



---

Creating A Single Global Electronic Market

1

# **OASIS/ebXML Registry Services Specification v2.0**

## **-Approved OASIS Standard**

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

April 2002

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

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

4 This document is an Approved OASIS Standard - April 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/2.0/specs/ebRS.pdf>

9 ***Latest version:***

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

## 11 **2 OASIS/ebXML Registry Technical Committee**

12 Prior to submission to the OASIS membership, the OASIS/ebXML Registry Technical  
13 Committee approved this document in its current form in December 2001. At the time of this  
14 approval the following were members of the OASIS/ebXML Registry Technical Committee.

15 Kathryn Breininger, Boeing  
16 Lisa Carnahan, NIST  
17 Joseph M. Chiusano, LMI  
18 Suresh Damodaran, Sterling Commerce  
19 Mike DeNicola, Fujitsu  
20 Anne Fischer, Drummond Group, Inc.  
21 Sally Fuger, AIAG  
22 Jong Kim, InnoDigital  
23 Kyu-Chul Lee, Chungnam National University  
24 Joel Munter, Intel  
25 Farrukh Najmi, Sun Microsystems  
26 Joel Neu, Vitria Technologies  
27 Sanjay Patil, IONA  
28 Neal Smith, Chevron  
29 Nikola Stojanovic, Encoda Systems, Inc.  
30 Prasad Yendluri, webmethods  
31 Yutaka Yoshida, Sun Microsystems

### 32 **2.1 Contributors**

33 The following persons contributed to the content of this document, but are not voting members  
34 of the OASIS/ebXML Registry Technical Committee.

35 Len Gallagher, NIST  
36 Sekhar Vajjhala, Sun Microsystems

37	<b>Table of Contents</b>	
38	<b>1 Status of this Document</b>	<b>3</b>
39	<b>2 OASIS/ebXML Registry Technical Committee</b>	<b>4</b>
40	2.1 Contributors	4
41	<b>Table of Contents</b>	<b>5</b>
42	<b>Table of Figures</b>	<b>9</b>
43	<b>Table of Tables</b>	<b>10</b>
44	<b>3 Introduction</b>	<b>11</b>
45	3.1 Summary of Contents of Document	11
46	3.2 General Conventions	11
47	3.3 Audience	11
48	<b>4 Design Objectives</b>	<b>12</b>
49	4.1 Goals	12
50	4.2 Caveats and Assumptions	12
51	<b>5 System Overview</b>	<b>13</b>
52	5.1 What The ebXML Registry Does	13
53	5.2 How The ebXML Registry Works	13
54	5.2.1 Schema Documents Are Submitted	13
55	5.2.2 Business Process Documents Are Submitted	13
56	5.2.3 Seller's Collaboration Protocol Profile Is Submitted	13
57	5.2.4 Buyer Discovers The Seller	13
58	5.2.5 CPA Is Established	14
59	5.3 Registry Users	14
60	5.4 Where the Registry Services May Be Implemented	15
61	5.5 Implementation Conformance	15
62	5.5.1 Conformance as an ebXML Registry	16
63	5.5.2 Conformance as an ebXML Registry Client	16
64	<b>6 ebXML Registry Architecture</b>	<b>17</b>
65	6.1 Registry Service Described	17
66	6.2 Abstract Registry Service	18
67	6.3 Concrete Registry Services	18
68	6.3.1 SOAP Binding	19
69	6.3.2 ebXML Message Service Binding	20
70	6.4 LifeCycleManager Interface	21
71	6.5 QueryManager Interface	22
72	6.6 Registry Clients	22
73	6.6.1 Registry Client Described	22
74	6.6.2 Registry Communication Bootstrapping	23
75	6.6.3 RegistryClient Interface	24
76	6.6.4 Registry Response	24
77	6.7 Interoperability Requirements	24
78	6.7.1 Client Interoperability	24
79	6.7.2 Inter-Registry Cooperation	25
80	<b>7 Life Cycle Management Service</b>	<b>26</b>
81	7.1 Life Cycle of a Repository Item	26
82	7.2 RegistryObject Attributes	26

