalizedStringFilter>
2866       <Clause>
2867         <SimpleClause leftArgument = "value">
2868           <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
2869         </SimpleClause>
2870       </Clause>
2871     </LocalizedStringFilter>
2872   </NameBranch>
2873 </ClassificationSchemeQuery>
2874 </ClassificationNodeQuery>
2875 </FilterQuery>
2876 </AdhocQueryRequest>
2877

```

2878 If, instead, the client wishes all levels returned, they could simply delete the
 2879 ClassificationNodeFilter element from the query.

2880 The following query finds all children nodes of a first level node whose code is NorthAmerica.

```

2881 <AdhocQueryRequest>
2882   <ResponseOption returnType = "LeafClass"/>
2883   <FilterQuery>
2884     <ClassificationNodeQuery>
2885       <ClassificationNodeFilter>
2886         <Clause>
2887           <SimpleClause leftArgument = "path">
2888             <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
2889           </SimpleClause>
2890         </Clause>
2891       </ClassificationNodeFilter>
2892     </ClassificationNodeQuery>
2893   </FilterQuery>
2894 </AdhocQueryRequest>
2895
2896

```

2897 The following query finds all third level nodes with code of Tokyo.

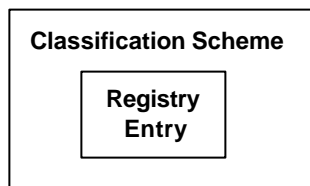
```

2898 <AdhocQueryRequest>
2899   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2900   <FilterQuery>
2901     <ClassificationNodeQuery>
2902       <ClassificationNodeFilter>
2903         <Clause>
2904           <SimpleClause leftArgument = "path">
2905             <StringClause stringPredicate = "Equal">/Geography-id/*/*Tokyo</StringClause>
2906           </SimpleClause>
2907         </Clause>
2908       </ClassificationNodeFilter>
2909     </ClassificationNodeQuery>
2910   </FilterQuery>
2911 </AdhocQueryRequest>
2912
2913

```

2914 **8.2.8 ClassificationSchemeQuery**2915 **Purpose**

2916 To identify a set of classification scheme instances as the result of a query over selected registry
2917 metadata.

2918 **ebRIM Binding**2919 **Figure 32: ebRIM Binding for ClassificationSchemeQuery**2920 **Definition**

```

2921 <complexType name="ClassificationSchemeQueryType">
2922   <complexContent>
2923     <extension base="tns:RegistryEntryQueryType">
2924       <sequence>
2925         <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
2926       </sequence>
2927     </extension>
2928   </complexContent>
2929 </complexType>
2930 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
2931
2932
  
```

2933 **Semantic Rules**

- 2934 1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The
2935 following steps will eliminate instances in CS that do not satisfy the conditions of the
2936 specified filters.
- 2937 a) If CS is empty then continue to the next numbered rule.
- 2938 b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x
2939 be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,
2940 then remove x from CS. If CS is empty then continue to the next numbered rule.
- 2941 c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited
2942 RegistryEntryQuery over CS as explained in Section 8.2.3.
- 2943 2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,
2944 set CS to be the result of the ClassificationSchemeQuery.
- 2945 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2946 within the RegistryResponse.

2947 **Examples**

2948 A client application wishes to identify all classification scheme instances in the Registry.

```

2949 <AdhocQueryRequest>
2950   <ResponseOption returnType = "LeafClass"/>
2951   <FilterQuery>
  
```

```

2952 <ClassificationSchemeQuery/>
2953 </FilterQuery>
2954 </AdhocQueryRequest>

```

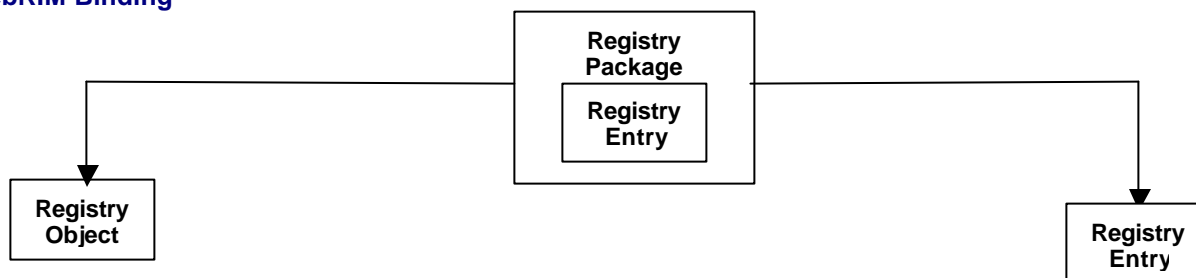
2955

2956 8.2.9 RegistryPackageQuery

2957 Purpose

2958 To identify a set of registry package instances as the result of a query over selected registry
 2959 metadata.

2960 ebRIM Binding



2961

☞☞ Figure 33: ebRIM Binding for RegistryPackageQuery

2962 Definition

```

2963 <complexType name="RegistryPackageQueryType">
2964 <complexContent>
2965 <extension base="tns:RegistryEntryQueryType">
2966 <sequence>
2967 <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
2968 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
2969 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
2970 </sequence>
2971 </extension>
2972 </complexContent>
2973 </complexType>
2974 <element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />
2975
2976 <element name="RegistryPackageQueryResult">
2977 <complexType>
2978 <choice minOccurs="0" maxOccurs="unbounded">
2979 <element ref="rim:ObjectRef" />
2980 <element ref="rim:RegistryEntry" />
2981 <element ref="rim:RegistryObject" />
2982 <element ref="rim:RegistryPackage" />
2983 </choice>
2984 </complexType>
2985 </element>
2986
2987

```

2988 **Semantic Rules**

- 2989 1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The
 2990 following steps will eliminate instances in RP that do not satisfy the conditions of the
 2991 specified filters.
- 2992 a) If RP is empty then continue to the next numbered rule.
- 2993 b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule;
 2994 otherwise, let x be a registry package instance in RP. If x does not satisfy the
 2995 RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next
 2996 numbered rule.
- 2997 c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery
 2998 element then treat each RegistryObjectQuery as follows: let RO be the set of
 2999 RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2
 3000 and let PO be the subset of RO that are members of the package x. If PO is empty, then
 3001 remove x from RP. If RP is empty then continue to the next numbered rule. If a
 3002 RegistryEntryQuery element is directly contained in the RegistryPackageQuery element
 3003 then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry
 3004 instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be
 3005 the subset of RE that are members of the package x. If PE is empty, then remove x from
 3006 RP. If RP is empty then continue to the next numbered rule.
- 3007 d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited
 3008 RegistryEntryQuery over RP as explained in Section 8.2.3.
- 3009 2. If RP is empty, then raise the warning: *registry package query result is empty*; otherwise set
 3010 RP to be the result of the RegistryPackageQuery.
- 3011 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 3012 within the RegistryResponse.

3013 **Examples**

3014 A client application wishes to identify all package instances in the Registry that contain an
 3015 Invoice extrinsic object as a member of the package.

```

3016 <AdhocQueryRequest>
3017   <ResponseOption returnType = "LeafClass"/>
3018   <FilterQuery>
3019     <RegistryPackageQuery>
3020       <RegistryEntryQuery>
3021         <RegistryEntryFilter>
3022           <Clause>
3023             <SimpleClause leftArgument = "objectType">
3024               <StringClause stringPredicate = "Equal">Invoice</StringClause>
3025             </SimpleClause>
3026           </Clause>
3027         </RegistryEntryFilter>
3028       </RegistryEntryQuery>
3029     </RegistryPackageQuery>
3030   </FilterQuery>
3031 </AdhocQueryRequest>
3032
3033
```


3034 A client application wishes to identify all package instances in the Registry that are not empty.

```
3035
3036 <AdhocQueryRequest>
3037   <ResponseOption returnType = "LeafClass"/>
3038   <FilterQuery>
3039     <RegistryPackageQuery>
3040       <RegistryObjectQuery/>
3041     </RegistryPackageQuery>
3042   </FilterQuery>
3043 </AdhocQueryRequest>
3044
```

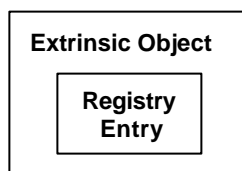
3045 A client application wishes to identify all package instances in the Registry that are empty. Since
 3046 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate
 3047 RegistryPackageQuery requests, one to find all packages and another to find all non-empty
 3048 packages, and then do the set difference themselves. Alternatively, they could do a more
 3049 complex RegistryEntryQuery and check that the packaging association between the package and
 3050 its members is non-existent.

3051 Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by
 3052 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an
 3053 equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However,
 3054 the equivalent RegistryEntryQuery is often more complicated to write.

3055 8.2.10 ExtrinsicObjectQuery

3056 Purpose

3057 To identify a set of extrinsic object instances as the result of a query over selected registry
 3058 metadata.



3059 ebRIM Binding

3060  **Figure 34: ebRIM Binding for ExtrinsicObjectQuery**

3061 Definition

```
3062
3063 <complexType name="ExtrinsicObjectQueryType">
3064   <complexContent>
3065     <extension base="tns:RegistryEntryQueryType">
3066       <sequence>
3067         <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
3068       </sequence>
3069     </extension>
3070   </complexContent>
3071 </complexType>
3072 <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
3073
```

```
3074 <element name="ExtrinsicObjectQueryResult">
3075   <complexType>
3076     <choice minOccurs="0" maxOccurs="unbounded">
3077       <element ref="rim:ObjectRef" />
3078       <element ref="rim:RegistryEntry" />
3079       <element ref="rim:RegistryObject" />
3080       <element ref="rim:ExtrinsicObject" />
3081     </choice>
3082   </complexType>
3083 </element>
3084
```

3085 **Semantic Rules**

- 3086 1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The
3087 following steps will eliminate instances in EO that do not satisfy the conditions of the
3088 specified filters.
 - 3089 a) If EO is empty then continue to the next numbered rule.
 - 3090 b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an
3091 extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from
3092 EO. If EO is empty then continue to the next numbered rule.
 - 3093 c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited
3094 RegistryEntryQuery over EO as explained in Section 8.2.3.
- 3095 2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set
3096 EO to be the result of the ExtrinsicObjectQuery.
- 3097 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
3098 within the RegistryResponse.

3099 **8.2.11 OrganizationQuery**

3100 **Purpose**

3101 To identify a set of organization instances as the result of a query over selected registry
3102 metadata.

3103 **ebRIM Binding**

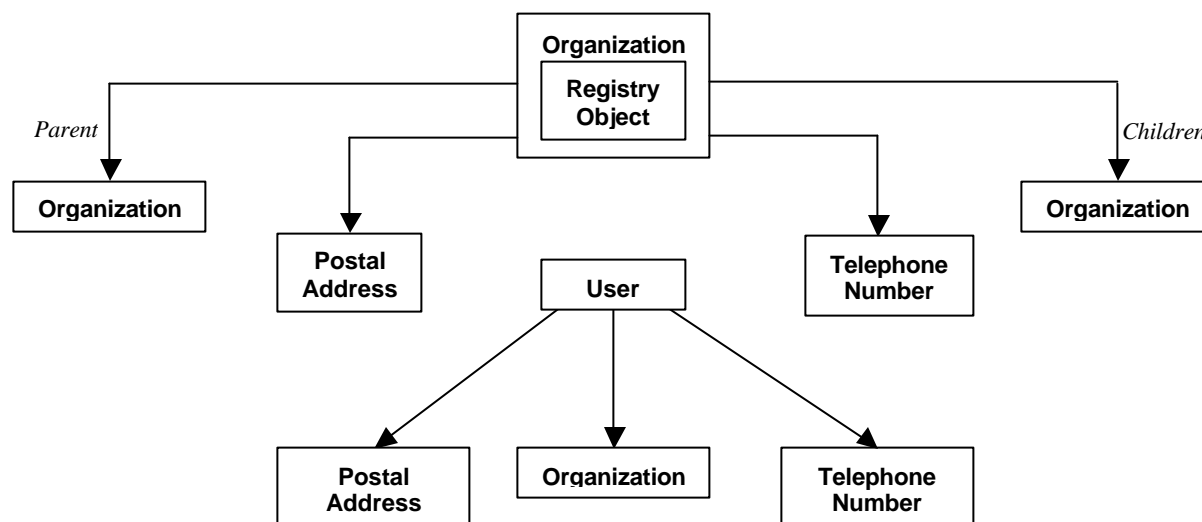


Figure 35: ebRIM Binding for OrganizationQuery

3104

Definition

3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134

```

<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 name="tns:UserBranch" minOccurs="0" maxOccurs="1" />
        <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0"
          maxOccurs="1" />
        <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="OrganizationQuery" type="tns:OrganizationQueryType" />

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

Semantic Rules

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

- 3140 b) If an OrganizationFilter element is not directly contained in the OrganizationQuery
3141 element, then go to the next step; otherwise let x be an organization instance in ORG. If x
3142 does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then
3143 continue to the next numbered rule.
- 3144 c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery
3145 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal
3146 address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is
3147 empty then continue to the next numbered rule.
- 3148 d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery
3149 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of
3150 the TelephoneNumberFilters isn't satisfied by all of the telephone numbers of x then
3151 remove x from ORG. If ORG is empty then continue to the next numbered rule.
- 3152 e) If a UserBranch element is not directly contained in the OrganizationQuery element then
3153 go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user
3154 instance that is affiliated with x. If a UserFilter element is specified within the
3155 UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a
3156 PostalAddressFilter element is specified within the UserBranch, and if the postal address
3157 of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s)
3158 are specified within the UserBranch and if any of the TelephoneNumberFilters isn't
3159 satisfied by all of the telephone numbers of x then remove x from ORG. If
3160 EmailAddressFilter(s) are specified within the UserBranch and if any of the
3161 EmailAddressFilters isn't satisfied by all of the email addresses of x then remove x from
3162 ORG. If an OrganizationQuery element is specified within the UserBranch, then let o be
3163 the Organization instance that is identified by the organization that u is affiliated with. If
3164 o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x from
3165 ORG. If ORG is empty then continue to the next numbered rule.
- 3166 f) If a OrganizationParentBranch element is not specified within the OrganizationQuery,
3167 then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the
3168 following paragraph with o = x:
3169 Let o be an organization instance. If an OrganizationFilter is not specified within the
3170 OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the
3171 Organization hierarchy), then remove x from ORG; otherwise, let p be the parent
3172 organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.
3173 If ORG is empty then continue to the next numbered rule.
3174 If another OrganizationParentBranch element is directly contained within this
3175 OrganizationParentBranch element, then repeat the previous paragraph with o = p.
- 3176 g) If a OrganizationChildrenBranch element is not specified, then continue to the next
3177 numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent
3178 node of some organization instance, then remove x from ORG and if ORG is empty
3179 continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch
3180 element separately and execute the following paragraph with n = x.

- 3181 Let n be an organization instance. If an `OrganizationFilter` element is not specified within
 3182 the `OrganizationChildrenBranch` element then let `ORGC` be the set of all organizations
 3183 that have n as their parent node; otherwise, let `ORGC` be the set of all organizations that
 3184 satisfy the `OrganizationFilter` and have n as their parent node. If `ORGC` is empty, then
 3185 remove x from `ORG` and if `ORG` is empty continue to the next numbered rule; otherwise,
 3186 let c be any member of `ORGC`. If a `PostalAddressFilter` element is directly contained in
 3187 the `OrganizationChildrenBranch` and if the postal address of c does not satisfy the
 3188 `PostalAddressFilter` then remove c from `ORGC`. If `ORGC` is empty then remove x from
 3189 `ORG`. If `ORG` is empty then continue to the next numbered rule. If no
 3190 `TelephoneNumberFilter` element is directly contained in the `OrganizationChildrenBranch`
 3191 and if any of the `TelephoneNumberFilters` isn't satisfied by all of the telephone numbers
 3192 of c then remove c from `ORGC`. If `ORGC` is empty then remove x from `ORG`. If `ORG` is
 3193 empty then continue to the next numbered rule; otherwise, let y be an element of `ORGC`
 3194 and continue with the next paragraph.
- 3195 If the `OrganizationChildrenBranch` element is terminal, i.e. if it does not directly contain
 3196 another `OrganizationChildrenBranch` element, then continue to the next numbered rule;
 3197 otherwise, repeat the previous paragraph with the new `OrganizationChildrenBranch`
 3198 element and with $n = y$.
- 3199 h) Let `ORG` be the set of remaining `Organization` instances. Evaluate inherited
 3200 `RegistryObjectQuery` over `ORG` as explained in Section 8.2.2.
- 3201 2. If `ORG` is empty, then raise the warning: *organization query result is empty*; otherwise set
 3202 `ORG` to be the result of the `OrganizationQuery`.
- 3203 3. Return the result and any accumulated warnings or exceptions (in the `RegistryErrorList`)
 3204 within the `RegistryResponse`.

3205 Examples

3206 A client application wishes to identify a set of organizations, based in France, that have
 3207 submitted a `PartyProfile` extrinsic object this year.

```

3208
3209 <AdhocQueryRequest>
3210   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
3211   <FilterQuery>
3212     <OrganizationQuery>
3213       <SourceAssociationBranch>
3214         <AssociationFilter>
3215           <Clause>
3216             <SimpleClause leftArgument = "associationType">
3217               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
3218             </SimpleClause>
3219           </Clause>
3220         </AssociationFilter>
3221       <RegistryObjectQuery>
3222         <RegistryObjectFilter>
3223           <Clause>
3224             <SimpleClause leftArgument = "objectType">
3225               <StringClause stringPredicate = "Equal">CPP</StringClause>
3226             </SimpleClause>
3227           </Clause>
3228         </RegistryObjectFilter>
3229       <AuditableEventQuery>
  
```

```

3230         <AuditableEventFilter>
3231         <Clause>
3232             <SimpleClause leftArgument = "timestamp">
3233                 <RationalClause logicalPredicate = "GE">
3234                     <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
3235                 </RationalClause>
3236             </SimpleClause>
3237         </Clause>
3238     </AuditableEventFilter>
3239 </AuditableEventQuery>
3240 </RegistryObjectQuery>
3241 </SourceAssociationBranch>
3242 <PostalAddressFilter>
3243     <Clause>
3244         <SimpleClause leftArgument = "country">
3245             <StringClause stringPredicate = "Equal">France</StringClause>
3246         </SimpleClause>
3247     </Clause>
3248 </PostalAddressFilter>
3249 </OrganizationQuery>
3250 </FilterQuery>
3251 </AdhocQueryRequest>
3252

```

3253 A client application wishes to identify all organizations that have Corporation named XYZ as a
3254 parent.

```

3255
3256 <AdhocQueryRequest>
3257     <ResponseOption returnType = "LeafClass"/>
3258     <FilterQuery>
3259         <OrganizationQuery>
3260             <OrganizationParentBranch>
3261                 <NameBranch>
3262                     <LocalizedStringFilter>
3263                         <Clause>
3264                             <SimpleClause leftArgument = "value">
3265                                 <StringClause stringPredicate = "Equal">XYZ</StringClause>
3266                             </SimpleClause>
3267                         </Clause>
3268                     </LocalizedStringFilter>
3269                 </NameBranch>
3270             </OrganizationParentBranch>
3271         </OrganizationQuery>
3272     </FilterQuery>
3273 </AdhocQueryRequest>
3274