83	7.3	The Submit Objects Protocol .....	27
84	7.3.1	Universally Unique ID Generation .....	27
85	7.3.2	ID Attribute And Object References.....	28
86	7.3.3	Audit Trail.....	28
87	7.3.4	Submitting Organization.....	28
88	7.3.5	Error Handling .....	28
89	7.3.6	Sample SubmitObjectsRequest.....	29
90	7.4	The Update Objects Protocol .....	32
91	7.4.1	Audit Trail.....	33
92	7.4.2	Submitting Organization.....	33
93	7.4.3	Error Handling .....	33
94	7.5	The Add Slots Protocol.....	34
95	7.6	The Remove Slots Protocol .....	34
96	7.7	The Approve Objects Protocol.....	35
97	7.7.1	Audit Trail.....	35
98	7.7.2	Submitting Organization.....	36
99	7.7.3	Error Handling .....	36
100	7.8	The Deprecate Objects Protocol .....	36
101	7.8.1	Audit Trail.....	37
102	7.8.2	Submitting Organization.....	37
103	7.8.3	Error Handling .....	37
104	7.9	The Remove Objects Protocol .....	38
105	7.9.1	Deletion Scope DeleteRepositoryItemOnly.....	38
106	7.9.2	Deletion Scope DeleteAll .....	38
107	7.9.3	Error Handling .....	39
108	<b>8</b>	<b>Query Management Service.....</b>	<b>40</b>
109	8.1	Ad Hoc Query Request/Response.....	40
110	8.1.1	Query Response Options.....	41
111	8.2	Filter Query Support .....	42
112	8.2.1	FilterQuery.....	44
113	8.2.2	RegistryObjectQuery .....	46
114	8.2.3	RegistryEntryQuery .....	59
115	8.2.4	AssociationQuery.....	62
116	8.2.5	AuditableEventQuery .....	64
117	8.2.6	ClassificationQuery.....	67
118	8.2.7	ClassificationNodeQuery.....	69
119	8.2.8	ClassificationSchemeQuery.....	74
120	8.2.9	RegistryPackageQuery.....	75
121	8.2.10	ExtrinsicObjectQuery .....	77
122	8.2.11	OrganizationQuery.....	78
123	8.2.12	ServiceQuery.....	82
124	8.2.13	Registry Filters.....	84
125	8.2.14	XML Clause Constraint Representation .....	88
126	8.3	SQL Query Support .....	92
127	8.3.1	SQL Query Syntax Binding To [ebRIM].....	92
128	8.3.2	Semantic Constraints On Query Syntax .....	94
129	8.3.3	SQL Query Results .....	94
130	8.3.4	Simple Metadata Based Queries .....	95

131	8.3.5 RegistryObject Queries.....	95
132	8.3.6 RegistryEntry Queries.....	95
133	8.3.7 Classification Queries.....	95
134	8.3.8 Association Queries.....	97
135	8.3.9 Package Queries.....	97
136	8.3.10 ExternalLink Queries.....	98
137	8.3.11 Audit Trail Queries.....	98
138	8.4 Content Retrieval.....	98
139	8.4.1 Identification Of Content Payloads.....	98
140	8.4.2 GetContentResponse Message Structure.....	99
141	<b>9 Registry Security.....</b>	<b>100</b>
142	9.1 Security Concerns.....	100
143	9.2 Integrity of Registry Content.....	100
144	9.2.1 Message Payload Signature.....	100
145	9.2.2 Payload Signature Requirements.....	101
146	9.3 Authentication.....	103
147	9.3.1 Message Header Signature.....	103
148	9.4 Key Distribution and KeyInfo Element.....	105
149	9.5 Confidentiality.....	106
150	9.5.1 On-the-wire Message Confidentiality.....	106
151	9.5.2 Confidentiality of Registry Content.....	106
152	9.6 Authorization.....	106
153	9.6.1 Actions.....	107
154	9.7 Access Control.....	107
155	<b>Appendix A Web Service Architecture.....</b>	<b>109</b>
156	A.1 Registry Service Abstract Specification.....	109
157	A.2 Registry Service SOAP Binding.....	109
158	<b>Appendix B ebXML Registry Schema Definitions.....</b>	<b>110</b>
159	B.1 RIM Schema.....	110
160	B.2 Query Schema.....	110
161	B.3 Registry Services Interface Schema.....	110
162	B.4 Examples of Instance Documents.....	110
163	<b>Appendix C Interpretation of UML Diagrams.....</b>	<b>111</b>
164	C.1 UML Class Diagram.....	111
165	C.2 UML Sequence Diagram.....	111
166	<b>Appendix D SQL Query.....</b>	<b>112</b>
167	D.1 SQL Query Syntax Specification.....	112
168	D.2 Non-Normative BNF for Query Syntax Grammar.....	112
169	D.3 Relational Schema For SQL Queries.....	114
170	<b>Appendix E Non-normative Content Based Ad Hoc Queries.....</b>	<b>115</b>
171	E.1 Automatic Classification of XML Content.....	115
172	E.2 Index Definition.....	115
173	E.3 Example Of Index Definition.....	115
174	E.4 Proposed XML Definition.....	116
175	E.5 Example of Automatic Classification.....	116
176	<b>Appendix F Security Implementation Guideline.....</b>	<b>117</b>
177	F.1 Security Concerns.....	117