```

3275 8.2.12 ServiceQuery

3276 Purpose

3277

3278 To identify a set of service instances as the result of a query over selected registry metadata.

3279 ebRIM Binding

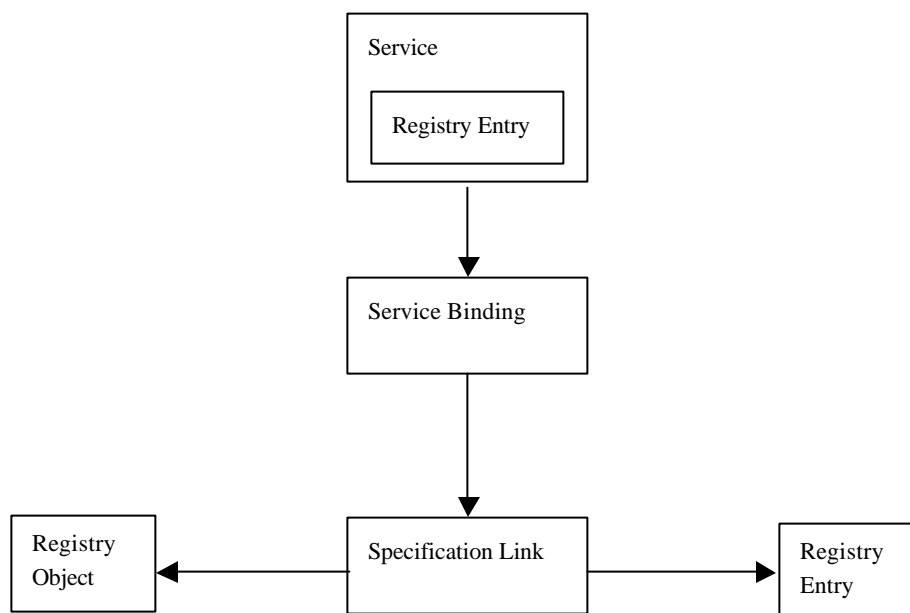


Figure 36: ebRIM Binding for ServiceQuery

3280

Definition

3281

3282

3283

3284

3285

3286

3287

3288

3289

3290

3291

3292

3293

3294

3295

3296

3297

3298

3299

3300

3301

3302

3303

3304

3305

3306

```

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

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

Semantic Rules

3307

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

3309

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

3310

- 3311 b) If a ServiceFilter is not specified then go to the next step; otherwise, let x be a service in
 3312 S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then
 3313 continue to the next numbered rule.
- 3314 c) If a ServiceBindingBranch is not specified then continue to the next numbered rule;
 3315 otherwise, consider each ServiceBindingBranch element separately as follows:
 3316 Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the
 3317 member of SB. If a ServiceBindingFilter element is specified within the
 3318 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If
 3319 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
 3320 If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then
 3321 continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch
 3322 element separately as follows:
 3323 Let sb be a remaining service binding in SB. Let SL be the set of all specification link
 3324 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
 3325 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
 3326 remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove
 3327 x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery
 3328 element is specified within the SpecificationLinkBranch then let sl be a remaining
 3329 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
 3330 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
 3331 specification link for some registry object in RO, then remove sl from SL. If SL is empty
 3332 then remove sb from SB. If SB is empty then remove x from S. If S is empty then
 3333 continue to the next numbered rule. If a RegistryEntryQuery element is specified within
 3334 the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
 3335 RegistryEntryQuery element as follows: Let RE be the result set of the
 3336 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
 3337 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If
 3338 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
- 3339 d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery
 3340 over AE as explained in Section 8.2.3.
- 3341 2. If S is empty, then raise the warning: *service query result is empty*; otherwise set S to be the
 3342 result of the ServiceQuery.
- 3343 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 3344 within the RegistryResponse.

3345 Examples

3346

3347 8.2.13 Registry Filters

3348 Purpose

3349 To identify a subset of the set of all persistent instances of a given registry class.

3350 Definition

3351 `<complexType name="FilterType">`
 3352


```

3353 <sequence>
3354   <element ref="tns:Clause" />
3355 </sequence>
3356 </complexType>
3357 <element name="RegistryObjectFilter" type="tns:FilterType" />
3358 <element name="RegistryEntryFilter" type="tns:FilterType" />
3359 <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
3360 <element name="RegistryPackageFilter" type="tns:FilterType" />
3361 <element name="OrganizationFilter" type="tns:FilterType" />
3362 <element name="ClassificationNodeFilter" type="tns:FilterType" />
3363 <element name="AssociationFilter" type="tns:FilterType" />
3364 <element name="ClassificationFilter" type="tns:FilterType" />
3365 <element name="ClassificationSchemeFilter" type="tns:FilterType" />
3366 <element name="ExternalLinkFilter" type="tns:FilterType" />
3367 <element name="ExternalIdentifierFilter" type="tns:FilterType" />
3368 <element name="SlotFilter" type="tns:FilterType" />
3369 <element name="AuditableEventFilter" type="tns:FilterType" />
3370 <element name="UserFilter" type="tns:FilterType" />
3371 <element name="SlotValueFilter" type="tns:FilterType" />
3372 <element name="PostalAddressFilter" type="tns:FilterType" />
3373 <element name="TelephoneNumberFilter" type="tns:FilterType" />
3374 <element name="EmailAddressFilter" type="tns:FilterType" />
3375 <element name="ServiceFilter" type="tns:FilterType" />
3376 <element name="ServiceBindingFilter" type="tns:FilterType" />
3377 <element name="SpecificationLinkFilter" type="tns:FilterType" />
3378 <element name="LocalizedStringFilter" type="tns:FilterType" />
3379

```

3380 Semantic Rules

- 3381 1. The Clause element is defined in Section 8.2.14.
- 3382 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
3383 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in
3384 [ebRIM]. If not, raise exception: *registry object attribute error*. The RegistryObjectFilter
3385 returns a set of identifiers for RegistryObject instances whose attribute values evaluate to
3386 *True* for the Clause predicate.
- 3387 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing
3388 SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in
3389 [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
3390 returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
3391 for the Clause predicate.
- 3392 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
3393 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in
3394 [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter
3395 returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to
3396 *True* for the Clause predicate.
- 3397 5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing
3398 SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in
3399 [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns
3400 a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for
3401 the Clause predicate.

- 3402 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing
3403 SimpleClause shall identify a public attribute of the Organization or PostalAddress UML
3404 classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The
3405 OrganizationFilter returns a set of identifiers for Organization instances whose attribute
3406 values evaluate to *True* for the Clause predicate.
- 3407 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
3408 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
3409 class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the
3410 leftAttribute is the visible attribute “path” then if stringPredicate of the StringClause is not
3411 “Equal” then raise exception: *classification node path attribute error*. The
3412 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose
3413 attribute values evaluate to *True* for the Clause predicate.
- 3414 8. For every AssociationFilter XML element, the leftArgument attribute of any containing
3415 SimpleClause shall identify a public attribute of the Association UML class defined in
3416 [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a
3417 set of identifiers for Association instances whose attribute values evaluate to *True* for the
3418 Clause predicate.
- 3419 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing
3420 SimpleClause shall identify a public attribute of the Classification UML class defined in
3421 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter
3422 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*
3423 for the Clause predicate.
- 3424 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any
3425 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
3426 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The
3427 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances
3428 whose attribute values evaluate to *True* for the Clause predicate.
- 3429 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing
3430 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in
3431 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns
3432 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the
3433 Clause predicate.
- 3434 12. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing
3435 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in
3436 [ebRIM]. If not, raise exception: *external identifier attribute error*. The
3437 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose
3438 attribute values evaluate to *True* for the Clause predicate.
- 3439 13. For every SlotFilter XML element, the leftArgument attribute of any containing
3440 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If
3441 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot
3442 instances whose attribute values evaluate to *True* for the Clause predicate.

- 3443 14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
3444 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in
3445 [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter
3446 returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to
3447 *True* for the Clause predicate.
- 3448 15. For every UserFilter XML element, the leftArgument attribute of any containing
3449 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If
3450 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User
3451 instances whose attribute values evaluate to *True* for the Clause predicate.
- 3452 16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one
3453 SlotValue instance for each “value” in the “values” list of a Slot instance. The visible
3454 attribute of SlotValue is “value”. It is a character string. The dynamic instances of SlotValue
3455 are derived from the “values” attribute defined in ebRIM for a Slot instance. For every
3456 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause
3457 shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception:
3458 *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value”
3459 attribute evaluates to *True* for the Clause predicate.
- 3460 17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing
3461 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in
3462 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter
3463 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*
3464 for the Clause predicate.
- 3465 18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any
3466 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML
3467 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.
3468 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances
3469 whose attribute values evaluate to *True* for the Clause predicate.
- 3470 19. For every EmailAddressFilter XML element, the leftArgument attribute of any containing
3471 SimpleClause shall identify a public attribute of the EmailAddress UML class defined in
3472 [ebRIM]. If not, raise exception: *email address attribute error*. The EmailAddressFilter
3473 returns a set of identifiers for EmailAddress instances whose attribute values evaluate to
3474 *True* for the Clause predicate.
- 3475 20. For every ServiceFilter XML element, the leftArgument attribute of any containing
3476 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].
3477 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for
3478 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 3479 21. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
3480 SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in
3481 [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter
3482 returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to
3483 *True* for the Clause predicate.

- 3484 22. For every SpecificationLinkFilter XML element, the leftArgument attribute of any
 3485 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class
 3486 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The
 3487 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose
 3488 attribute values evaluate to *True* for the Clause predicate.
- 3489 23. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing
 3490 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in
 3491 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter
 3492 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to
 3493 *True* for the Clause predicate.

3494 **8.2.14 XML Clause Constraint Representation**

3495 **Purpose**

3496 The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses.
 3497 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to
 3498 simply as Clauses in this specification.

3499 **Conceptual Diagram**

3500 The following is a conceptual diagram outlining the Clause structure.

3501

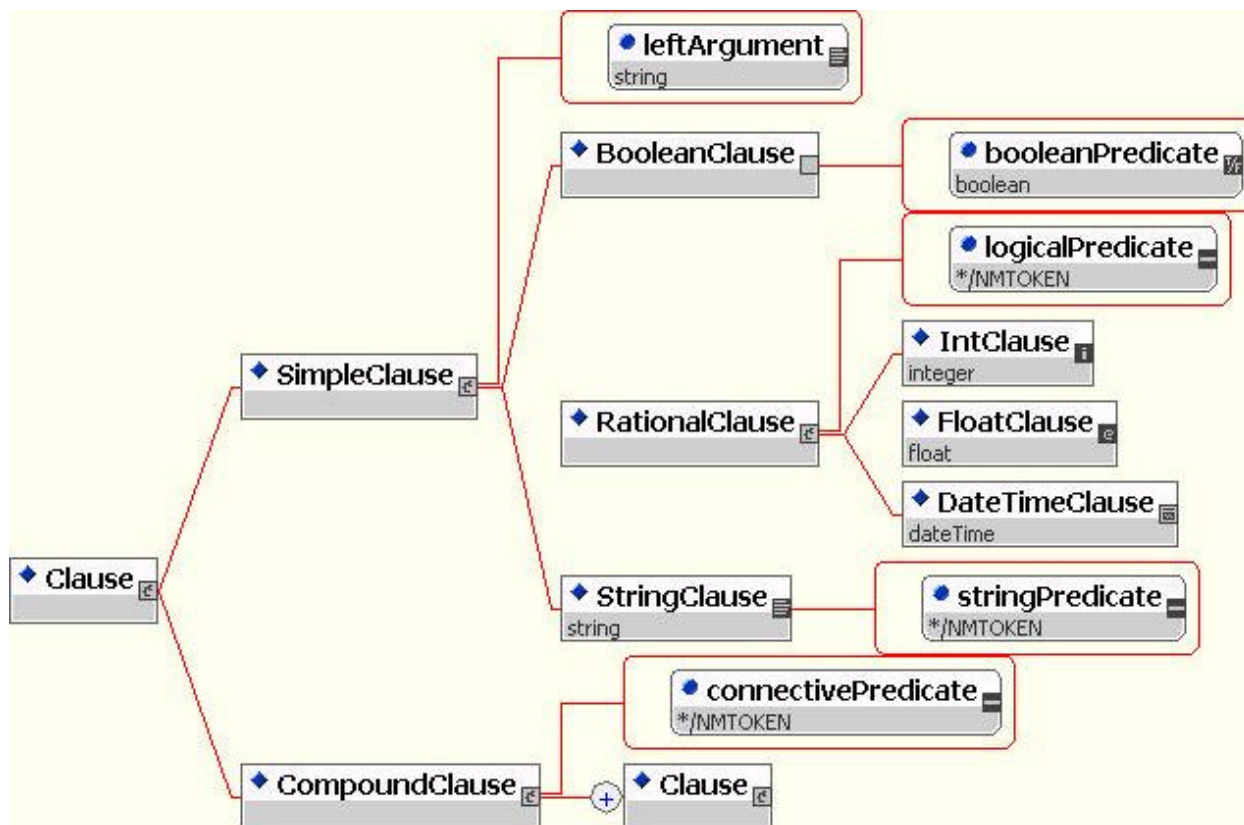


Figure 37: The Clause Structure

3502
3503

3504 **Semantic Rules**

3505 Predicates and Arguments are combined into a "LeftArgument - Predicate - RightArgument"
 3506 format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.
 3507 SimpleClauses

3508 A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the
 3509 Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.
 3510 SimpleClause is extended to support BooleanClause, StringClause, and RationalClause
 3511 (abstract).

3512 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
 3513 boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-
 3514 compare operations and a right argument as the element's text data. Rational number support is
 3515 provided through a common RationalClause providing an enumeration of appropriate rational
 3516 number compare operations, which is further extended to IntClause and FloatClause, each with
 3517 appropriate signatures for the right argument.

3518 CompoundClauses

3519 A CompoundClause contains two or more Clauses (Simple or Compound) and a connective
 3520 predicate. This provides for arbitrarily complex Clauses to be formed.

3521 **Definition**

3522
 3523 <element name = "Clause">
 3524 <annotation>
 3525 <documentation xml:lang = "en">
 3526 The following lines define the XML syntax for Clause.

3527
 3528 </documentation>
 3529 </annotation>
 3530 <complexType>
 3531 <choice>
 3532 <element ref = "tns:SimpleClause"/>
 3533 <element ref = "tns:CompoundClause"/>
 3534 </choice>
 3535 </complexType>
 3536 </element>
 3537 <element name = "SimpleClause">
 3538 <complexType>
 3539 <choice>
 3540 <element ref = "tns:BooleanClause"/>
 3541 <element ref = "tns:RationalClause"/>
 3542 <element ref = "tns:StringClause"/>
 3543 </choice>
 3544 <attribute name = "leftArgument" use = "required" type = "string"/>
 3545 </complexType>
 3546 </element>
 3547 <element name = "CompoundClause">
 3548 <complexType>
 3549 <sequence>
 3550 <element ref = "tns:Clause" maxOccurs = "unbounded"/>
 3551 </sequence>
 3552 <attribute name = "connectivePredicate" use = "required">
 3553 <simpleType>
 3554 <restriction base = "NMTOKEN">

```

3555     <enumeration value = "And"/>
3556     <enumeration value = "Or"/>
3557   </restriction>
3558 </simpleType>
3559 </attribute>
3560 </complexType>
3561 </element>
3562 <element name = "BooleanClause">
3563   <complexType>
3564     <attribute name = "booleanPredicate" use = "required" type = "boolean"/>
3565   </complexType>
3566 </element>
3567 <element name = "RationalClause">
3568   <complexType>
3569     <choice>
3570       <element ref = "tns:IntClause"/>
3571       <element ref = "tns:FloatClause"/>
3572       <element ref = "tns:DateTimeClause"/>
3573     </choice>
3574     <attribute name = "logicalPredicate" use = "required">
3575       <simpleType>
3576         <restriction base = "NMTOKEN">
3577           <enumeration value = "LE"/>
3578           <enumeration value = "LT"/>
3579           <enumeration value = "GE"/>
3580           <enumeration value = "GT"/>
3581           <enumeration value = "EQ"/>
3582           <enumeration value = "NE"/>
3583         </restriction>
3584       </simpleType>
3585     </attribute>
3586   </complexType>
3587 </element>
3588 <element name = "IntClause" type = "integer"/>
3589 <element name = "FloatClause" type = "float"/>
3590 <element name = "DateTimeClause" type = "dateTime"/>
3591
3592 <element name = "StringClause">
3593   <complexType>
3594     <simpleContent>
3595       <extension base = "string">
3596         <attribute name = "stringPredicate" use = "required">
3597           <simpleType>
3598             <restriction base = "NMTOKEN">
3599               <enumeration value = "Contains"/>
3600               <enumeration value = "-Contains"/>
3601               <enumeration value = "StartsWith"/>
3602               <enumeration value = "-StartsWith"/>
3603               <enumeration value = "Equal"/>
3604               <enumeration value = "-Equal"/>
3605               <enumeration value = "EndsWith"/>
3606               <enumeration value = "-EndsWith"/>
3607             </restriction>
3608           </simpleType>
3609         </attribute>
3610       </extension>
3611     </simpleContent>
3612   </complexType>

```

3613 </element>
3614

3615 Examples

3616 Simple BooleanClause: "Smoker" = True

```
3617 <Clause>
3618   <SimpleClause leftArgument="Smoker">
3619     <BooleanClause booleanPredicate="True"/>
3620   </SimpleClause>
3621 </Clause>
3622
3623
```

3624 Simple StringClause: "Smoker" contains "mo"

```
3625 <Clause>
3626   <SimpleClause leftArgument = "Smoker">
3627     <StringClause stringPredicate = "Contains">mo</StringClause>
3628   </SimpleClause>
3629 </Clause>
3630
```

3631 Simple IntClause: "Age" >= 7

```
3632 <Clause>
3633   <SimpleClause leftArgument="Age">
3634     <RationalClause logicalPredicate="GE">
3635       <IntClause>7</IntClause>
3636     </RationalClause>
3637   </SimpleClause>
3638 </Clause>
3639
3640
```

3641 Simple FloatClause: "Size" = 4.3

```
3642 <Clause>
3643   <SimpleClause leftArgument="Size">
3644     <RationalClause logicalPredicate="Equal">
3645       <FloatClause>4.3</FloatClause>
3646     </RationalClause>
3647   </SimpleClause>
3648 </Clause>
3649
3650
```

3651 Compound with two Simples (("Smoker" = False)AND("Age" =< 45))

```
3652 <Clause>
3653   <CompoundClause connectivePredicate="And">
3654     <Clause>
3655       <SimpleClause leftArgument="Smoker">
3656         <BooleanClause booleanPredicate="False"/>
3657       </SimpleClause>
3658     </Clause>
3659     <Clause>
3660       <SimpleClause leftArgument="Age">
3661         <RationalClause logicalPredicate="LE">
3662           <IntClause>45</IntClause>
3663
```

```

3664     </RationalClause>
3665     </SimpleClause>
3666   </Clause>
3667 </CompoundClause>
3668 </Clause>
3669

```

3670 Coumpound with one Simple and one Compound

3671 (("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

```

3672 <Clause>
3673 <CompoundClause connectivePredicate="And">
3674   <Clause>
3675     <SimpleClause leftArgument="Smoker">
3676       <BooleanClause booleanPredicate="False"/>
3677     </SimpleClause>
3678   </Clause>
3679   <Clause>
3680     <CompoundClause connectivePredicate="Or">
3681       <Clause>
3682         <SimpleClause leftArgument="Age">
3683           <RationalClause logicalPredicate="LE">
3684             <IntClause>45</IntClause>
3685           </RationalClause>
3686         </SimpleClause>
3687       </Clause>
3688       <Clause>
3689         <SimpleClause leftArgument="American">
3690           <BooleanClause booleanPredicate="True"/>
3691         </SimpleClause>
3692       </Clause>
3693     </CompoundClause>
3694   </CompoundClause>
3695 </Clause>
3696 </CompoundClause>
3697 <Clause>
3698

```

3699 **8.3 SQL Query Support**

3700 The Registry may optionally support an SQL based query capability that is designed for Registry
 3701 clients that demand more advanced query capability. The optional SQLQuery element in the
 3702 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query
 3703 language.

3704 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of
 3705 the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database
 3706 Language SQL [SQL], extended to include `<sql invoked routines>` (also known as
 3707 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined
 3708 in template form in Appendix D.3. The syntax of the Registry query language is defined by the
 3709 BNF grammar in D.1.

3710 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use
 3711 relational databases in a Registry implementation.

3712 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

3713 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational
3714 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as
3715 described in the following sections.

3716 **8.3.1.1 Class Binding**

3717 A subset of the class names defined in [ebRIM] map to table names that may be queried by an
3718 SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an
3719 SQL query.

3720 The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3
3721 is as follows:

3722 ?? Classes that have concrete instances are mapped to relational tables. In addition entity classes
3723 (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.

3724 ?? The intermediate classes in the inheritance hierarchy, namely RegistryObject and
3725 RegistryEntry, map to relational views.

3726 ?? The names of relational tables and views are the same as the corresponding [ebRIM] class
3727 name. However, the name binding is case insensitive.

3728 ?? Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in
3729 Appendix D.3 where the column definitions are based on a subset of attributes defined for
3730 that class in [ebRIM]. The attributes that map to columns include the inherited attributes for
3731 the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed
3732 which column definitions.

3733 An SQLQuery against a table not defined in Appendix D.3 may raise an error condition:
3734 InvalidQueryException.

3735 The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn
3736 definitions.

3737 **8.3.1.2 Primitive Attributes Binding**

3738 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same
3739 way as column names in SQL. Again the exact attribute names are defined in the class
3740 definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is
3741 therefore valid for a query to contain attribute names that do not exactly match the case defined
3742 in [ebRIM].

3743 **8.3.1.3 Reference Attribute Binding**

3744 A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a
3745 class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject
3746 class returns a reference to an instance of an AccessControlPolicy object.

3747 In such cases the reference maps to the id attribute for the referenced object. The name of the
3748 resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data
3749 type for the column is VARCHAR(64) as defined in Appendix D.3.

3750 When a reference attribute value holds a null reference, it maps to a null value in the SQL
3751 binding and may be tested with the <null specification> (“IS [NOT] NULL” syntax) as defined

3752 by [SQL].

3753 Reference attribute binding is a special case of a primitive attribute mapping.

3754 **8.3.1.4 Complex Attribute Binding**

3755 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of
3756 a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type
3757 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

3758 The SQL query schema does not map complex attributes as columns in the table for the class for
3759 which the attribute is defined. Instead the complex attributes are mapped to columns in the table
3760 for the domain class that represents the data type for the complex attribute (e.g.

3761 TelephoneNumber). A column links the row in the domain table to the row in the parent table
3762 (e.g. User). An additional column named 'attribute_name' identifies the attribute name in the
3763 parent class, in case there are multiple attributes with the same complex attribute type.

3764 This mapping also easily allows for attributes that are a collection of a complex type. For
3765 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the
3766 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier
3767 and an attribute_name.

3768 **8.3.1.5 Binding of Methods Returning Collections**

3769 Several of the [ebRIM] classes define methods in addition to attributes, where these methods
3770 return collections of references to instances of classes defined by [ebRIM]. For example, the
3771 getPackages method of the RegistryObject class returns a Collection of references to instances of
3772 Packages that the object is a member of.

3773 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in
3774 Appendix D.3 such that these stored procedures return a collection of i.d attribute values. The
3775 returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

3776 These stored procedures may be used as the right-hand-side of an SQL IN clause to test for
3777 membership of an object in such collections of references.

3778 **8.3.2 Semantic Constraints On Query Syntax**

3779 This section defines simplifying constraints on the query syntax that cannot be expressed in the
3780 BNF for the query syntax. These constraints must be applied in the semantic analysis of the
3781 query.

- 3782 1. Class names and attribute names must be processed in a case insensitive manner.
- 3783 2. The syntax used for stored procedure invocation must be consistent with the syntax of an
3784 SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 3785 3. For this version of the specification, the SQL select column list consists of exactly one
3786 column, and must always be t.i.d, where t is a table reference in the FROM clause.
- 3787 4. Join operations must be restricted to simple joins involving only those columns that have an
3788 index defined within the normative SQL schema. This constraint is to prevent queries that
3789 may be computationally too expensive.

3790 8.3.3 SQL Query Results

3791 The result of an SQL query resolves to a collection of objects within the registry. It never
3792 resolves to partial attributes. The objects related to the result set may be returned as an
3793 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the
3794 responseOption parameter specified by the client on the AdHocQueryRequest. The entire result
3795 set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 8.1.

3796 8.3.4 Simple Metadata Based Queries

3797 The simplest form of an SQL query is based upon metadata attributes specified for a single class
3798 within [ebRIM]. This section gives some examples of simple metadata based queries.

3799 For example, to get the collection of ExtrinsicObjects whose name contains the word 'Acme'
3800 and that have a version greater than 1.3, the following query must be submitted:

```
3801 SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND  
3802 eo.id = nm.parent AND  
3803 eo.majorVersion >= 1 AND  
3804 (eo.majorVersion >= 2 OR eo.minorVersion > 3);  
3805  
3806
```

3807 Note that the query syntax allows for conjugation of simpler predicates into more complex
3808 queries as shown in the simple example above.

3809 8.3.5 RegistryObject Queries

3810 The schema for the SQL query defines a special view called RegistryObject that allows doing a
3811 polymorphic query against all RegistryObject instances regardless of their actual concrete type or
3812 table name.

3813 The following example is the similar to that in Section 8.3.4 except that it is applied against all
3814 RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id
3815 for all qualifying RegistryObject instances whose name contains the word 'Acme' and whose
3816 description contains the word "bicycle".

```
3817 SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND  
3818 d.value LIKE '%bicycle%' AND  
3819 ro.id = nm.parent AND ro.id = d.parent;  
3820  
3821
```

3822 8.3.6 RegistryEntry Queries

3823 The schema for the SQL query defines a special view called RegistryEntry that allows doing a
3824 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or
3825 table name.

3826 The following example is the same as Section 8.3.4 except that it is applied against all
3827 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id
3828 for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a
3829 version greater than 1.3.

```
3830 SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND  
3831 re.id = nm.parent AND  
3832 re.majorVersion >= 1 AND  
3833 (re.majorVersion >= 2 OR re.minorVersion > 3);  
3834
```

3835

3836 **8.3.7 Classification Queries**

3837 This section describes the various classification related queries that must be supported.

3838 **8.3.7.1 Identifying ClassificationNodes**

3839 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may
 3840 also be identified as a path attribute that specifies an XPATH expression [XPT] from a root
 3841 classification node to the specified classification node in the XML document that would
 3842 represent the ClassificationNode tree including the said ClassificationNode.

3843 **8.3.7.2 Getting ClassificationSchemes**

3844 To get the collection of ClassificationSchemes the following query predicate must be supported:

```
3845
3846 SELECT scheme.id FROM ClassificationScheme scheme;
3847
```

3848 The above query returns all ClassificationSchemes. Note that the above query may also specify
 3849 additional predicates (e.g. name, description etc.) if desired.

3850 **8.3.7.3 Getting Children of Specified ClassificationNode**

3851 To get the children of a ClassificationNode given the ID of that node the following style of query
 3852 must be supported:

```
3853
3854 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
3855
```

3856 The above query returns all ClassificationNodes that have the node specified by <id> as their
 3857 parent attribute.

3858 **8.3.7.4 Getting Objects Classified By a ClassificationNode**

3859 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the
 3860 following style of query must be supported:

```
3861
3862 SELECT id FROM ExtrinsicObject
3863 WHERE
3864     id IN (SELECT classifiedObject FROM Classification
3865           WHERE
3866             classificationNode IN (SELECT id FROM ClassificationNode
3867                                   WHERE path = '/Geography/Asia/Japan'))
3868 AND
3869     id IN (SELECT classifiedObject FROM Classification
3870           WHERE
3871             classificationNode IN (SELECT id FROM ClassificationNode
3872                                   WHERE path = '/Industry/Automotive'))
3873
```

3874 The above query gets the collection of ExtrinsicObjects that are classified by the Automotive
 3875 Industry and the Japan Geography. Note that according to the semantics defined for
 3876 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by
 3877 descendants of the specified ClassificationNodes.

3878 **8.3.7.5 Getting Classifications That Classify an Object**

3879 To get the collection of Classifications that classify a specified Object the following style of
3880 query must be supported:

```
3881  
3882 SELECT id FROM Classification c  
3883 WHERE c.classifiedObject = <id>;  
3884
```

3885 **8.3.8 Association Queries**

3886 This section describes the various Association related queries that must be supported.

3887 **8.3.8.1 Getting All Association With Specified Object As Its Source**

3888 To get the collection of Associations that have the specified Object as its source, the following
3889 query must be supported:

```
3890  
3891 SELECT id FROM Association WHERE sourceObject = <id>  
3892
```

3893 **8.3.8.2 Getting All Association With Specified Object As Its Target**

3894 To get the collection of Associations that have the specified Object as its target, the following
3895 query must be supported:

```
3896  
3897 SELECT id FROM Association WHERE targetObject = <id>  
3898
```

3899 **8.3.8.3 Getting Associated Objects Based On Association Attributes**

3900 To get the collection of Associations that have specified Association attributes, the following
3901 queries must be supported:

3902 Select Associations that have the specified name.

```
3903  
3904 SELECT id FROM Association WHERE name = <name>  
3905
```

3906 Select Associations that have the specified association type, where association type is a string
3907 containing the corresponding field name described in [eBRIM].

```
3908  
3909 SELECT id FROM Association WHERE  
3910 associationType = <associationType>  
3911
```

3912 **8.3.8.4 Complex Association Queries**

3913 The various forms of Association queries may be combined into complex predicates. The
3914 following query selects Associations that have a specific sourceObject, targetObject and
3915 associationType:

```
3916  
3917 SELECT id FROM Association WHERE  
3918 sourceObject = <id1> AND  
3919 targetObject = <id2> AND  
3920 associationType = <associationType>;  
3921
```

3922 8.3.9 Package Queries

3923 To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
3924
3925 SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));
3926
```

3927 8.3.9.1 Complex Package Queries

3928 The following query gets all Packages that a specified object belongs to, that are not deprecated
3929 and where name contains "RosettaNet."

```
3930
3931 SELECT id FROM Package p, Name n WHERE
3932     p.id IN (RegistryObject_packages(<id>)) AND
3933     nm.value LIKE '%RosettaNet%' AND nm.parent = p.id AND
3934     p.status <> 'Deprecated'
3935
```

3936 8.3.10 ExternalLink Queries

3937 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is
3938 specified:

```
3939
3940 SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
3941
```

3942 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is
3943 specified:

```
3944
3945 SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
3946
```

3947 8.3.10.1 Complex ExternalLink Queries

3948 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that
3949 contain the word 'legal' in their description and have a URL for their externalURI.

```
3950
3951 SELECT id FROM ExternalLink WHERE
3952     id IN (RegistryObject_externalLinks(<id>)) AND
3953     description LIKE '%legal%' AND
3954     externalURI LIKE '%http://%'
3955
```

3956 8.3.11 Audit Trail Queries

3957 To get the complete collection of AuditableEvent objects for a specified RegistryObject, the
3958 following query is specified:

```
3959
3960 SELECT id FROM AuditableEvent WHERE registryObject = <id>
3961
```

3962 8.4 Content Retrieval

3963 A client retrieves content via the Registry by sending the GetContentRequest to the
3964 QueryManager. The GetContentRequest specifies a list of Object references for Objects that
3965 need to be retrieved. The QueryManager returns the specified content by sending a
3966 GetContentResponse message to the RegistryClient interface of the client. If there are no errors
3967 encountered, the GetContentResponse message includes the specified content as additional

3968 payloads within the message. In addition to the GetContentResponse payload, there is one
 3969 additional payload for each content that was requested. If there are errors encountered, the
 3970 RegistryResponse payload includes an error and there are no additional content specific
 3971 payloads.

3972 **8.4.1 Identification Of Content Payloads**

3973 Since the GetContentResponse message may include several repository items as additional
 3974 payloads, it is necessary to have a way to identify each payload in the message. To facilitate this
 3975 identification, the Registry must do the following:

3976 ?? Use the ID of the ExtrinsicObject instance as the value of the Content-ID header parameter
 3977 for the mime multipart that contains the corresponding repository item for the
 3978 ExtrinsicObject.

3979 ?? In case of [ebMS] transport, use the ID of the ExtrinsicObject instance in the Reference
 3980 element for that object in the Manifest element of the ebXMLHeader.

3981 **8.4.2 GetContentResponse Message Structure**

3982 The following message fragment illustrates the structure of the GetContentResponse Message
 3983 that is returning a Collection of Collaboration Protocol Profiles as a result of a
 3984 GetContentRequest that specified the IDs for the requested objects. Note that the boundary
 3985 parameter in the Content-Type headers in the example below are meant to be illustrative not
 3986 prescriptive.

```

3987 Content-type: multipart/related; boundary="MIME_boundary"; type="text/xml";
3988
3989 --MIME_boundary
3990 Content-ID: <GetContentRequest@example.com>
3991 Content-Type: text/xml
3992
3993 <?xml version="1.0" encoding="UTF-8"?>
3994 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
3995   xmlns:eb='http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
3996   <SOAP-ENV:Header>
3997
3998     <!--ebMS header goes here if using ebMS-->
3999     ...
4000
4001     <ds:Signature ...>
4002     <!--signature over soap envelope-->
4003     ...
4004   </ds:Signature>
4005 </SOAP-ENV:Header>
4006
4007 </SOAP-ENV:Header>
4008
4009 <SOAP-ENV:Body>
4010
4011   <!--ebMS manifest goes here if using ebMS-->
4012   ...
4013
4014   <?xml version="1.0" encoding="UTF-8"?>
4015
4016   <GetContentResponse>
4017     <ObjectRefList>
4018       ObjectRef id="urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff"/>
4019       <ObjectRef id="urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f"/>
4020     </ObjectRefList>
4021   </GetContentResponse>
4022
4023 </SOAP-ENV:Body>
4024 </SOAP-ENV:Envelope>

```

```
4025 --MIME_boundary
4026
4027 Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4028 Content-Type: Multipart/Related; boundary=payload1_boundary; type=text/xml
4029 Content-Description: Optionally describe payload1 here
4030
4031 --payload1_boundary
4032 Content-Type: text/xml; charset=UTF-8
4033 Content-ID: signature:urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4034
4035 <ds:Signature ...>
4036   ... Signature for payload1
4037 </ds:Signature>
4038
4039 --payload1_boundary
4040 Content-ID: urn:uuid:d8163dfb-f45a-4798-81d9-88aca29c24ff
4041 Content-Type: text/xml
4042
4043 <?xml version="1.0" encoding="UTF-8"?>
4044 <tp:CollaborationProtocolProfile ...>
4045   .....
4046 </tp:CollaborationProtocolProfile>
4047 --payload1_boundary--
4048
4049 --MIME_boundary
4050
4051 Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
4052 Content-Type: Multipart/Related; boundary=payload2_boundary; type=text/xml
4053 Content-Description: Optionally describe payload2 here
4054
4055 --payload2_boundary
4056 Content-Type: text/xml; charset=UTF-8
4057 Content-ID: signature:urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
4058
4059 <ds:Signature ...>
4060   ... Signature for payload2
4061 </ds:Signature>
4062 --payload2_boundary
4063 Content-ID: urn:uuid:212c3a78-1368-45d7-acc9-a935197e1e4f
4064 Content-Type: text/xml
4065
4066 <?xml version="1.0" encoding="UTF-8"?>
4067 <tp:CollaborationProtocolProfile ...>
4068   .....
4069 </tp:CollaborationProtocolProfile>
4070 --payload2_boundary--
4071
4072 --MIME_boundary--
4073
4074
4075
4076
```


4077 **9 Content-based Discovery**

4078 This chapter describes the Content-based discovery facility of the ebXML Registry. This facility
4079 enables clients to discover repository items based upon the content contained within the
4080 repository item. The Content-based discovery facility is a required normative feature of ebXML
4081 Registries compliant to version 3 or later of this specification.

4082 *The essence of the content-based discovery feature is based upon the ability to selectively*
4083 *convert repository item content into metadata consisting of instances of RegistryObject sub-*
4084 *classes (RegistryObject Metadata).*

4085 A registry uses one or more content indexing services to automatically index repository items
4086 when they are submitted to the registry. Indexing a repository item creates RegistryObject
4087 metadata such as Classification instances. The indexed metadata enables clients to discover the
4088 repository item using existing query capabilities of the registry.

4089 *The term index is used to refer to RegistryObject Metadata generated from selective repository*
4090 *item content. It should not be confused with databases indexes. It is named such because it is*
4091 *similar in concept to database indexes, which are metadata generated from content.*

4092 **9.1 Content-based Discovery: Use Cases**

4093 There are many scenarios where content-based discovery is necessary.

4094 **9.1.1 Find All CPPs Where Role is “Buyer”**

4095 A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to
4096 find CPPs for other companies where the Role element of the CPP is that of “Buyer”.

4097 **9.1.2 Find All XML Schema’s That Use Specified Namespace**

4098 A client may wish to discover all XML Schema documents in the registry that use an XML
4099 namespace containing the word “oasis”.

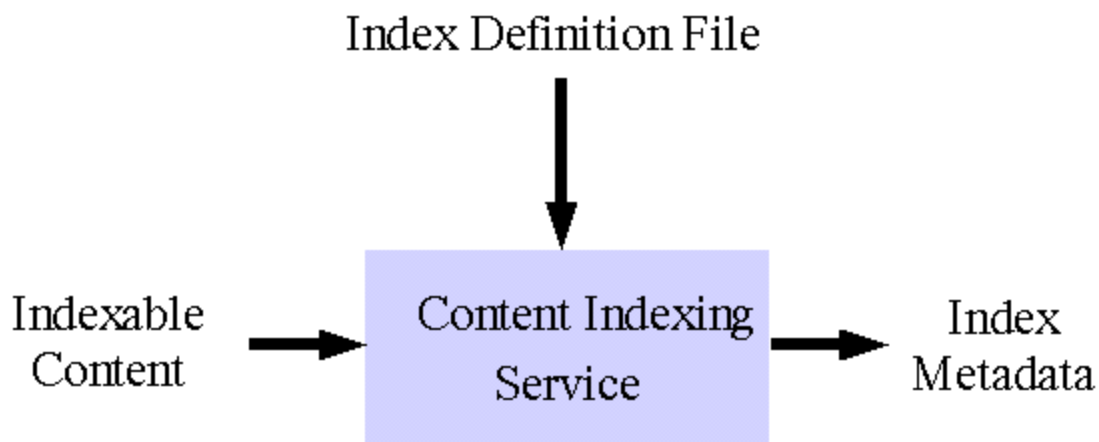
4100 **9.1.3 Find All WSDL Descriptions with a SOAP Binding**

4101 An ebXML registry client is attempting to discover all repository items that are WSDL
4102 descriptions that have a SOAP binding defined. Note that SOAP binding related information is
4103 content within the WSDL document and not metadata

4104 **9.2 Content Indexing Service**

4105 Figure 38 shows that conceptually, a content indexing service (or indexer) accepts as input a
4106 repository item and generates as output one or more *RegistryObject Metadata* instances that are
4107 used to catalog the *ExtrinsicObject* for that repository item. In addition an indexer accepts as
4108 control input an index definition file, which is also a repository item.

4109



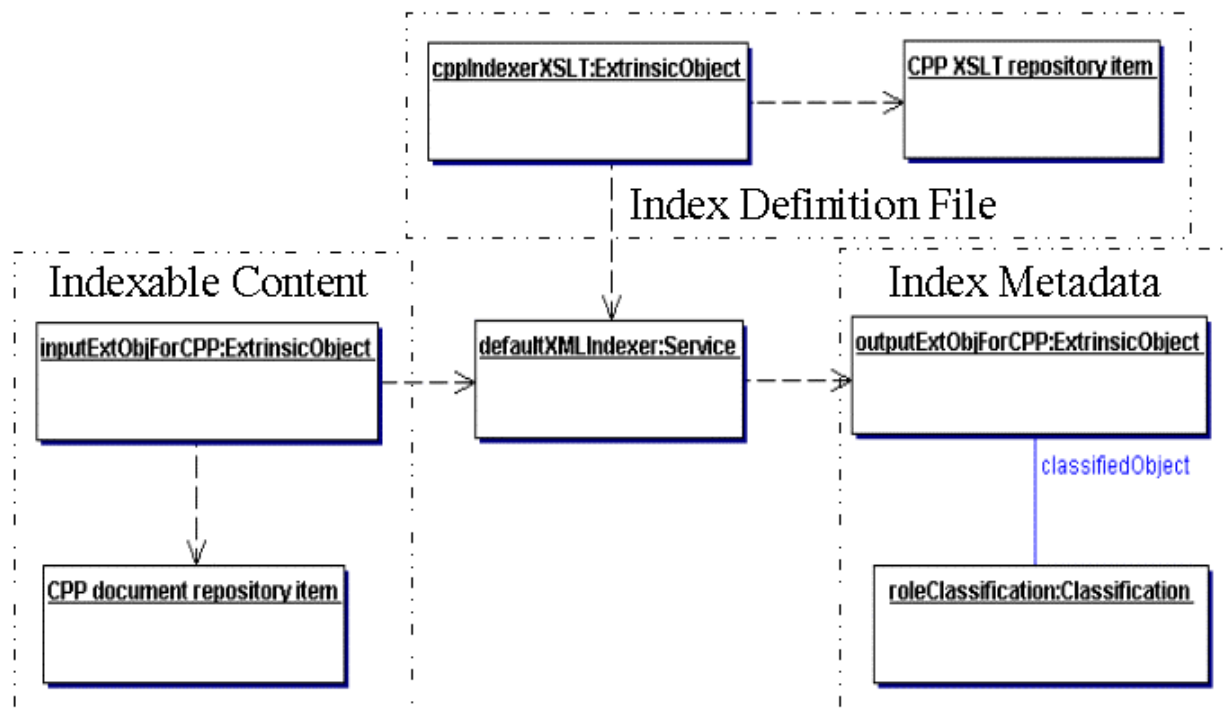
4110
4111

Figure 38: Abstract Content Indexing Service: Inputs and Outputs

4112 9.2.1 Illustrative Example

4113 Figure 39 shows a UML instance diagram to illustrate how a Content Indexing Service is used.
4114 The content indexing service is the normative Default XML Indexing Service described in
4115 section 9.10.

- 4116 ○ In the center we see a Content Indexing Service name defaultXMLIndexer.
- 4117 ○ On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP
4118 being input as Indexable Content to the defaultXMLIndexer.
- 4119 ○ On top we see an XSLT style sheet repository item and its ExtrinsicObject being sent as
4120 an Index Definition File to the defaultXMLIndexer.
- 4121 ○ On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for
4122 the CPP. We also see a Classification roleClassification, which classifies the CPP by the
4123 Role element within the CPP. These are the Index Metadata generated as a result of the
4124 indexer indexing the CPP.



4125
4126

Figure 39: Example of CPP indexing using Default XML Indexer

4127 **9.3 Index Definition File**

4128 The Index Definition File describes the information that the indexer must extract from the
 4129 repository item and subsequently map it to the generated *RegistryObject Metadata*. This
 4130 specification does not define the format of the Index Definition File. Each indexer is free to
 4131 define its own Index Definition File format in an indexer specific manner. The only constraint in
 4132 this specification is that the index definition file must be a repository item.

4133 **9.4 Indexable Content**

4134 The indexable content is the content that the client wishes to be indexed by the Content Indexing
 4135 Service. As such it is the subject of the content indexing action.

4136 This specification does not define the format of indexable content. This specification describes
 4137 how a client may register arbitrary indexers for indexing arbitrary content types.

4138 The most common use case for an indexer is for indexing XML documents. Therefore, this
 4139 specification also provides a normative definition for a specialized XML Content Indexer in
 4140 section 9.10.

4141 *An ebXML Registry must provide native built-in support for the normative default XML Content
 4142 Indexer.*

4143 In addition, an ebXML Registry may optionally allow clients to register arbitrary indexers for
 4144 arbitrary content. In either case the registry must use the appropriate indexer if one exists, to
 4145 index a repository item when it is submitted.

4146 9.5 Index Metadata

4147 A content indexing service indexes a repository item by processing it and extracting specific
 4148 information content as specified by the Index Definition File. The content indexing service must
 4149 map the extracted content to index metadata in form of instances of RIM classes.

4150 For example, the index metadata may consists of:

- 4151 o Classification instances
- 4152 o ExternalIdentifier instances
- 4153 o ExternalLink instances
- 4154 o The name attribute for the ExtrinsicObject for the indexable content
- 4155 o The description attribute for the ExtrinsicObject for the able-able content

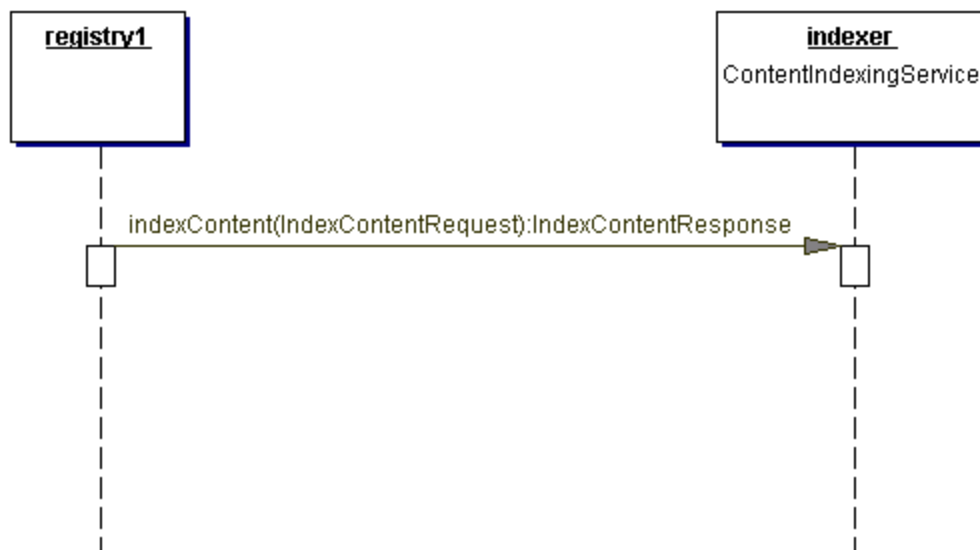
4156 A content indexing service is free to generate any class defined by RIM as index metadata in an
 4157 application specific manner.

4158 9.6 Content Indexing Protocol

4159 The interface of the content indexing service must implement a single method called
 4160 `indexContent`. The `indexContent` method accepts an `IndexContentRequest` as parameter and
 4161 returns an `IndexContentResponse` as its response if there are no errors.

4162 The `IndexContentRequest` contains repository items that need to be indexed. The resulting
 4163 `IndexContentResponse` contains the metadata that gets generated by the Content Indexing
 4164 Service as a result of indexing the specified repository items.

4165 The content indexing protocol is abstract and does not specify the implementation details of any
 4166 specific Content Indexing Service.

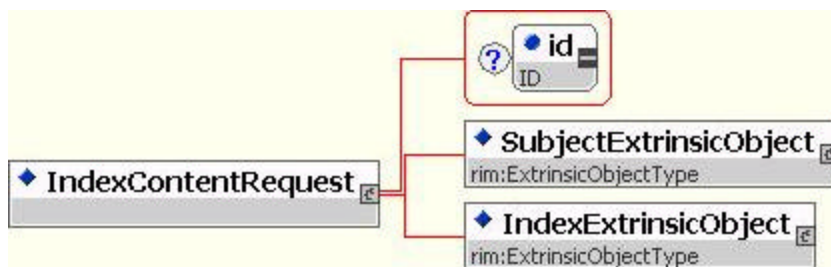


4167
 4168

Figure 40: Content Indexing Protocol

4169 9.6.1 IndexContentRequest

4170 The `IndexContentRequest` is used to submit repository items to a Content Indexing Service so
 4171 that it can create index metadata for the specified repository items.

4172 **9.6.1.1 Syntax:**4173 **Figure 41: IndexContentRequest Syntax**
41744175 **9.6.1.2 Parameters:**4176 *id*: Inherited request id attribute common to all requests.

4177 *IndexExtrinsicObject*: This parameter specifies the ExtrinsicObject for the
 4178 repository item that the caller wishes to specify as the Index Definition file. This
 4179 specification does not specify the format of this repository item. There must a
 4180 corresponding repository item as an attachment to this request. The corresponding
 4181 repository item should follow the same rules as attachments in
 4182 SubmitObjectsRequest.

4183 *SubjectExtrinsicObject*: This parameter specifies the ExtrinsicObject for the
 4184 repository item that the caller wishes to be indexed. This specification does not
 4185 specify the format of this repository item. There must a corresponding repository
 4186 item as an attachment to this request. The corresponding repository item should
 4187 follow the same rules as attachments in SubmitObjectsRequest.

4188

4189 **9.6.1.3 Returns:**

4190 This request returns an IndexContentResponse. See section 9.6.2 for details.

4191 **9.6.1.4 Exceptions:**

4192 In addition to the exceptions common to all requests, the following exceptions may be returned:

4193 *MissingRepositoryItemException*: signifies that the caller did not provide a
 4194 required repository item as an attachment to this request.

4195 *UnsupportedIndexException*: signifies that this Content Indexing Service did not
 4196 support the IndexExtrinsicObject provided by the client.

4197 *UnsupportedSubjectException*: signifies that this Content Indexing Service did
 4198 not support the SubjectExtrinsicObject provided by the client.

4199

4200 **9.6.2 IndexContentResponse**

4201 The IndexContentRequest is sent by the Content Indexing Service as a response to an
 4202 IndexContentRequest.

4203

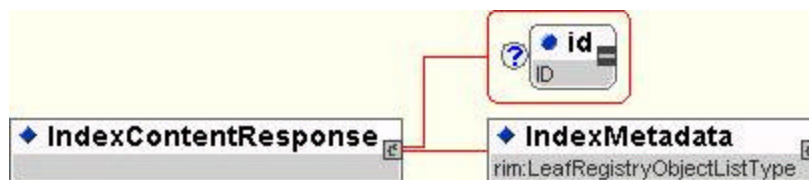
4204 **9.6.2.1 Syntax:**4205
4206

Figure 42: IndexContentResponse Syntax

4207 **9.6.2.2 Parameters:**

- 4208 *id*: id attribute inherited from RegistryResponseType.
- 4209 *IndexMetadata*: This parameter specifies a collection of RegistryObject instances
 4210 that were created as a result of dynamic content indexing by a content indexing
 4211 service. It may include a modified ExtrinsicObject for the repository item that has
 4212 been indexed by the Content Indexing Service. The Content Indexing Service may
 4213 add metadata such as Classifications, ExternalIdentifiers, name, description etc. to
 4214 the IndexedExtrinsicObject element. There must not be an accompanying
 4215 repository item as an attachment to this message.
- 4216

4217 **9.7 Publishing a Content Indexing Service**

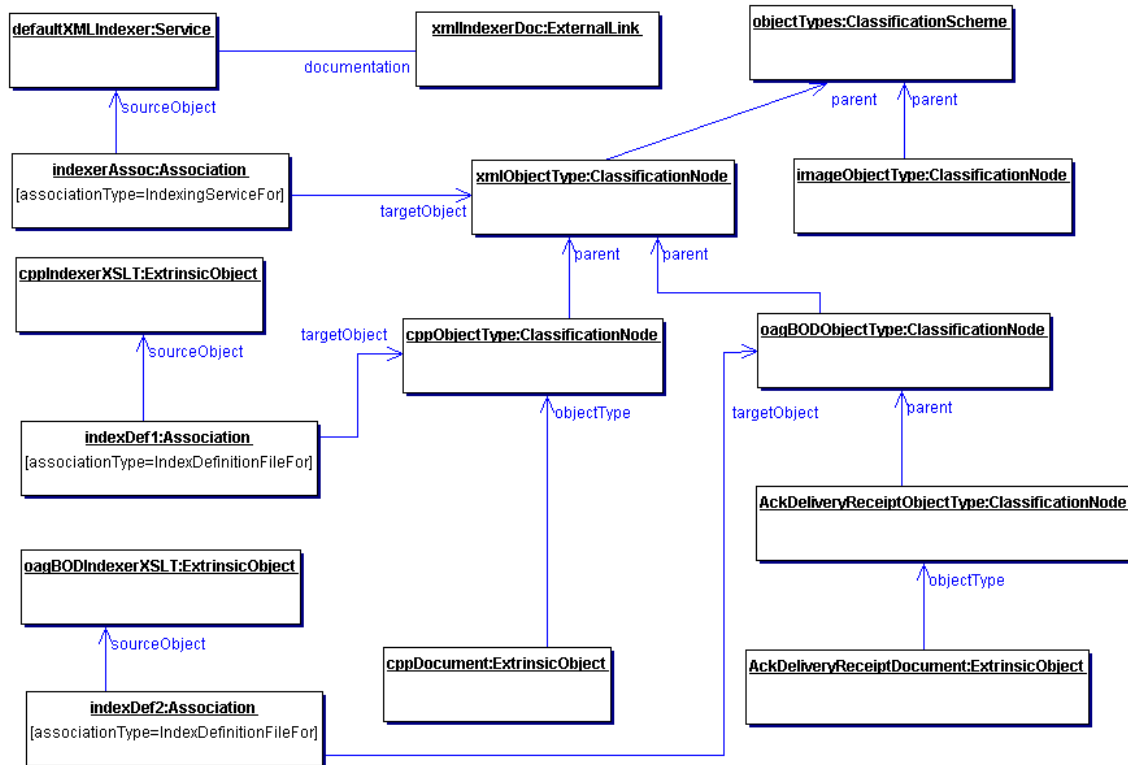
4218 Any publisher may publish an arbitrary content indexing service to an ebXML Registry. The
 4219 content indexing service must be published using the existing LifeCycleManager interface.

4220 The publisher must use the existing SubmitObjectsRequest to publish:

- 4221 ○ A Service instance that must have a required Association with associationType of
 4222 “IndexingServiceFor”. In Figure 43 this is exemplified by the defaultXMLIndexerService
 4223 in the upper-left corner. The Service must be the sourceObject while a
 4224 ClassificationNode in the canonical ObjectType ClassificationScheme must be the
 4225 targetObject.
- 4226 ○ A ServiceBinding instance contained within the Service instance that must provide the
 4227 accessURI to the indexing Service.
- 4228 ○ An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page
 4229 describing:
 - 4230 ○ The format of the supported Indexable Content
 - 4231 ○ The format of the supported Index Definition File

4232 Note that no SpecificationLink is required since this specification is implicit for Content
 4233 Indexing Services.

- 4234 ○ One or more index definition file(s) that must be an ExtrinsicObject and repository item
 4235 pair. The ExtrinsicObject for the index definition must have a required Association with
 4236 associationType of “IndexDefinitionFileFor”. In Figure 43 this is exemplified by the
 4237 cppIndexerXSLT and the oagBODIndexerXSLT objects on the left side. The Service
 4238 must be the sourceObject while a ClassificationNode in the canonical ObjectType
 4239 ClassificationScheme must be the targetObject.
- 4240 ○ Zero or more ClassificationScheme(s) and ClassificationNodes(s) that may be
 4241 referenced (used) in the indexed metadata generated by the content indexing Service.



4242
4243

Figure 43: Indexing Service Configuration

4244 Figure 43 shows the configuration of the default XML indexer which is associated with the
 4245 objectType for XML content. Thus this indexer may be used with any XML content that has its
 4246 objectType attribute reference the xmlObjectType ClassificationNode or one of its descendents.
 4247 The figure also shows two different Index Definition Files, cppIndexerXSLT and
 4248 oagBODIndexerXSLT that may be used to index ebXML CPP and OAG Business Object
 4249 Documents (BOD) respectively.
 4250

4251 9.7.1 Multiple Indexers and Index Definition Files

4252 Cleanup verbage here??

4253 This specification allows clients to submit multiple indexers and index definition files for the
 4254 same objectType. How a registry handles multiple indexer and index definition file submission
 4255 for the same type of content is a matter of registry specific policy. If a registry does not allow
 4256 this then it must send an InvalidRequestException with a reason, when a duplicate indexer or
 4257 index def is submitted. If a registry allows this then it must provide a conflict resolution
 4258 mechanism to select the appropriate indexer and index definition file in some registry specific
 4259 manner.

4260 **9.7.2 Restrictions On Publishing Content Indexing Services**

4261 A client may submit any content indexing service or index definition file. A registry may use
4262 registry specific policies to determine whether a client submitted content indexing service or
4263 index definition file are acceptable. For example a registry may require that the content indexing
4264 service or index definition file does not create excessive metadata. A registry may reject a
4265 SubmitObjectRequest with an InvalidRequestException and give a reason why the request was
4266 rejected, upon receiving requests publishing Content Indexing Service or Index Definition File
4267 that is unreasonable. In effect support for user-defined content indexing services is optional in
4268 this version of the specification.

4269 **9.8 Dynamic Content Indexing**

4270 Some time during or after a publisher submits a repository item, the registry must check to see if
4271 there is a Content Indexing Service and index definition file registered for that type of repository
4272 item. This is referred to as Content Indexing Service resolution and index definition file
4273 resolution as described in section described in section 9.8.3.

4274 If a Content Indexing Service and index definition file are found then the registry must invoke
4275 that service using the Content Indexing Protocol. In the invocation, it gives a repository item as
4276 Indexable Content and a repository item as Index Definition File within an
4277 IndexContentRequest. The Content Indexing Service must index the content and return the
4278 modified ExtrinsicObject for the Indexable Content such that it has index metadata generated
4279 from relevant portions of the Indexable Content.

4280 The registry must store the repository item along with the modified ExtrinsicObject annotated
4281 with the index metadata once the Content Indexing Protocol is completed.

4282 The result of dynamic content indexing is that indexable content gets indexed automatically as a
4283 consequence of being submitted. Once indexed it is possible to use the index metadata to do
4284 dynamic content-based discovery of the indexable content.

4285 **9.8.1 Threading Model for Dynamic Content Indexing**

4286 A registry may do dynamic content indexing synchronous with the original
4287 SubmitObjectRequest request or it may do so asynchronously sometime after the request is
4288 committed.

4289 **9.8.2 Referential Integrity and Dynamic Content Indexing**

4290 A registry must maintain referential integrity between the RegistryObjects and repository items
4291 in the submission and the generated RegistryObjects in the indexed metadata.

4292 **9.8.3 Error Handling Model for Dynamic Content Indexing**

4293 Any errors generated during dynamic content indexing must not effect the storing of the
4294 RegistryObjects and repository items that were submitted. Such indexing errors are internal
4295 registry errors due to implementation errors or configuration errors.

4296 A registry must return a normal response with status = "Success" if the submitted content and
4297 metadata is stored successfully even when there are errors encountered during dynamic content
4298 indexing. A registry should log such indexing errors like any other internal registry errors so that
4299 a Registry Operator may be able to explore the problem at a later time.

4300 **9.8.4 Updates and Dynamic Content Indexing**

4301 When an ExtrinsicObject and its repository item are updated within a registry, the registry must
4302 remove any previously created indexed metadata and regenerate the index metadata.

4303 **9.8.5 Resolution Algorithm For Indexer and Index Definition File**

4304 When a registry receives a submission of an ExtrinsicObject EO1 and repository item pair, it
4305 must use the following algorithm to determine or resolve the content indexing service and index
4306 definition file to be used to index that content:

- 4307 1. Get the objectType attribute of the ExtrinsicObject. If the objectType is a UUID to a
4308 classificationNode (referred to as objectType ClassificationNode) then proceed to next
4309 step. **Need to update objectType in ebRIM to say it must be a ref to a node in**
4310 **ObjectType. For backward compatibility allow non-UUID values??**
- 4311 2. Query to see if the objectType ClassificationNode is the targetObject of an Association of
4312 type "IndexingServiceFor". If not then repeat this step with the parent ClassificationNode
4313 of this ClassificationNode. Repeat until the parent is the ClassificationScheme or until the
4314 desired Association is found. If desired Association is found proceed to next step.
- 4315 3. Check if the sourceObject of the desired Association is a Service instance. If not throw an
4316 InvalidRequestException. If it is a Service instance, then use this Service as the content
4317 indexing service for the ExtrinsicObject.
- 4318 4. Query to see if the objectType ClassificationNode is the targetObject of an Association of
4319 type "IndexDefinitionFileFor". If not then repeat this step with the parent
4320 ClassificationNode of this ClassificationNode. Repeat until the parent is the
4321 ClassificationScheme or until the desired Association is found.
- 4322 5. Check if the sourceObject of the desired Association is an ExtrinsicObject instance. If not
4323 throw an InvalidRequestException. If it is a ExtrinsicObject instance, then use this
4324 ExtrinsicObject and its repository item as the index definition file.

4325 The above algorithm allows for objectType hierarchy to be used to configure indexer and index
4326 definition files with varying degrees of specificity or specialization with respect to the type of
4327 content.

4328 If no indexer or index definition file is found then content should not be indexed.

4329 **9.9 Dynamic Content-based Discovery**

4330 As described earlier, indexable content is automatically indexed when it is submitted to the
4331 registry. This content may subsequently be dynamically discovered using the index metadata
4332 within existing AdhocQueryRequest. Because the index metadata is based upon indexable
4333 content, an AdhocQueryRequest can perform dynamic content- based discovery.

4334 **9.10 Default XML Content Indexer**

4335 An ebXML Registry must provide the XML Content Indexing Service natively as a built-in
4336 service. The XML content indexing service accepts an XML instance document as its input and
4337 it accepts an XSLT Style sheet as a Content Definition File. Each type of content should have its
4338 own unique XSLT style sheet. For example and ebXML CPP document should have a specialize
4339 ebXML CPP index definition style sheet. The XML content indexing service must apply the
4340 XSLT style sheet to the XML instance document input to generate the index metadata. Since a
4341 single style sheet must be applied to both the ExtrinsicObject and the Indexable Content, we
4342 must assume the two documents to be composed within a single virtual document the schema for
4343 which is as follows:

```
4344  
4345 <cbd:MetaDataAndContent>  
4346     <rim:ExtrinsicObject/>  
4347     <someXMLTag/>  
4348 </cbd:MetaDataAndContent>
```

4349 **9.10.1 Publishing of Default XML Content Indexer**

4350 The default XML Content Indexing Service need not be explicitly published to an ebXML
4351 Registry. An ebXML Registry must provide the XML Content Indexing Service natively as a
4352 built-in service. This built-in service must be published to the registry as part of the intrinsic
4353 bootstrapping of required data within the registry.