178	F.2	Authentication.....	118
179	F.3	Authorization .....	118
180	F.4	Registry Bootstrap .....	118
181	F.5	Content Submission – Client Responsibility .....	118
182	F.6	Content Submission – Registry Responsibility .....	119
183	F.7	Content Delete/Deprecate – Client Responsibility .....	119
184	F.8	Content Delete/Deprecate – Registry Responsibility .....	119
185	F.9	Using ds:KeyInfo Field.....	119
186	<b>Appendix G</b>	<b>Native Language Support (NLS).....</b>	<b>121</b>
187	G.1	Definitions.....	121
188	G.1.1	Coded Character Set (CCS):.....	121
189	G.1.2	Character Encoding Scheme (CES):.....	121
190	G.1.3	Character Set (charset):.....	121
191	G.2	NLS And Request / Response Messages .....	121
192	G.3	NLS And Storing of RegistryObject.....	121
193	G.3.1	Character Set of <i>LocalizedString</i> .....	122
194	G.3.2	Language Information of <i>LocalizedString</i> .....	122
195	G.4	NLS And Storing of Repository Items.....	122
196	G.4.1	Character Set of Repository Items .....	122
197	G.4.2	Language information of repository item .....	122
198	<b>Appendix H</b>	<b>Registry Profile.....</b>	<b>123</b>
199	<b>10</b>	<b>References .....</b>	<b>124</b>
200	<b>11</b>	<b>Disclaimer .....</b>	<b>126</b>
201	<b>12</b>	<b>Contact Information .....</b>	<b>127</b>
202	<b>13</b>	<b>Copyright Statement.....</b>	<b>128</b>
203			



## 204 **Table of Figures**

205	Figure 1: Actor Relationships .....	15
206	Figure 2: ebXML Registry Service Architecture .....	17
207	Figure 3: The Abstract ebXML Registry Service .....	18
208	Figure 4: A Concrete ebXML Registry Service.....	19
209	Figure 5: Registry Architecture Supports Flexible Topologies .....	23
210	Figure 6: Life Cycle of a Repository Item .....	26
211	Figure 7: Submit Objects Sequence Diagram .....	27
212	Figure 8: Update Objects Sequence Diagram .....	33
213	Figure 9: Add Slots Sequence Diagram .....	34
214	Figure 10: Remove Slots Sequence Diagram.....	35
215	Figure 11: Approve Objects Sequence Diagram .....	35
216	Figure 12: Deprecate Objects Sequence Diagram.....	37
217	Figure 13: Remove Objects Sequence Diagram .....	39
218	Figure 14: Submit Ad Hoc Query Sequence Diagram.....	41
219	Figure 15: Example ebRIM Binding.....	43
220	Figure 16: ebRIM Binding for RegistryObjectQuery .....	46
221	Figure 17: ebRIM Binding for RegistryEntryQuery .....	59
222	Figure 18: ebRIM Binding for AssociationQuery .....	62
223	Figure 19: ebRIM Binding for AuditableEventQuery .....	64
224	Figure 20: ebRIM Binding for ClassificationQuery .....	67
225	Figure 21: ebRIM Binding for ClassificationNodeQuery.....	69
226	Figure 22: ebRIM Binding for ClassificationSchemeQuery.....	74
227	Figure 23: ebRIM Binding for RegistryPackageQuery .....	75
228	Figure 24: ebRIM Binding for ExtrinsicObjectQuery .....	77
229	Figure 25: ebRIM Binding for OrganizationQuery .....	79
230	Figure 26: ebRIM Binding for ServiceQuery .....	83
231	Figure 27: The Clause Structure.....	88
232		