4354 **9.11 Canonical Index Definition Files**

4355 It is desirable to have identical index definition files and indexing services for a given object type
4356 across all registry implementations. This provides a consistent behavior for dynamic content
4357 indexing across registries. To facilitate consistency a non-normative set of canonical index
4358 definition files will be maintained at:

4359 <http://www.oasis-open.org/committees/regrep/documents/3.0/contentBasedDiscovery>

4360 **10 Event Notification**

4361 This chapter defines the Event Notification feature of the OASIS ebXML Registry. The Event
4362 Notification feature is a normative optional feature of the ebXML Registry.
4363 Event Notification feature allows OASIS ebXML Registries to notify its users and / or other
4364 registries about events of interest. It allows users to stay informed about registry events without
4365 being forced to periodically poll the registry. It also allows a registry to propagate internal
4366 changes to other registries whose content might be affected by those changes.
4367 ebXML registries support content-based Notification where interested parties express their
4368 interest in form of a query. This is different from subject (sometimes referred to as topic) – based
4369 Notification where information is categorized by subjects and interested parties express their
4370 interests in those predefined subjects.

4371 **10.1 Use Cases**

4372 The following use cases illustrate different ways in which ebXML registries notify users or other
4373 registries.

4374 **10.1.1 New Service is Offered**

4375 A user wishes to know when a new Plumbing service is offered in her town. When that happens,
4376 she might try to learn more about that service and compare it with her current Plumbing service
4377 provider's offering.

4378 **10.1.2 Monitor Download of Content**

4379 User wishes to know whenever her CPP [ebCPP] is downloaded in order to evaluate on an
4380 ongoing basis the success of her recent advertising campaign. She might also want to analyze
4381 who the interested parties are.

4382 **10.1.3 Monitor Price Changes**

4383 User wishes to know when the price of a product that she is interested to buy drops below a
4384 certain amount. If she buys it she would also like to be notified when the product has been
4385 shipped to her.

4386 **10.1.4 Keep Replicas Consistent With Source Object**

4387 In order to improve performance and availability of accessing some registry objects, a local
4388 registry may make replicas of certain objects that are hosted by another registry. The registry
4389 would like to be notified when the source object for a replica is updated so that it can
4390 synchronize the replica with the latest state of the source object.

4391 **10.2 Registry Events**

4392 Activities within a registry result in meaningful events. Typically, registry events are generated
4393 when a registry processes client requests. In addition, certain registry events may be caused by
4394 administrative actions performed by a registry operator. [ebRIM] defines the AuditableEvent
4395 class, instances of which represent registry events. An AuditableEvent is generated by the
4396 registry in response to a registry event.

4397 **10.3 Subscribing to Events**

4398 A User may create a subscription with a registry if she wishes to receive notification for a
4399 specific type of event. A User creates a subscription by submitting a Subscription instance to a
4400 registry using the SubmitObjectsRequest. If a Subscription is submitted to a registry that does
4401 not support event notification then the registry must return an UnsupportedOperationException.

4402 **10.3.1 Event Selection**

4403 In order for a User to only be notified of specific events of interest, she must specify a Selector
4404 within the Subscription instance. A Selector contains a query that determines whether an event
4405 qualifies for that Subscription or not. The query syntax is the normal ad hoc query syntax
4406 describes in chapter 8.

4407 **10.3.2 Notification Action**

4408 When creating a Subscription, a User may also specify what the registry should do when an
4409 event matching the Selector for that Subscription (Subscription's event) transpires. A User may
4410 specify Actions within the Subscription. Each Action defines an action that the registry must
4411 undertake when a Subscription's event transpires. If no Actions are defined within the
4412 Subscription that implies that the user does not wish to be notified asynchronously by the
4413 registry and instead intends to periodically poll the registry and pull the pending Notifications.

4414 [ebRIM] defines two standard Actions that allow delivery of event notifications via email to a
4415 human user or by invocation web service based programmatic interface.

4416 For each event that transpires in the registry, if the registry supports event notification, it must
4417 check all registered and active Subscriptions and see if any Subscriptions match the event. If a
4418 match is found then the registry must perform all the Action's described by the Subscription.
4419 *Performing the Actions for a Subscription by a registry does the actual delivery of events.*

4420 **10.3.3 Subscription Authorization**

4421 A registry may use registry specific policies to decide which User is authorized to create a
4422 subscription. A Registry must return an AuthorizationException in the event that an
4423 Unauthorized User submits a Subscription to a registry.

4424 **10.3.4 Subscription Quotas**

4425 A registry may use registry specific policies to decide an upper limit on the number of
4426 Subscriptions a User is allowed to create. A Registry must return a QuotaExceededException in
4427 the event that an Authorized User submits more Subscriptions than allowed by their registry
4428 specific quota.

4429 **10.3.5 Subscription Expiration**

4430 Each subscription defines a startDate and an endDate attribute which determines the period
4431 within which a Subscription is active. Outside the bounds of the active period, a Subscription may
4432 exist in an inactive state within the registry. A Registry must not consider inactive Subscriptions
4433 when delivering notifications for an event to its Subscriptions.

4434 **10.4 Unsubscribing from Events**

4435 A User may terminate a Subscription with a registry if she no longer wishes to be notified of
 4436 events related to that Subscription. A User terminates a Subscription by deleting the
 4437 corresponding Subscription object using the RemoveObjectsRequest to the registry.

4438 A registry itself may remove a Subscription instance after it has expired. In such cases the
 4439 identity of a RegistryOperator User must be used for the request in order to have sufficient
 4440 authorization to remove a User’s Subscription.

4441 Removal of a Subscription object follows the same rules as removal of any other object.

4442 **10.5 Notification of Events**

4443 A registry performs the Actions for a Subscription in order to actually deliver the events.
 4444 However, regardless of the specific delivery action, the registry must communicate the
 4445 Subscription’s events. The Subscription’s events are delivered within a Notification instance as
 4446 described by [ebRIM].

4447 [ebRIM] defines an extensible description of Notifications, making it possible to allow for
 4448 registry or application specific Notifications. It defines several normative types of Notifications

4449 A client may specify the type of Notification they wish to receive using the notificationOption
 4450 attribute of the Action within the Subscription they register with a registry as a hint to the
 4451 registry. The registry may override this hint based upon registry specific operational policies.

4452 **10.6 Retrieval of Events**

4453 The registry provides asynchronous *PUSH style* delivery of Notifications via notify Actions as
 4454 described earlier. However, a client may also use a *PULL style* to retrieve any pending events for
 4455 their Subscriptions. Pulling of events is done using the GetNotificationsRequest protocol as
 4456 described next.

4457 **10.6.1 GetNotificationsRequest**

4458 The GetNotificationsRequest is used by a client to pull retrieve any pending events for their
 4459 Subscriptions.

4460 **10.6.1.1 Syntax:**

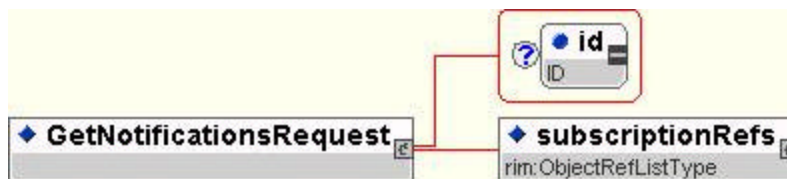


Figure 44: GetNotificationsRequest Syntax

4463 **10.6.1.2 Parameters:**

4464 *subscriptionRefs*: This parameter specifies a collection of ObjectRefs to
 4465 Subscription objects for which the client wishes to get Notifications.

4466

4467

4468

4469 **10.6.1.3 Returns:**

4470 This request returns a GetNotificationsResponse. See section 10.6.2 for details.

4471 **10.6.1.4 Exceptions:**

4472 In addition to the exceptions common to all requests, the following exceptions may be returned:

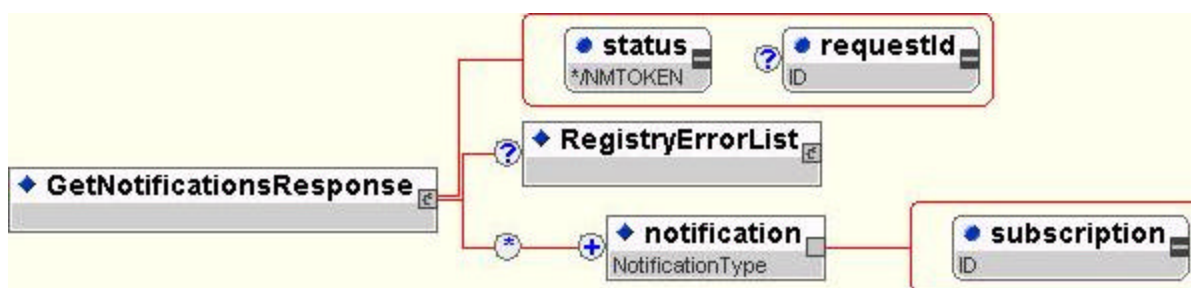
4473 *ObjectNotFoundException*: signifies that a specified Subscription was not found
 4474 in the registry.

4475 **10.6.2 GetNotificationsResponse**

4476 The GetNotificationsResponse is sent by the registry as a response to GetNotificationsRequest. It
 4477 contains the Notifications for the Subscription specified in the GetNotificationsRequest.

4478

4479 **10.6.2.1 Syntax:**



4480
 4481

Figure 45: GetNotificationsResponse Syntax

4482 **10.6.2.2 Parameters:**

4483 *notification*: This parameter specifies a notification contained within the
 4484 GetNotificationsResponse. The notification is actually delivered as a more
 4485 specialized sub-type (e.g. EventRefsNotification) as defined by [ebRIM].

4486
 4487

4488 **10.7 Event Management Policies**

4489 **This section needs to be assimilated into other sections similar to Quota section etc.??.**

4490 There might be several registry specific characteristics that might differ between different
 4491 registries. These characteristics could be seen as some kind of configuration / policy parameters.
 4492 It seems that natural home for those parameters would be Registry class, and in case of
 4493 cooperation registries Federation class (for Registry and Federation definitions see Cooperating
 4494 Registries proposal). Registry parameters that are related to Event Notification might define:

4495 ?? What is audited and whether dynamic Auditing (don't audit everything all the time) is
 4496 supported or not.

- 4497 ?? Whether only Registered actors (use the terminology from security sections) can
4498 subscribe. This might not be a policy, but a spec requirement.
- 4499 ?? What action types are supported.
- 4500 ?? Who can be notified of what (security concerns that some events has to be kept private).
- 4501 ?? When to purge Subscriptions and related Notifications / Events.
- 4502 ?? How long is the retention period in which Events and / or Notifications for undelivered
4503 pending notifications won't be purged?
- 4504 As this functionality is beyond the scope of this proposal, it is assumed that there will be an
4505 effort to design policy management in which case Event Notification policy parameters would
4506 follow that design.

4507 **10.8 Notes**

- 4508 These notes are here to not loose the thought and will be merged into the proposal later.
- 4509 - What to do with user defined subject (topic) – based notifications?
 - 4510 - Do we need discovery of supported Events / Notifications (what event type, source, ...)?
 - 4511 - Filter Query changes to support new objects like Subscriptions.
 - 4512 - Security issues: How to control what notifications others receive about my content or
4513 events.
 - 4514 - How about a style sheet on EmailNotification to make notification be human readable?
 - 4515 - Terminology issue: Action Vs. Notification is confusing. Can we find another name for
4516 NotificationType in xsd.
- 4517
- 4518

4519 11 Cooperating Registries Support

4520 This chapter describes the capabilities and protocols that enable multiple ebXML registries to
4521 cooperate with each other to meet advanced use cases as described next.

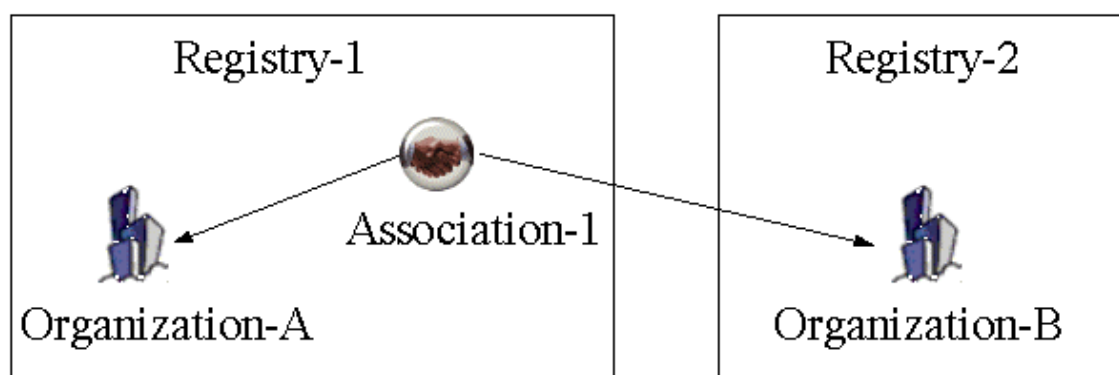
4522 11.1 Cooperating Registries Use Cases

4523 The following is a list of use cases that illustrate different ways that ebXML registries cooperate
4524 with each other.

4525 11.1.1 Inter-registry Object References

4526 A Submitting Organization wishes to submit a RegistryObject to a registry such that the
4527 submitted object references a RegistryObject in another registry.

4528 An example might be where a RegistryObject in one registry is associated with a RegistryObject
4529 in another registry.



4530
4531

☞ Figure 46: Inter-registry Object References

4532

4533 11.1.2 Federated Queries

4534 A client wishes to issue a single query against multiple registries and get back a single response
4535 that contains results based on all the data contained in all the registries. From the client's
4536 perspective it is issuing its query against a single logical registry that has the union of all data
4537 within all the physical registries.

4538 11.1.3 Local Caching of Data from Another Registry

4539 A destination registry wishes to cache some or all the data of another source registry that is
4540 willing to share its data. The shared dataset is copied from the source registry to the destination
4541 registry and is visible to queries on the destination registry even when the source registry is not
4542 available.

4543 Local caching of data may be desirable in order to improve performance and availability of
4544 accessing that object.

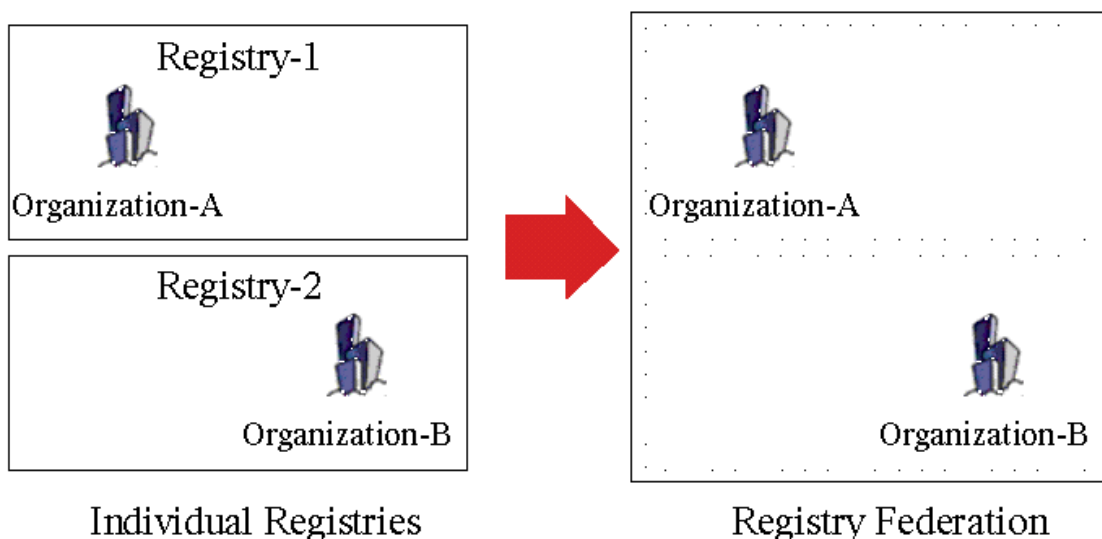
4545 An example might be where a RegistryObject in one registry is associated with a RegistryObject
4546 in another registry, and the first registry caches the second RegistryObject locally.

4547 **11.1.4 Object Relocation**

4548 A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from
 4549 the registry where it was submitted to another registry.

4550 **11.2 Registry Federations**

4551 A registry federation is a group of registries that have voluntarily agreed to form a loosely
 4552 coupled union. Such a federation may be based on common business interests and specialties that
 4553 the registries may share. Registry federations appear as a single logical registry, to registry
 4554 clients.



4555
 4556

☞☞ **Figure 47: Registry Federations**

4557 Registry federations are based on a peer-to-peer (P2P) model where all participating registries
 4558 are equal. Each participating registry is called a *registry peer*. There is no distinction between the
 4559 registry operator that created a federation and those registry operators that joined that Federation
 4560 later.

4561 Any registry operator may form a registry federation at any time. When a federation is created it
 4562 must have exactly one registry peer which is the registry operated by the registry operator that
 4563 created the federation.

4564 Any registry may choose to voluntarily join or leave a federation at any time.

4565 **11.2.1 Federation Metadata**

4566 The Registry Information model defines the Registry and Federation classes, instances of these
 4567 classes and the associations between these instances describe a federation and its members. Such
 4568 instance data is referred to as `Federation Metadata`. The Registry and Federation classes
 4569 are described in detail in [ebRIM].

4570 They Federation information model is summarized here as follows:

- 4571 ○ A Federation instance represents a registry federation.
- 4572 ○ A Registry instance represents a registry

- 4573 ○ An Association instance with associationType “HasFederationMember” between a
4574 Federation instance and a Registry instance represents that the registry is a member of the
4575 federation.
4576

4577 **11.2.2 Local Vs. Federated Queries**

4578 A federation appears to registry clients as a single unified logical registry. An
4579 AdhocQueryRequest sent by a client to a federation member may be local or federated. A new
4580 boolean attribute named `federated` is added to AdhocQueryRequest to indicate whether the
4581 query is federated or not.

4582 **11.2.2.1 Local Queries**

4583 When the federated attribute of AdhocQueryRequest has the value of `false` then the query is a
4584 local query. In the absence of a federated attribute the default value of federated attribute is false.
4585 A local AdhocQueryRequest is only processed by the registry that receives the request. A local
4586 AdhocQueryRequest does not operate on data that belongs to other registries.

4587 **11.2.2.2 Federated Queries**

4588 When the federated attribute of AdhocQueryRequest has the value of `true` then the query is a
4589 federated query.

4590 A federated query to any federation member, must be routed by that member to all other
4591 federation member registries as parallel-distributed queries. A federated query operates on data
4592 that belongs to all members of the federation. **Need to indicate whether results were partial**
4593 **because a member was not available??**

4594 A registry that is not a federation member must silently handle a federated query by treating it as
4595 a local query.

4596 **11.2.2.3 Membership in Multiple Federations**

4597 A registry may be a member of multiple federations. In such cases if the federated attribute of
4598 AdhocQueryRequest has the value of `true` then the registry must route the federated query to
4599 *all* federations that it is a member of.

4600 Alternatively, the client may specify the id of a specific federation that the registry is a member
4601 of, as the value of the `federation` parameter. The type of the federation parameter is ID.

4602 In such cases the registry must route the federated query to the specified federation only.

4603 **11.2.3 Federated Life Cycle Management Operations**

4604 Details on how to create and delete federations and how to join and leave a federation are
4605 described in 11.2.8.

4606 All lifecycle operations must be performed on a RegistryObject within its home registry using
4607 the operations defined by the LifeCycleManager interface. Unlike query requests, lifecycle
4608 management operations do not support any federated capabilities.

4609 **11.2.4 Federations and Local Caching of Remote Data**

4610 A federation member is not required to maintain a local cache of replicas of RegistryObjects and
4611 repository items that belong to other members of the federation.

4612 A registry may choose to locally cache some or all data from any other registry whether that
4613 registry is a federation member or not. Data caching is orthogonal to registry federation and is
4614 described in section 11.3.

4615 Since by default there is minimal replication in the members of a federation, the federation
4616 architecture scales well with respect to memory and disk utilization at each registry.

4617 Data replication is often necessary for performance, scalability and fault-tolerance reasons.

4618 **11.2.5 Caching of Federation Metadata**

4619 A special case for local caching is the caching of the Federation and Registry instances and
4620 related Associations that define a federation and its members. Such data is referred to as
4621 federation metadata. A federation member is required to locally cache the federation metadata,
4622 from the federation home for each federation that it is a member of. The federation member must
4623 keep the cached federation metadata synchronized with the master copy in the Federation home,
4624 within the time period specified by the replicationSyncLatency attribute of the Federation.
4625 Synchronization of cached Federation metadata may be done via synchronous polling or
4626 asynchronous event notification using the event notification feature of the registry.

4627 **11.2.6 Time Synchronization Between Registry Peers**

4628 Federation members are not required to synchronize their system clocks with each other.

4629 However, it is recommended that a Federation member keep its clock synchronized with an
4630 atomic clock server within the latency described by the replicationSyncLatency attribute of the
4631 Federation.

4632 **11.2.7 Federations and Security**

4633 Federation life cycle management operations abide by the same security rules as normal life
4634 cycle management.

4635 **11.2.8 Federation Life Cycle Management Protocols**

4636 This chapter describes the various operations that manage the life cycle of a federation and its
4637 membership. A key design objective is to allow federation life cycle operations to be done using
4638 existing LifeCycleManager interface of the registry in a stylized manner.

4639 **11.2.8.1 Joining a Federation**

4640 The following rules govern how a registry joins a federation:

4641 ?? Each registry must have exactly one Registry instance within that registry for which it is
4642 a home. The Registry instance is owned by the RegistryOperator and may be placed in
4643 the registry using any operator specific means. The Registry instance must never change
4644 its home registry via Object Relocation.

- 4645 ?? A registry may request to join an existing federation by submitting an instance of an
4646 Extramural Association that associates the Federation instance as sourceObject, to its
4647 Registry instance as targetObject, using an associationType of “HasFederationMember”.
4648 The home registry for the Association and the Federation objects must be the same.
4649 ?? The owner of the Federation instance must confirm the Extramural Association in order
4650 for the registry to be accepted as a member of the federation.

4651 **11.2.8.2 Creating a Federation**

4652 The following rules govern how a federation is created:

- 4653 ?? A Federation is created by submitting a Federation instance to a registry using
4654 SubmitObjectsRequest.
4655 ?? The registry where the Federation is submitted is referred to as the federation home.
4656 ?? The federation home may or may not be a member of that Federation.
4657 ?? A federation home may contain multiple Federation instances.

4658 **11.2.8.3 Leaving a Federation**

4659 The following rules govern how a registry leaves a federation:

- 4660 A registry may leave a federation at any time by removing its “HasFederationMember”
4661 Association instance that links it with the Federation instance. This is done using the normal
4662 RemoveObjectsRequest.

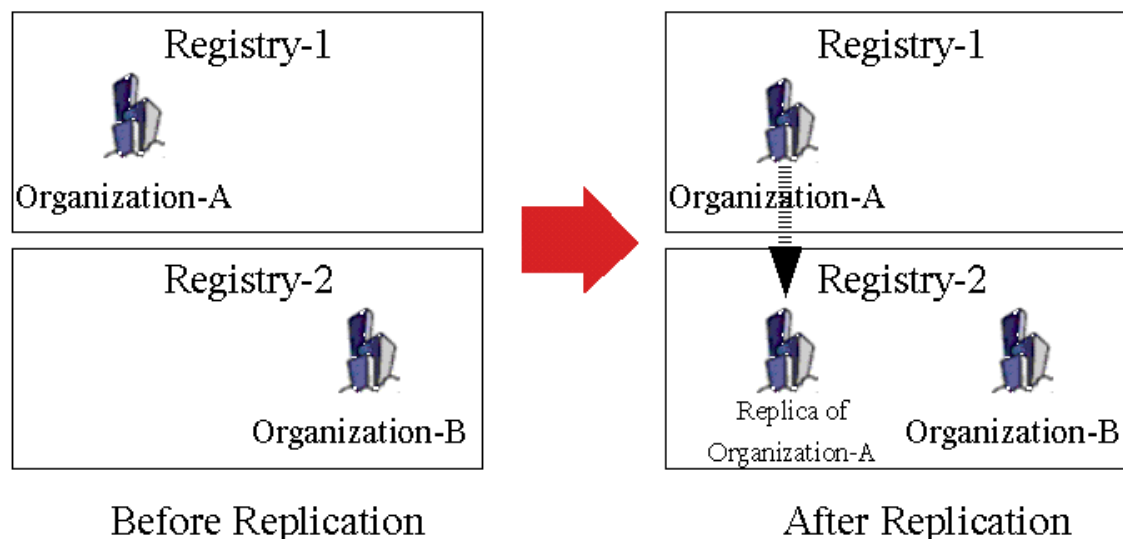
4663 **11.2.8.4 Dissolving a Federation**

4664 The following rules govern how a federation is dissolved:

- 4665 ?? A federation is dissolved by removing its Federation instance by sending a
4666 RemoveObjectsRequest to its home registry.
4667 ?? The removal of a Federation instance is controlled by the same Access Control Policies
4668 that govern any RegistryObject.
4669 ?? The removal of a Federation instance is controlled by the same life cycle management
4670 rules that govern any RegistryObject. Typically, this means that a federation may not be
4671 dissolved while it has federation members. It may however be deprecated at any time.
4672 Once a Federation is deprecated no new members can join it.
4673

4674 **11.3 Object Replication**

- 4675 RegistryObjects within a registry may be replicated in another registry. A replicated copy of a
4676 remote object is referred to as a replica of the remote object. The remote object may be the
4677 original object or it may itself be a replica. A replica from an original is referred to as a first-
4678 generation replica. A replica of a replica is referred to as a second-generation replica (and so on).
4679 The registry that replicates a remote object locally is referred to as the destination registry for the
4680 replication. The registry that contains the remote object being replicated is referred to as the
4681 source registry for the replication.
4682



☞☞ Figure 48: Object Replication

4683
4684
4685

4686 11.3.1 Use Cases for Object Replication

4687 A registry may create a local replica of a remote object for a variety of reasons. A few sample
4688 use cases follow:

- 4689 ○ Improve access time and fault tolerance via locally caching remote objects. For example,
4690 a registry may automatically create a local replica when a remote ObjectRef is submitted
4691 to the registry.
- 4692 ○ Improve scalability by distributing access to hotly contested object, such as NAICS
4693 scheme, across multiple replicas.
- 4694 ○ Enable cooperating registry features such as hierarchical registry topology and local
4695 caching of federation metadata.

4696 11.3.2 Queries And Replicas

4697 A registry must support client queries to consider a local replica of remote object as if it were a
4698 local object. Local replicas are considered within the extent of the data set of a registry as far as
4699 local queries are concerned.

4700 When a client submits a local query that retrieves a remote object by its id attribute, if the
4701 registry contains a local replica of that object then the registry should return the state defined by
4702 the local replica.

4703 11.3.3 Lifecycle Operations And Replicas

4704 A registry must not allow life cycle management operations on objects that are local replicas of
4705 remote objects. In such cases it should send an InvalidRequestException.

4706 **11.3.4 Object Replication and Federated Registries**

4707 Object replication capability is orthogonal to the registry federation capability. Objects may be
4708 replicated from any registry to any other registry without any requirement that the registries
4709 belong to the same federation.

4710 **11.3.5 Creating a Local Replica**

4711 Any Submitting Organization can create a replica by using the existing SubmitObjectsRequest. If
4712 a registry receives a SubmitObjectRequest which has an RegistryObjectList containing a remote
4713 ObjectRef, then it must create a replica for that remote ObjectRef.

4714 In addition to Submitting Organizations, a registry itself may create a replica under specific
4715 situations in a registry specific manner.

4716 Creating a local replica requires the destination registry to read the state of the remote object
4717 from the source registry and then create a local replica of the remote object.

4718 A registry may use normal QueryManager interface to read the state of a remote object (whether
4719 it is an original or a replica). No new APIs are needed to read the state of a remote object. Since
4720 query functionality does not need prior registration, no prior registration or contract is needed for
4721 a registry to read the state of a remote object.

4722 Once the state of the remote object has been read, a registry may use registry specific means to
4723 create a local replica of the remote object. Such registry specific means may include the use of
4724 the LifeCycleManager interface.

4725 **How to distinguish a replica from an original??**

- 4726 ○ **Add isReplica to RegistryObject??**
- 4727 ○ **Add home attribute to RegistryObject??**
- 4728 ○ **Should we have a sub-class of ObjectRef called RemoteObjectRef??**

4729 **11.3.6 Transactional Replication**

4730 Transactional replication enables a registry to replicate events in another registry in a
4731 transactionally consistent manner. This is typically the case when entire registries are replicated
4732 to another registry.

4733 Registry implementations are not required to implement transactional replication.

4734 **11.3.7 Keeping Replicas Current**

4735 A registry must keep its replicas current within the latency specified by the value of the
4736 replicationSyncLatency attribute defined by the registry. This includes removal of the replica
4737 when its original is removed from its home registry.

4738 Replicas may be kept current using the event notification feature of the registry or via periodic
4739 polling.

4740 **11.3.8 Write Operations on Local Replica**

4741 Local Replicas are read-only objects. Lifecycle management operations of RegistryObjects are
4742 not permitted on local replicas. All lifecycle management operation to RegistryObjects must be
4743 performed in the home registry for the object.

4744 **11.3.9 Tracking Location of a Replica**

4745 A local replica of a remote RegistryObject instance must have exactly one ObjectRef instance
 4746 within the local registry. The home attribute of the ObjectRef associated with the replica tracks
 4747 its home location. A RegistryObject must have exactly one home. The home for a RegistryObject
 4748 may change via Object Relocation as described in section 11.4. It is optional for a registry to
 4749 track location changes for replicas within it.

4750 **11.3.10 Remote Object References to a Replica**

4751 It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another
 4752 RegistryObject. In such cases the home contains the base URI to the home registry for the
 4753 replica.

4754 **11.3.11 Removing a Local Replica**

4755 A Submitting Organization can remove a replica by using the DeleteObjectsRequest. If a registry
 4756 receives a DeleteObjectRequest which has an ObjectRefList containing a remote ObjectRef, then
 4757 it must remove the local replica for that remote ObjectRef.

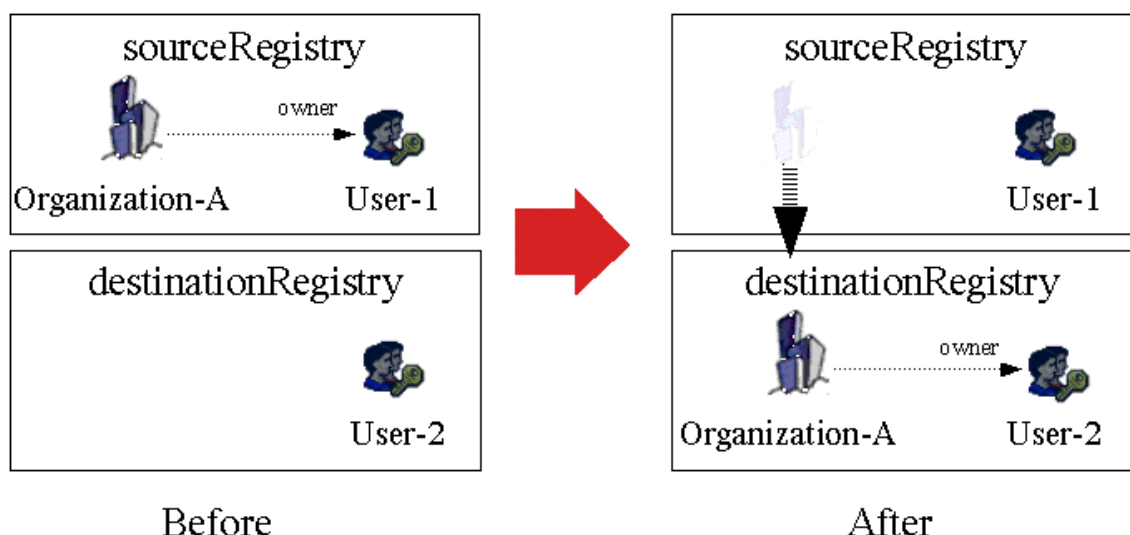
4758 **11.4 Object Relocation Protocol**

4759 **Need to replace this with Registry Import/Export capability??**

4760 Every RegistryObject has a home registry and a User within the home registry that is
 4761 the publisher or owner of that object. Initially, the home registry is the where the object
 4762 is originally submitted. Initially, the owner is the User that submitted the object.

4763 A RegistryObject may be relocated from one home registry to another home registry
 4764 using the Object Relocation protocol.

4765 Within the Object Relocation protocol, the new home registry is referred to as the
 4766 *destination* registry while the previous home registry is called the *source* registry.



4767

Before

4768

Figure 49: Object Relocation

After

4769 The User at the source registry who owns the objects being relocated is referred to as
4770 the *ownerAtSource*. The User at the destination registry, who is the new owner of the
4771 objects, is referred to as the *ownerAtDestination*. While the *ownerAtSource* and the
4772 *ownerAtDestination* may often be the same identity, the Object Relocation protocol
4773 treats them as two distinct identities.

4774 A special case usage of the Object Relocation protocol is to transfer ownership of
4775 RegistryObjects from one User to another within the same registry. In such cases the protocol is
4776 the same except for the fact that the source and destination registries are the same.

4777 Following are some notable points regarding object relocation:

- 4778 ?? Object relocation does not require that the source and destination registries be in the same
4779 federation or that either registry have a prior contract with the other.
- 4780 ?? Object relocation must preserve object id. While the home registry for a RegistryObject
4781 may change due to object relocation, its id never changes.
- 4782 ?? ObjectRelocation must preserve referential integrity of RegistryObjects. Relocated
4783 objects that have references to an object that did not get relocated must preserve their
4784 reference. Similarly objects that have references to a relocated object must also preserve
4785 their reference. Thus, relocating an object may result in making the value of a reference
4786 attribute go from being a local reference to being a remote reference or vice versa.
- 4787 ?? AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque
4788 transactionId identifying the relocateObjects transaction.
- 4789 ?? The requests defined by the Relocate Objects protocol must be sent to the source or
4790 destination registry only.
- 4791 ?? When an object is relocated an AuditableEvent of type "Relocated" happens. Relocated
4792 events must have the source and target registry's base URIs recorded as two Slots on the
4793 Relocated event. The names of these Slots are sourceRegistry and targetRegistry
4794 respectively.

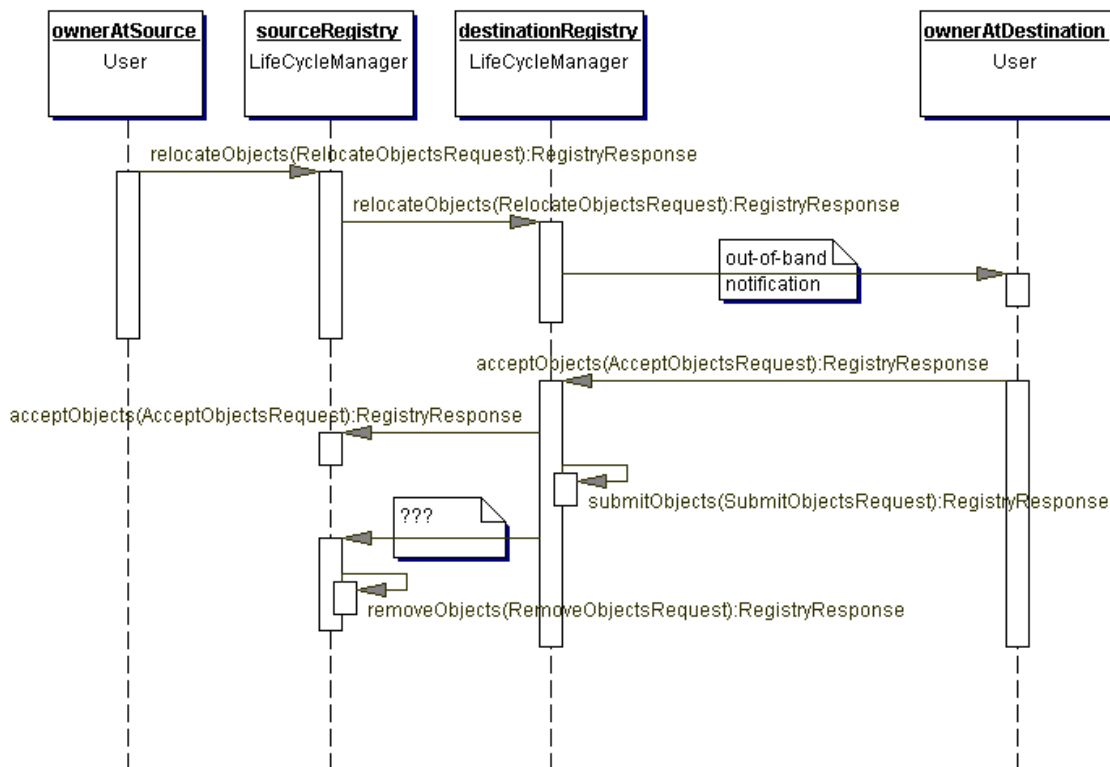


Figure 50: Relocate Objects Protocol

4795
4796

4797 Figure 40 illustrates the Relocate Objects Protocol. The participants in the protocol are the
4798 ownerAtSource and ownerAtDestination User instances as well as the LifeCycleManager
4799 interfaces of the sourceRegistry and destinationRegistry.

4800 The steps in the protocol are described next:

- 4801 1. The protocol is initiated by the ownerAtSource sending a RelocateObjectsRequest
4802 message to the LifeCycleManager interface of the sourceRegistry. The sourceRegistry
4803 must make sure that the ownerAtSource is authorized to perform this request. The id of
4804 this RelocateObjectsRequest is used as the transaction identifier for this instance of the
4805 protocol. This RelocateObjectsRequest message must contain an ObjectRefList element
4806 specifying the objects that are to be relocated.
- 4807 2. Next, the sourceRegistry must send a different RelocateObjectsRequest message to the
4808 LifeCycleManager interface of the destinationRegistry. This RelocateObjectsRequest
4809 message must not contain the ObjectRefList element. This message signals the
4810 destinationRegistry to participate in relocation protocol.
- 4811 3. The destinationRegistry must relay the RelocateObjectsRequest message to the
4812 ownerAtDestination using the event notification feature of the registry as described in
4813 chapter 10. This concludes the sequence of events that were a result of the
4814 ownerAtSource sending the RelocateObjectsRequest message to the sourceRegistry.
- 4815 4. The ownerAtDestination at a later time may send an AcceptObjectsRequest message to
4816 the destinationRegistry. This request must identify the object relocation transaction via
4817 the *relocateObjectsRequestId*. The value of this attribute must be the id of the original
4818 RelocateObjectsRequest.

- 4819 5. The destinationRegistry relays the AcceptObjectsRequest message to the sourceRegistry.
- 4820 The source registry returns the objects being relocated as an AdhocQueryResponse.
- 4821 6. The registry submits the relocated data to itself assigning the identity of the
- 4822 ownerAtDestination as the owner. The relocated data may be submitted to the destination
- 4823 registry using any registry specific means or a SubmitObjectsRequest.
- 4824 7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been
- 4825 safely committed. **Need to decide on the signal message name here??**
- 4826 8. The sourceRegistry removes the relocated objects using any registry specific means or a
- 4827 RemoveObjectsRequest. This concludes the Object Relocation transaction.

4828 **11.4.1 RelocateObjectsRequest**

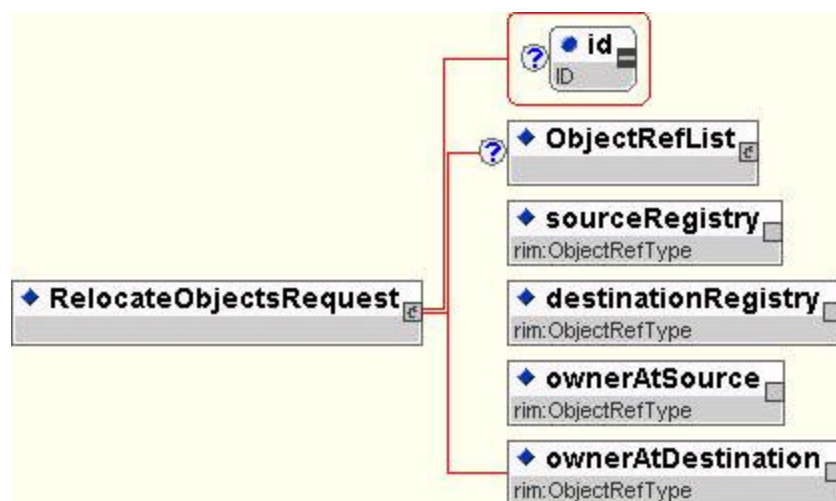


Figure 51: RelocateObjectsRequest XML Schema

4829
4830

4831 **11.4.1.1 Attribute id**

4832 The attribute id provides the transaction identifier for this instance of the protocol.

4833 **11.4.1.2 Element ObjectRefList**

4834 This element specifies the set of id attributes of RegistryObjects that are being relocated. This
4835 attribute must be present when the message is sent by the ownerAtSource to the sourceRegistry.
4836 This attribute must not be present when the message is sent on any other occasion.

4837 **11.4.1.3 Element sourceRegistry**

4838 This element specifies the ObjectRef to the sourceRegistry Registry instance. The value of this
4839 attribute must be a local reference when the message is sent by the ownerAtSource to the
4840 sourceRegistry.