**233 Table of Tables**

234	Table 1: Registry Users .....	14
235	Table 2: LifeCycle Manager Summary .....	21
236	Table 3: Query Manager .....	22
237	Table 4: RegistryClient Summary .....	24
238	Table 5 Submit Objects Error Handling .....	28
239	Table 6: Update Objects Error Handling .....	34
240	Table 7: Approve Objects Error Handling .....	36
241	Table 8: Deprecate Objects Error Handling .....	37
242	Table 9: Remove Objects Error Handling .....	39
243	Table 10: Path Filter Expressions for Use Cases .....	72
244	Table 11: Default Access Control Policies .....	107
245		

## 246 **3 Introduction**

### 247 **3.1 Summary of Contents of Document**

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

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

### 253 **3.2 General Conventions**

254 The following conventions are used throughout this document:

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

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

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

262 Capitalized Italic words are defined in the ebXML Glossary.

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

### 266 **3.3 Audience**

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

- 268 • Implementers of ebXML Registry Services
- 269 • Implementers of ebXML Registry Clients

#### 270 **Related Documents**

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

- 272 a) *ebXML Registry Information Model* [ebRIM]
- 273 b) *ebXML Message Service Specification* [ebMS]
- 274 c) *ebXML Business Process Specification Schema* [ebBPSS]
- 275 d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