4841 **11.4.1.4 Element destinationRegistry**

4842 This element specifies the ObjectRef to the destinationRegistry Registry instance.

4843 **11.4.1.5 Element ownerAtSource**

4844 This element specifies the ObjectRef to the ownerAtSource User instance.

4845 **11.4.1.6 Element ownerAtDestination**

4846 This element specifies the ObjectRef to the ownerAtDestination User instance.

4847 **11.4.2 AcceptObjectsRequest**

4848

4849 **11.4.2.1 Attribute relocateObjectsRequestId**4850 The attribute relocateObjectsRequestId provides the transaction identifier for this instance of the
4851 protocol.4852 **11.4.3 Object Relocation and Remote ObjectRefs**

4853 The following scenario describes what typically happens when a person moves:

- 4854 1. When a person moves from one house to another, other persons may have their old postal
4855 addresses.
- 4856 2. When a person moves, they leave their new address as the forwarding address with the
4857 post office.
- 4858 3. The post office forwards their mail for some time to their new address.
- 4859 4. Eventually the forwarding request expires and the post office no longer forwards mail for
4860 that person.
- 4861 5. During this forwarding interval the person notifies interested parties of their change of
4862 address.

4863 The registry must support a similar model for relocation of RegistryObjects. The following steps
4864 describe the expected behavior when an object is relocated.

- 4865 1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other
4866 RegistryObjects may have remote ObjectRefs to O1.
- 4867 2. The registry R1 must leave an AuditableEvent of type Relocated that includes the home
4868 URI for the new registry R2.
- 4869 3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it
4870 must forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the
4871 client. The object O1 must include the home URI to R2 within the optional home
4872 attribute of RegistryObject. Clients are advised to check the home attribute and update
4873 the home attribute of their local ObjectRef to match the new home URI value for the
4874 object.
- 4875 4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no
4876 longer required to relay requests for O1 to R2 transparent to the client. Instead R1 must
4877 return an ObjectNotFoundException.
- 4878 5. Clients that are interested in the relocation of O1 and being notified of its new address
4879 may choose to be notified by having a prior subscription using the event notification
4880 facility of the registry. For example a Registry that has a remote ObjectRefs to O1 may
4881 create a subscription on relocation events for O1. This however, is not required behavior.

4882 11.4.4 Notification of Object Relocation

4883 This section describes how the destinationRegistry uses the event notification feature of the
4884 registry to notify the ownerAtDestination of a Relocated event.

4885 <details TBD>

4886 11.4.5 Object Relocation and Timeouts

4887 No timeouts are specified for the Object Relocation protocol. Registry implementations may
4888 cleanup incomplete Object Relocation transactions in a registry specific manner as an
4889 administrative task using registry specific policies.

4890 **12 Registry Security**

4891 This chapter describes the security features of the ebXML Registry. It is assumed that the reader
4892 is familiar with the security related classes in the Registry information model as described in
4893 [ebRIM]. Security glossary terms can be referenced from RFC 2828.

4894 **12.1 Security Concerns**

4895 In the current version of this specification, we address data integrity and source integrity (item 1
4896 in Appendix E.1). We have used a minimalist approach to address the access control concern as
4897 in item 2 of Appendix E.1. Essentially, “any known entity (Submitting Organization) can publish
4898 content and anyone can view published content.” The Registry information model has been
4899 designed to allow more sophisticated security policies in future versions of this specification.

4900 **12.2 Integrity of Registry Content**

4901 It is assumed that most business registries do not have the resources to validate the veracity of
4902 the content submitted to them. "The mechanisms described in this section can be used to ensure
4903 that any tampering with the content submitted by a Submitting Organization can be detected.
4904 Furthermore, these mechanisms support unambiguous identification of the Responsible
4905 Organization for any registry content. The Registry Client has to sign the contents before
4906 submission – otherwise the content will be rejected. Note that in the discussions in this section
4907 we assume a Submitting Organization to be also the Responsible Organization. Future version of
4908 this specification may provide more examples and scenarios where a Submitting Organization
4909 and Responsible Organization are different.

4910 **12.2.1 Message Payload Signature**

4911 The integrity of the Registry content requires that all submitted content be signed by the Registry
4912 client. The signature on the submitted content ensures that:

4913 ?? Any tampering of the content can be detected.

4914 ?? The content’s veracity can be ascertained by its association with a specific Submitting
4915 Organization.

4916 This section specifies the requirements for generation, packaging and validation of payload
4917 signatures. A payload signature is packaged with the payload. Therefore the requirements apply
4918 regardless of whether the Registry Client and the Registration Authority communicate over
4919 vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML
4920 Messaging Service does not specify the generation, validation and packaging of payload
4921 signatures. The specification of payload signatures is left upto the application (such as Registry).
4922 So the requirements on the payload signatures augment the [ebMS] specification.

4923 **Use Case**

4924 This Use Case illustrates the use of header and payload signatures (we discuss header signatures
4925 later).

4926 ?? RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the
4927 content along with the payload signature to the Registry.

4928 ?? RC2 (Registry Client 2) retrieves RC1’s content from the Registry.

- 4929 ?? RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
4930 the content, the response from the Registration Authority to RC2 contains the following:
- 4931 ○ Payload containing the content that has been published by RC1.
 - 4932 ○ RC1's payload signature (represented by a ds:Signature element) over RC1's published
4933 content.
 - 4934 ○ The public key for validating RC1's payload signature in ds:Signature element (using the
4935 KeyInfo element as specified in [XMLDSIG]) so RC2 can obtain the public key for
4936 signature (e.g. retrieve a certificate containing the public key for RC1).
 - 4937 ○ A ds:Signature element containing the header signature. Note that the Registration
4938 Authority (not RC1) generates this signature.

4939 **12.2.2 Payload Signature Requirements**

4940 **12.2.2.1 Payload Signature Packaging Requirements**

4941 A payload signature is represented by a ds:Signature element. The payload signature must be
4942 packaged with the payload as specified here. This packaging assumes that the payload is always
4943 signed.

4944 ?? The payload and its signature must be enclosed in a MIME multipart message with a
4945 Content-Type of multipart/related.

4946 ?? The first body part must contain the XML signature as specified in Section 12.2.2.2,
4947 "Payload Signature Generation Requirements".

4948 ?? The second body part must be the content.

4949 The packaging of the payload signature with two payloads is as shown in the example in Section
4950 8.4.2.

4951 **12.2.2.2 Payload Signature Generation Requirements**

4952 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in
4953 this section. Note: the "ds" name space reference is to <http://www.w3.org/2000/09/xmlsig#>

4954 ?? ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
4955 using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a
4956 client may use any of these attributes. However, signing using the following Algorithm
4957 attribute: <http://www.w3.org/2000/09/xmlsig#dsa-sha1> will allow interoperability with all
4958 XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this
4959 algorithm.

4960 The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The following
4961 Canonicalization algorithm (specified in [XMLDSIG]) must be supported
4962 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

4963 ?? One ds:Reference element to reference each of the payloads that needs to be signed must be
4964 created. The ds:Reference element:

- 4965 ○ Must identify the payload to be signed using the URI attribute of the ds:Reference
4966 element.
- 4967 ○ Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be
4968 support the following digest algorithm:
- 4969 ○ <http://www.w3.org/2000/09/xmlsig#sha1>

- 4970 o Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].
- 4971 The ds:SignatureValue must be generated as specified in [XMLDSIG].
- 4972 The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject
- 4973 to the requirements stated in Section 12.4, "KeyDistribution and KeyInfo element".

4974 **12.2.2.3 Message Payload Signature Validation**

- 4975 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

4976 **12.2.2.4 Payload Signature Example**

- 4977 The following example shows the format of the payload signature:

```

4978
4979 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
4980 <ds:SignedInfo>
4981   <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1" />
4982   <ds:CanonicalizationMethod>
4983     Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
4984   </ds:CanonicalizationMethod>
4985   <ds:Reference URI=#Payload1>
4986     <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmldsig#sha1">
4987     <ds:DigestValue> ... </ds:DigestValue>
4988   </ds:Reference>
4989 </ds:SignedInfo>
4990 <ds:SignatureValue> ... </ds:SignatureValue>
4991 </ds:Signature>
4992

```

4993 **12.3 Authentication**

- 4994 The Registry must be able to authenticate the identity of the Principal associated with client
- 4995 requests. The identity of the Principal can be identified by verifying the message header
- 4996 signature with the certificate of the Principal. The certificate may be in the message itself or
- 4997 provided to the registry through means unspecified in this specification. If not provided in the
- 4998 message, this specification does not specify how the Registry correlates a specific message with
- 4999 a certificate. Authentication of each payload must also be possible by using the signature
- 5000 associated with each payload. Authentication is also required to identify the "privileges" a
- 5001 Principal is authorized ("authorization") to have with respect to specific objects in the Registry.

- 5002 The Registry must perform authentication on a per message basis. From a security point of view,
- 5003 all messages are independent and there is no concept of a session encompassing multiple
- 5004 messages or conversations. Session support may be added as an optimization feature in future
- 5005 versions of this specification.

- 5006 It is important to note that the message header signature can only guarantee data integrity and it
- 5007 may be used for Authentication knowing that it is vulnerable to replay types of attacks. True
- 5008 support for authentication requires timestamps or nonce (nonrecurring series of numbers to
- 5009 identify each message) that are signed.

5010 **12.3.1 Message Header Signature**

5011 Message headers are signed to provide data integrity while the message is in transit. Note that the
5012 signature within the message header also signs the digests of the payloads.

5013 Header Signature Requirements

5014 Message headers can be signed and are referred to as a header signature. When a request is sent
5015 by a Registered User, the Registration Authority may use the pre-established contract or a default
5016 policy to determine whether the response contains a header signature. . When a request is sent
5017 by a Registry Guest, the Registration Authority may use a default policy to determine whether
5018 the response contains a header signature.

5019 This section specifies the requirements for generation, packaging and validation of a header
5020 signature. These requirements apply when the Registry Client and Registration Authority
5021 communicate using vanilla SOAP with Attachments. When ebXML MS is used for
5022 communication, then the message handler (i.e. [ebMS]) specifies the generation, packaging and
5023 validation of XML signatures in the SOAP header. Therefore the header signature requirements
5024 do not apply when the ebXML MS is used for communication. However, payload signature
5025 generation requirements (specified elsewhere in this document) do apply whether vanilla SOAP
5026 with Attachments or ebXML MS is used for communication.

5027 **12.3.1.1 Packaging Requirements**

5028 A header signature is represented by a ds:Signature element. The ds:Signature element generated
5029 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature
5030 element in the SOAP header field is shown in Section 8.4.2.

5031 **12.3.1.2 Header Signature Generation Requirements**

5032 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in
5033 this section. A ds:Signature element contains:

- 5034 ?? ds:SignedInfo
- 5035 ?? ds:SignatureValue
- 5036 ?? ds:KeyInfo

5037 The ds:SignedInfo element must be generated as follows:

- 5038 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
5039 using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,
5040 a client must be capable of signing using only the following Algorithm attribute:
5041 <http://www.w3.org/2000/09/xmldsig#dsa-sha1> This algorithm is being chosen because all
5042 XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
- 5043 2. The ds:SignedInfo element must contain a ds:CanonicalizationMethod element. The
5044 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
5045 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- 5046 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.
5047 This signs the entire ds:Reference element and:
 - 5048 ○ Must include the following ds:Transform:
5049 <http://www.w3.org/2000/09/xmldsig#enveloped-signature>

- 5050 This ensures that the signature (which is embedded in the <SOAP-ENV:Header>
5051 element) is not included in the signature calculation.
- 5052 ○ Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the
5053 ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) .
5054 The URI attribute must be "".
 - 5055 ○ Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support
5056 the following digest algorithm: <http://www.w3.org/2000/09/xmlsig#sha1>
 - 5057 ○ Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].
- 5058 The ds:SignatureValue must be generated as specified in [XMLDSIG].
- 5059 The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in
5060 Section 12.4, “KeyDistribution and KeyInfo element”.

5061 12.3.1.3 Header Signature Validation Requirements

5062 The ds:Signature element for the ebXML message header must be validated by the recipient as
5063 specified by [XMLDSIG].

5064 12.3.1.4 Header Signature Example

5065 The following example shows the format of a header signature:

```
5066 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
5067   <ds:SignedInfo>
5068     <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmlsig#dsa-sha1/>
5069     <ds:CanonicalizationMethod>
5070       Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
5071     </ds:CanonicalizationMethod>
5072     <ds:Reference URI= "">
5073       <ds:Transform>
5074         http://www.w3.org/2000/09/xmlsig#enveloped-signature
5075       </ds:Transform>
5076       <ds:DigestMethod DigestAlgorithm="./xmlsig#sha1">
5077       <ds:DigestValue> ... </ds:DigestValue>
5078     </ds:Reference>
5079   </ds:SignedInfo>
5080   <ds:SignatureValue> ... </ds:SignatureValue>
5081 </ds:Signature>
```

5084 12.4 Key Distribution and KeyInfo Element

5085 To validate a signature, the recipient of the signature needs the public key corresponding to the
5086 signer’s private key. The participants may use the KeyInfo field of ds:Signature, or distribute the
5087 public keys out-of-band. In this section we consider the case when the public key is sent in the
5088 KeyInfo field. The following use cases need to be handled:

- 5089 ?? Registration Authority needs the public key of the Registry Client to validate the signature

5090 ?? Registry Client needs the public key of the Registration Authority to validate the Registry's
5091 signature.
5092 ?? Registry Client RC1 needs the public key of Registry Client (RC2) to validate the content
5093 signed by RC1.
5094 ?? [XMLDSIG] provides a *ds:KeyInfo* element that can be used to pass the recipient
5095 information for retrieving the public key. *ds:KeyInfo* is an optional element as specified in
5096 [XMLDSIG]. This field together with the procedures outlined in this section is used to
5097 securely pass the public key to a recipient. *ds:KeyInfo* can be used to pass information such
5098 as keys, certificates, names etc. The intended usage of *KeyInfo* field is to send the X509
5099 Certificate, and subsequently extract the public key from the certificate. Therefore, the
5100 *KeyInfo* field must contain a X509 Certificate as specified in [XMLDSIG], if the *KeyInfo*
5101 field is present.

5102 The following assumptions are also made:

- 5103 1. A Certificate is associated both with the Registration Authority and a Registry Client.
- 5104 2. A Registry Client registers its certificate with the Registration Authority. The mechanism
5105 used for this is not specified here.
- 5106 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local
5107 key store. The mechanism is not specified here.

5108 Couple of scenarios on the use of *KeyInfo* field is in Appendix F.8.

5109 **12.5 Confidentiality**

5110 **12.5.1 On-the-wire Message Confidentiality**

5111 It is suggested but not required that message payloads exchanged between clients and the
5112 Registry be encrypted during transmission. This specification does not specify how payload
5113 encryption is to be done.

5114 **12.5.2 Confidentiality of Registry Content**

5115 In the current version of this specification, there are no provisions for confidentiality of Registry
5116 content. All content submitted to the Registry may be discovered and read by any client. This
5117 implies that the Registry and the client need to have an a priori agreement regarding encryption
5118 algorithm, key exchange agreements, etc. This service is not addressed in this specification.

5119 **12.6 Authorization**

5120 The Registry must provide an authorization mechanism based on the information model defined
5121 in [ebRIM]. In this version of the specification the authorization mechanism is based on a default
5122 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of
5123 this specification will allow for custom Access Control Policies to be defined by the Submitting
5124 Organization. The authorization is going to be applied on a specific set of privileges. A
5125 privilege is the ability to carry a specific action.

5126 **12.6.1 Actions**

5127 Life Cycle Actions

5128 submitObjects

5129 updateObjects

5130 addSlots

5131 removeSlots

5132 approveObjects

5133 deprecateObjects

5134 removeObjects

5135 Read Actions

5136 The various getXXX() methods in QueryManagement Service.

5137 **12.7 Access Control**

5138 The Registry must create a default AccessControlPolicy object that grants the default
 5139 permissions to Registry users (as defined in Section 5.3 of this document) to access Registry
 5140 Objects based upon their assigned role. The following table defines the Permissions granted by
 5141 the Registry to the various pre-defined roles for Registry users.

5142 **Table 9: Role to Permissions Mapping**

Role	Permissions
ContentOwner	Access to <i>all</i> methods on Registry Objects that are owned by the actor who is assigned this role.
RegistryAdministrator	Access to <i>all</i> methods on <i>all</i> Registry Objects
GuestReader	Access to <i>some</i> read-only (getXXX) methods on <i>some</i> Registry Objects (read-only access to some content) as defined in the default access control policy.

5143 The mapping of actors listed in Section 5.3 and their default roles in the following table.

5144

☞☞ **Table 10: Default Actor to Role Mappings**

Actor	Role
Submitting Organization Responsible Organization	ContentOwner
Registry Administrator Registration Authority	RegistryAdministrator
Registry Guest	GuestReader
Registry Reader	GuestReader

5145 The Registry must implement the default AccessControlPolicy and associate it with all Objects
5146 in the Registry. The following list summarizes the role-based AccessControlPolicy:

5147 ?? Only a Registered User can publish content.

5148 ?? Any unauthenticated Registry Client can only access some read-only (getXXX) methods
5149 permitted for GuestReader role. The Registry must assign the default GuestReader role to
5150 such Registry Clients.

5151 ?? The SubmittingOrganization has access to all methods of Registry Objects submitted or
5152 updated by the Submitting Organization. This version of the specification does not
5153 distinguish between Submitting Organization and Responsible Organization, and assumes
5154 that the Submitting Organization is also the Responsible Organization.

5155 ?? The RegistryAdministrator and Registry Authority have access to all methods on all
5156 Registry Objects

5157 ?? At the time of content submission, the Registry must assign the default ContentOwner role to
5158 the Submitting Organization (SO) as authenticated by the credentials in the submission
5159 message. In the current version of this specification, the Submitting Organization will be the
5160 DN (Distinguished Name) as identified by the certificate presented during authentication.
5161 This version of the specification does not specify where credentials go in the message.

5162 ?? A Registry Reader can access *some* read-only (getXXX) methods on *some* Registry Objects
5163 (read-only access to some content) as defined in the custom access control policy agreed
5164 upon in a contract between the Registry and Registry Reader. Such access MAY be a
5165 superset of access granted to the GuestReader role.

5166 **Appendix A Web Service Architecture**

5167 **A.1 Registry Service Abstract Specification**

5168 The normative definition of the Abstract Registry Service in WSDL is defined at the following
5169 location on the web:

5170 <http://www.oasis-open.org/committees/regrep/documents/3.0/services/Registry.wsdl>

5171 **Need to update WSDL for V3??**

5172 **A.2 Registry Service SOAP Binding**

5173 The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined
5174 at the following location on the web:

5175 <http://www.oasis-open.org/committees/regrep/documents/3.0/services/RegistrySOAPBinding.wsdl>

5176

5177 **Appendix B ebXML Registry Schema Definitions**

5178 **B.1 RIM Schema**

5179 The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the
5180 following location on the web:

5181 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>

5182 **B.2 Query Schema**

5183 The normative XML Schema definition for the XML query syntax for the registry service
5184 interface can be found at the following location on the web:

5185 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd>

5186 **B.3 Registry Services Interface Schema**

5187 The normative XML Schema definition that defines the XML requests and responses supported
5188 by the registry service interfaces in this document can be found at the following location on the
5189 web:

5190 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rs.xsd>

5191 **B.4 Examples of Instance Documents**

5192 A growing number of non-normative XML instance documents that conform to the normative
5193 Schema definitions described earlier may be found at the following location on the web:

5194 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/>

5195

5196 **Appendix C Interpretation of UML Diagrams**

5197 This section describes in *abstract terms* the conventions used to define ebXML business process
5198 description in UML.

5199 **C.1 UML Class Diagram**

5200 A UML class diagram is used to describe the Service Interfaces required to implement an
5201 ebXML Registry Services and clients. The UML class diagram contains:

- 5202
- 5203 1. A collection of UML interfaces where each interface represents a Service Interface for a
5204 Registry service.
 - 5205 2. Tabular description of methods on each interface where each method represents an
5206 Action (as defined by [ebCPP]) within the Service Interface representing the UML
5207 interface.
 - 5208 3. Each method within a UML interface specifies one or more parameters, where the type of
5209 each method argument represents the ebXML message type that is exchanged as part of
5210 the Action corresponding to the method. Multiple arguments imply multiple payload
5211 documents within the body of the corresponding ebXML message.

5212 **C.2 UML Sequence Diagram**

5213 A UML sequence diagram is used to specify the business protocol representing the interactions
5214 between the UML interfaces for a Registry specific ebXML business process. A UML sequence
5215 diagram provides the necessary information to determine the sequencing of messages, request to
5216 response association as well as request to error response association.

5217 Each sequence diagram shows the sequence for a specific conversation protocol as method calls
5218 from the requestor to the responder. Method invocation may be synchronous or asynchronous
5219 based on the UML notation used on the arrow-head for the link. A half arrow-head represents
5220 asynchronous communication. A full arrow-head represents synchronous communication.

5221 Each method invocation may be followed by a response method invocation from the responder to
5222 the requestor to indicate the ResponseName for the previous Request. Possible error response is
5223 indicated by a conditional response method invocation from the responder to the requestor. See
5224 Figure 7 on page 34 for an example.

5225 Appendix D SQL Query

5226 D.1 SQL Query Syntax Specification

5227 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The
5228 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax
5229 conforms to the <query specification>, modulo the restrictions identified below:

- 5230 1. A <select list> may contain at most one <select sublist>.
- 5231 2. In a <select list> must be is a single column whose data type is UUID, from the table in the
5232 <from clause>.
- 5233 3. A <derived column> may not have an <as clause>.
- 5234 4. <table expression> does not contain the optional <group by clause> and <having clause>
5235 clauses.
- 5236 5. A <table reference> can only consist of <table name> and <correlation name>.
- 5237 6. A <table reference> does not have the optional AS between <table name> and
5238 <correlation name>.
- 5239 7. There can only be one <table reference> in the <from clause>.
- 5240 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows
5241 for the right hand side of the <in predicate> to be limited to a restricted <query
5242 specification> as defined above.
- 5243 9. A <search condition> within the <where clause> may not include a <query expression>.
- 5244 10. Simple joins are allowed only if they are based on indexed columns within the relational
5245 schema.
- 5246 11. The SQL query syntax allows for the use of <sql invoked routines> invocation from
5247 [SQL/PSM] as the RHS of the <in predicate>.

5248 D.2 Non-Normative BNF for Query Syntax Grammar

5249 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as
5250 an aid to implementers. Since this BNF is not based on [SQL] it is provided as non-normative
5251 syntax. For the normative syntax rules see Appendix D.1.

```

5252 /*****
5253 * The Registry Query (Subset of SQL-92) grammar starts here
5254 *****/
5255 RegistryQuery = SQLSelect [ ";" ]
5256
5257 SQLSelect = "SELECT" [ "DISTINCT" ] SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
5258
5259 SQLSelectCols = ID
5260
5261 SQLTableList = SQLTableRef
5262
5263 SQLTableRef = ID
5264
5265 SQLWhere = "WHERE" SQLOrExpr
5266
5267 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
5268
5269
5270

```



```

5271 SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*
5272
5273 SQLNotExpr = [ "NOT" ] SQLCompareExpr
5274
5275 SQLCompareExpr =
5276     (SQLColRef "IS") SQLIsClause
5277     | SQLSumExpr [ SQLCompareExprRight ]
5278
5279
5280 SQLCompareExprRight =
5281     SQLLikeClause
5282     | SQLInClause
5283     | SQLCompareOp SQLSumExpr
5284
5285 SQLCompareOp =
5286     "="
5287     | "<>"
5288     | ">"
5289     | ">="
5290     | "<"
5291     | "<="
5292
5293 SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
5294
5295 SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
5296
5297 SQLLValueElement = "NULL" | SQLSelect
5298
5299 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
5300
5301 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
5302
5303 SQLPattern = STRING_LITERAL
5304
5305 SQLLiteral =
5306     STRING_LITERAL
5307     | INTEGER_LITERAL
5308     | FLOATING_POINT_LITERAL
5309
5310 SQLColRef = SQLLvalue
5311
5312 SQLLvalue = SQLLvalueTerm
5313
5314 SQLLvalueTerm = ID ( "." ID )*
5315
5316 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
5317
5318 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr )*
5319
5320 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
5321
5322 SQLTerm = "(" SQLOrExpr ")"
5323     | SQLColRef
5324     | SQLLiteral
5325
5326 INTEGER_LITERAL = ([ "0"-"9" ])+
5327
5328 FLOATING_POINT_LITERAL =
5329     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ (EXONENT)?
5330     | "." ([ "0"-"9" ])+ (EXONENT)?
5331     | ([ "0"-"9" ])+ EXPONENT
5332     | ([ "0"-"9" ])+ (EXONENT)?
5333
5334 EXPONENT = [ "e", "E" ] ( [ "+" , "-" ] )? ([ "0"-"9" ])+
5335
5336 STRING_LITERAL: "'" (~[ "'" ])* ( "'" (~[ "'" ])* )* "'"
5337
5338 ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
5339 LETTER = [ "A"-"Z", "a"-"z" ]
5340 DIGIT = [ "0"-"9" ]

```

5341 **D.3 Relational Schema For SQL Queries**

5342 The normative Relational Schema definition for SQL queries can be found at the following
5343 location on the web:

5344 <http://www.oasis-open.org/committees/regrep/documents/3.0/sql/database.sql>

5345

5346 The stored procedures that must be supported by the SQL query feature are defined at the following
5347 location on the web:

5348 <http://www.oasis-open.org/committees/regrep/documents/3.0/sql/storedProcedures.sql>

5349

5350 **Appendix E Security Implementation Guideline**

5351 This section provides a suggested blueprint for how security processing may be implemented in
5352 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have
5353 different implementations as long as they support the default security roles and authorization
5354 rules described in this document.

5355 **E.1 Security Concerns**

5356 The security risks broadly stem from the following concerns. After a description of these
5357 concerns and potential solutions, we identify the concerns that we address in the current
5358 specification

- 5359 1. Is the content of the registry (data) trustworthy?
 - 5360 a) How to make sure “what is in the registry” is “what is put there” by a submitting
5361 organization? This concern can be addressed by ensuring that the publisher is
5362 authenticated using digital signature (Source Integrity), message is not corrupted during
5363 transfer using digital signature (Data Integrity), and the data is not altered by
5364 unauthorized subjects based on access control policy (Authorization)
 - 5365 b) How to protect data while in transmission?
5366 Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data
5367 Confidentiality that can be addressed by encrypting the data in transmission. How to
5368 protect against a replay attack?
 - 5369 c) Is the content up to date? The versioning as well as any time stamp processing, when
5370 done securely will ensure the “latest content” is guaranteed to be the latest content.
 - 5371 d) How to ensure only bona fide responsible organizations add contents to registry?
5372 Ensuring Source Integrity (as in 1a).
 - 5373 e) How to ensure that bona fide publishers add contents to registry only at authorized
5374 locations? (System Integrity)
 - 5375 f) What if the publishers deny modifying certain content after-the-fact? To prevent this
5376 (Nonrepudiation) audit trails may be kept which contain signed message digests.
 - 5377 g) What if the reader denies getting information from the registry?
- 5378 2. How to provide selective access to registry content? The broad answer is, by using an access
5379 control policy – applies to (a), (b), and (c) directly.
 - 5380 a) How does a submitting organization restrict access to the content to only specific registry
5381 readers?
 - 5382 b) How can a submitting organization allow some “partners” (fellow publishers) to modify
5383 content?
 - 5384 c) How to provide selective access to partners the registry usage data?
 - 5385 d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw
5386 failure of the registry security components? The solution to this problem is by having
5387 System Integrity.
 - 5388 e) Data confidentiality of RegistryObject

- 5389 3. How do we make “who can see what” policy itself visible to limited parties, even excluding
5390 the administrator (self & confidential maintenance of access control policy). By making sure
5391 there is an access control policy for accessing the policies themselves.
- 5392 4. How to transfer credentials? The broad solution is to use credentials assertion (such as being
5393 worked on in [Security Assertions Markup Language \(SAML\)](#)). Currently, Registry does not
5394 support the notion of a session. Therefore, some of these concerns are not relevant to the
5395 current specification.
- 5396 a) How to transfer credentials (authorization/authentication) to federated registries?
5397 b) How do aggregators get credentials (authorization/authentication) transferred to them?
5398 c) How to store credentials through a session?

5399 **E.2 Authentication**

- 5400 1. As soon as a message is received, the first work is the authentication. A principal object is
5401 created.
- 5402 2. If the message is signed, it is verified (including the validity of the certificate) and the DN of
5403 the certificate becomes the identity of the principal. Then the Registry is searched for the
5404 principal and if found, the roles and groups are filled in.
- 5405 3. If the message is not signed, an empty principal is created with the role RegistryGuest. This
5406 step is for symmetry and to decouple the rest of the processing.
- 5407 4. Then the message is processed for the command and the objects it will act on.

5408 **E.3 Authorization**

5409 For every object, the access controller will iterate through all the AccessControlPolicy objects
5410 with the object and see if there is a chain through the permission objects to verify that the
5411 requested method is permitted for the Principal. If any of the permission objects which the object
5412 is associated with has a common role, or identity, or group with the principal, the action is
5413 permitted.

5414 **E.4 Registry Bootstrap**

5415 When a Registry is newly created, a default Principal object should be created with the identity
5416 of the Registry Admin’s certificate DN with a role RegistryAdmin. This way, any message
5417 signed by the Registry Admin will get all the privileges.

5418 When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the
5419 default AccessControlPolicy. This includes the creation of the necessary Permission instances as
5420 well as the Privileges and Privilege attributes.

5421 **E.5 Content Submission – Client Responsibility**

5422 The Registry client must sign the contents before submission – otherwise the content will be
5423 rejected.

5424 **E.6 Content Submission – Registry Responsibility**

- 5425 1. As with any other request, the client will first be authenticated. In this case, the Principal
5426 object will get the DN from the certificate.
- 5427 2. As per the request in the message, the RegistryEntry will be created.
- 5428 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 5429 4. If a principal with the identity of the SO is not available, an identity object with the SO's DN
5430 is created.
- 5431 5. A principal with this identity is created.

5432 **E.7 Content Delete/Deprecate – Client Responsibility**

5433 The Registry client must sign the header before submission, for authentication purposes;
5434 otherwise, the request will be rejected

5435 **E.8 Content Delete/Deprecate – Registry Responsibility**

- 5436 1. As with any other request, the client will first be authenticated. In this case, the Principal
5437 object will get the DN from the certificate. As there will be a principal with this identity in
5438 the Registry, the Principal object will get all the roles from that object
- 5439 2. As per the request in the message (delete or deprecate), the appropriate method in the
5440 RegistryObject class will be accessed.
- 5441 3. The access controller performs the authorization by iterating through the Permission objects
5442 associated with this object via the singleton default AccessControlPolicy.
- 5443 4. If authorization succeeds then the action will be permitted. Otherwise an error response is
5444 sent back with a suitable AuthorizationException error message.

5445 **E.9 Using ds:KeyInfo Field**

5446 Two typical usage scenarios for ds:KeyInfo are described below.

5447 **Scenario 1**

- 5448 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
- 5449 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
- 5450 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
- 5451 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature
- 5452 5. Registration Authority validates the header signature using the public key from the
5453 certificate.
- 5454 6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the
5455 certificate from the KeyInfo field of the payload signature. Note that this step is not an
5456 essential one if the onus of validation is that of the eventual user, another Registry Client, of
5457 the content.

5458 **Scenario 2**

- 5459 1. RC1 signs the payload and SOAP envelope using its private key and publishes to the
5460 Registry.
- 5461 2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
- 5462 3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature.
5463 This step is required in addition to step 2 because when RC2 retrieves content, it should see
5464 RC1's signature with the payload.
- 5465 4. RC2 retrieves content from the Registry.
- 5466 5. Registration Authority signs the SOAP envelope using its private key. Registration Authority
5467 sends RC1's content and the RC1's signature (signed by RC1).
- 5468 6. Registration Authority need not send its certificate in the KeyInfo field since RC2 is assumed
5469 to have obtained the Registration Authority's certificate out of band and installed it in its
5470 local key store.
- 5471 7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the
5472 Registration Authority's signature.
- 5473 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates
5474 the signature on the payload.

5475 **Appendix F Native Language Support (NLS)**

5476 **F.1 Definitions**

5477 Although this section discusses only character set and language, the following terms have to be
5478 defined clearly.

5479 **F.1.1 Coded Character Set (CCS):**

5480 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of
5481 CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

5482 **F.1.2 Character Encoding Scheme (CES):**

5483 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are
5484 ISO-2022, UTF-8.

5485 **F.1.3 Character Set (charset):**

5486 ?? charset is a set of rules for mapping from a sequence of octets to a sequence of
5487 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.

5488 ?? A list of registered character sets can be found at [IANA].

5489 **F.2 NLS And Request / Response Messages**

5490 For the accurate processing of data in both registry client and registry services, it is essential to
5491 know which character set is used. Although the body part of the transaction may contain the
5492 charset in xml encoding declaration, registry client and registry services shall specify charset
5493 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a
5494 text/xml entity is received with the charset parameter omitted, MIME processors and XML
5495 processors MUST use the default charset value of "us-ascii". For example:

```
5496 Content-Type: text/xml; charset=ISO-2022-JP  
5497  
5498
```

5499 Also, when an application/xml entity is used, the charset parameter is optional, and registry
5500 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which
5501 directly address this contingency.

5502 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

5503 **F.3 NLS And Storing of RegistryObject**

5504 This section provides NLS guidelines on how a registry should store RegistryObject instances.

5505 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple
5506 locales. Thus there is no language or character set associated with a specific RegistryObject
5507 instance.

5508 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows.
5509 Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in

5510 RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed
5511 as LocalizedString sub-elements within the XML element representing the I18N capable
5512 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a
5513 specific locale. Each LocalizedString element has a charset and lang attribute as well as a value
5514 attribute of type string.

5515 **F.3.1 Character Set of *LocalizedString***

5516 The character set used by a locale specific String (LocalizedString) is defined by the charset
5517 attribute. It is highly recommended to use UTF-8 or UTF-16 for maximum interoperability.

5518 **F.3.2 Language Information of *LocalizedString***

5519 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

5520 **F.4 NLS And Storing of Repository Items**

5521 This section provides NLS guidelines on how a registry should store repository items.
5522 While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is
5523 always associated with a single repository item. The repository item may be in a single locale or
5524 may be in multiple locales. This specification does not specify the repository item.

5525 **F.4.1 Character Set of Repository Items**

5526 The MIME Content-Type mime header for the mime multi-part containing the repository
5527 item MAY contain a "charset" attribute that specifies the character set used by the repository
5528 item. For example:
5529

```
5530 Content-Type: text/xml; charset="UTF-8"
```

5532 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset
5533 of a repository item must be preserved as it is originally specified in the transaction.

5534 **F.4.2 Language information of repository item**

5535 The Content-language mime header for the mime bodypart containing the repository item may
5536 specify the language for a locale specific repository item. The value of the Content-language
5537 mime header property must conform to [RFC 1766].

5538 This document currently specifies only the method of sending the information of character set
5539 and language, and how it is stored in a registry. However, the language information may be used
5540 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a
5541 language negotiation procedure, like registry client is asking a favorite language for messages
5542 from registry services, could be another functionality for the future revision of this document.

5543 **Appendix G Registry Profile**

5544 Every registry must support exactly one Registry Profile. The Registry Profile is an XML
5545 document that describes the capabilities of the registry. The profile document must conform to
5546 the RegistryProfile element as described in the Registry Services Interface schema defined in
5547 Appendix B. The registry must make the RegistryProfile accessible over HTTP protocol via a
5548 URL. The URL must conform to the pattern:

5549 <http://<base url>/registryProfile>

5550

5551 13 References

- 5552 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 5553 [ebRIM] ebXML Registry Information Model version 3.0
- 5554 <http://www.oasis-open.org/committees/regrep/documents/3.0/specs/ebRIM.pdf>
- 5555 [ebRIM Schema] ebXML Registry Information Model Schema
- 5556 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>
- 5557 [ebBPSS] ebXML Business Process Specification Schema
- 5558 <http://www.ebxml.org/specs>
- 5559 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 5560 <http://www.ebxml.org/specs/>
- 5561 [ebMS] ebXML Messaging Service Specification, Version 1.0
- 5562 <http://www.ebxml.org/specs/>
- 5563 [XPT] XML Path Language (XPath) Version 1.0
- 5564 <http://www.w3.org/TR/xpath>
- 5565 [SQL] Structured Query Language (FIPS PUB 127-2)
- 5566 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 5567 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 5568 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 5569 [IANA] IANA (Internet Assigned Numbers Authority).
- 5570 Official Names for Character Sets, ed. Keld Simonsen et al.
- 5571 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 5572 [RESTThesis] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral Dissertation, University of California, Irvine. 2000.
- 5573 <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- 5574 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
- 5575 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
- 5576 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 5577 [RFC 2119] IETF (Internet Engineering Task Force). RFC 2119
- 5578 [RFC 2130] IETF (Internet Engineering Task Force). RFC 2130
- 5579 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
- 5580 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
- 5581 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 5582 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
- 5583 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
- 5584 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 5585 [RFC2616] RFC 2616:
- 5586 Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1*. 1999.
- 5587 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 5588 [RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
- 5589 Internet Security Glossary, ed. R. Shirey. May 2000.
- 5590 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html>
- 5591 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
- 5592 XML Media Types, ed. M. Murata. 2001.
- 5593 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
- 5594

- 5595 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
5596 <http://www.w3.org/TR/REC-xml>
- 5597 [UUID] DCE 128 bit Universal Unique Identifier
5598 http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh_20
5599 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
- 5600 [WSDL] W3C Note. Web Services Description Language (WSDL) 1.1
5601 <http://www.w3.org/TR/wsdl>
- 5602 [SOAP11] W3C Note. Simple Object Access Protocol, May 2000,
5603 <http://www.w3.org/TR/SOAP>
- 5604 [SOAPAt] W3C Note: SOAP with Attachments, Dec 2000,
5605 <http://www.w3.org/TR/SOAP-attachments>
- 5606 [XMLDSIG] XML-Signature Syntax and Processing,
5607 <http://www.w3.org/TR/2001/PR-xmlsig-core-20010820/>

5608 **14 Disclaimer**

5609 The views and specification expressed in this document are those of the authors and are not
5610 necessarily those of their employers. The authors and their employers specifically disclaim
5611 responsibility for any problems arising from correct or incorrect implementation or use of this
5612 design.

5613 15 Contact Information

5614 Team Leader

5615 Name: Kathryn R. Breininger
5616 Company: The Boeing Company
5617 Street: P.O. Box 3707 MC 62-LC
5618 City, State, Postal Code: Seattle, WA 98124-2207
5619 Country: USA
5620 Phone: 425-965-0182
5621 Email: kathryn.r.breininger@boeing.com

5622

5623 Editor

5624 Name: Farrukh S. Najmi
5625 Company: Sun Microsystems
5626 Street: 1 Network Dr., MS BUR02-302
5627 City, State, Postal Code: Burlington, MA, 01803-0902
5628 Country: USA
5629 Phone: (781) 442-0703
5630 Email: farrukh.najmi@sun.com

5631

5632 Technical Editor

5633 Name: Farrukh S. Najmi
5634 Company: Sun Microsystems
5635 Street: 1 Network Dr., MS BUR02-302
5636 City, State, Postal Code: Burlington, MA, 01803-0902
5637 Country: USA
5638 Phone: (781) 442-0703
5639 Email: farrukh.najmi@sun.com

5640

5641 16 Copyright Statement

5642 Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

5643 **Copyright (C) The Organization for the Advancement of Structured Information**
5644 **Standards [OASIS], 2002. All Rights Reserved.**

5645 This document and translations of it may be copied and furnished to others, and derivative works
5646 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
5647 published and distributed, in whole or in part, without restriction of any kind, provided that the
5648 above copyright notice and this paragraph are included on all such copies and derivative works.
5649 However, this document itself may not be modified in any way, such as by removing the
5650 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
5651 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
5652 Property Rights document must be followed, or as required to translate it into languages other
5653 than English.

5654 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
5655 successors or assigns.

5656 This document and the information contained herein is provided on an "AS IS" basis and OASIS
5657 **DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT**
5658 **LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN**
5659 **WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF**
5660 **MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

5661 OASIS takes no position regarding the validity or scope of any intellectual property or other
5662 rights that might be claimed to pertain to the implementation or use of the technology described
5663 in this document or the extent to which any license under such rights might or might not be
5664 available; neither does it represent that it has made any effort to identify any such rights.
5665 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
5666 at the OASIS website. Copies of claims of rights made available for publication and any
5667 assurances of licenses to be made available, or the result of an attempt made to obtain a general
5668 license or permission for the use of such proprietary rights by implementers or users of this
5669 specification, can be obtained from the OASIS Executive Director.

5670 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
5671 applications, or other proprietary rights which may cover technology that may be required to
5672 implement this specification. Please address the information to the OASIS Executive Director.

5673 **17 Notes**

5674 These notes are here to not lose the thought and will be merged into the spec later as issues get
5675 resolved.

- 5676 ○ CBD:IndexContentResponse: how to handle any non-composed metadata such as
5677 ExternalIdentifier, Package etc.?
- 5678 ○ CBD: Need to fix schema for virtual document processed by default XML Indexer.
- 5679 ○ CBD: How to track generated metadata separate from submitted metadata (and to do so
5680 efficiently)? Should we also log which indexer and index file created it?

5681