## 276 **4 Design Objectives**

### 277 **4.1 Goals**

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

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

### 283 **4.2 Caveats and Assumptions**

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

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

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

## 294 **5 System Overview**

### 295 **5.1 What The ebXML Registry Does**

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

### 300 **5.2 How The ebXML Registry Works**

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

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

#### 310 **5.2.1 Schema Documents Are Submitted**

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

#### 314 **5.2.2 Business Process Documents Are Submitted**

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

#### 318 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

#### 323 **5.2.4 Buyer Discovers The Seller**

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

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

### 329 5.2.5 CPA Is Established

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

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

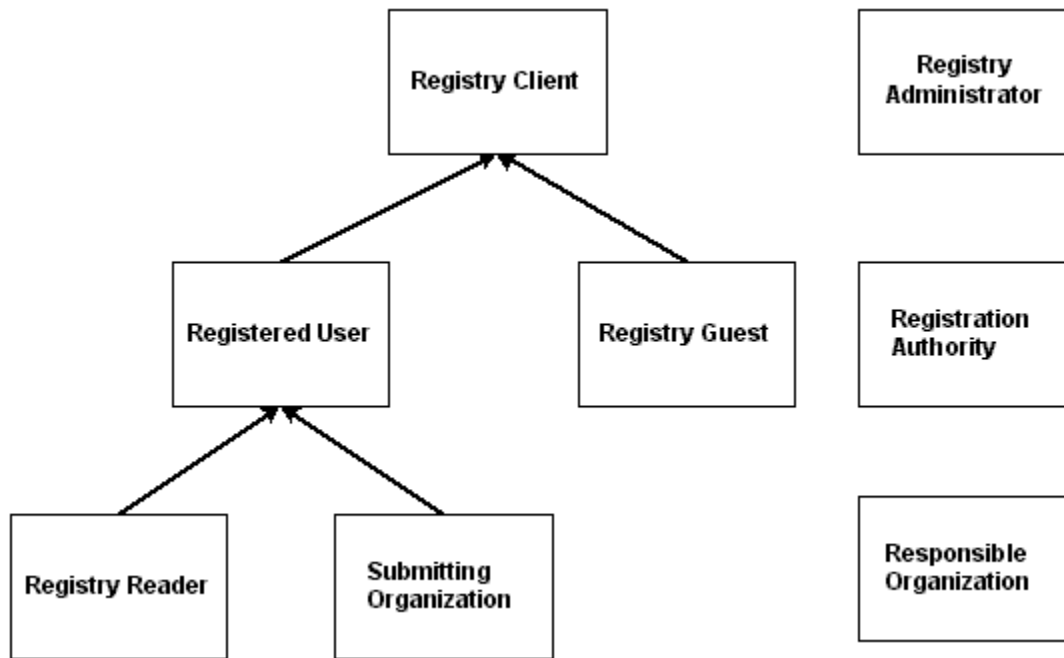
### 336 5.3 Registry Users

337 We describe the actors who use the registry from the point of view of security and analyze the  
 338 security concerns of the registry below. This analysis leads up to the security requirements for  
 339 version 2.0. Some of the actors are defined in Section 9.7. Note that the same entity may  
 340 represent different actors. For example, a Registration Authority and Registry Administrator may  
 341 have the same identity.

342 **Table 1: Registry Users**

<b>Actor</b>	<b>Function</b>	<b>ISO/IEC 11179</b>	<b>Comments</b>
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 MAY have the same identity as SO

		(RO)	
Registry Client	Registered User or Registered Guest		



343  
344

Figure 1: Actor Relationships

345 Note:

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

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

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

349 A Registry Administrator and Registration Authority are the same.

### 350 5.4 Where the Registry Services May Be Implemented

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

### 353 5.5 Implementation Conformance

354 An implementation is a *conforming* ebXML Registry if the implementation meets the conditions  
355 in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the  
356 implementation meets the conditions in Section 5.5.2. An implementation is a conforming  
357 ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to  
358 the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming  
359 ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and  
360 Registry Client.

### 361 **5.5.1 Conformance as an ebXML Registry**

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

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

### 368 **5.5.2 Conformance as an ebXML Registry Client**

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

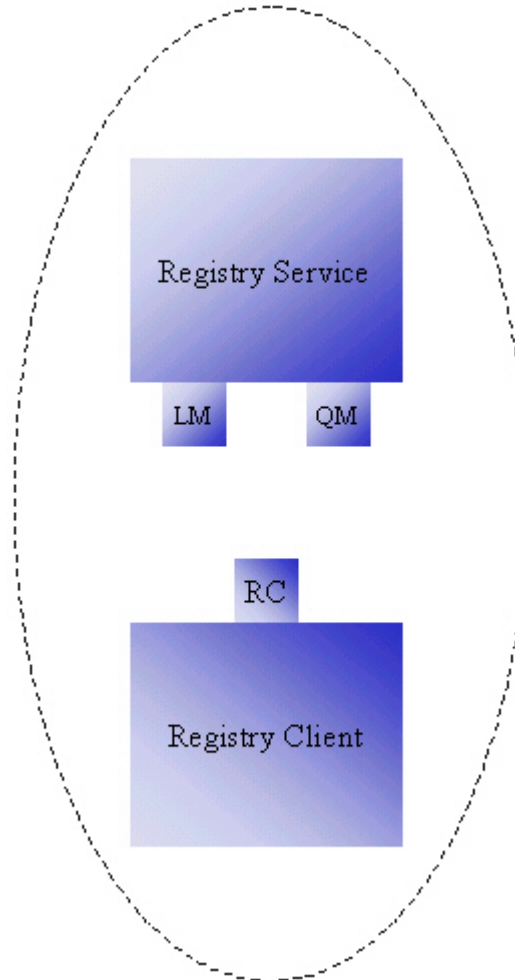
- 371 1. Supports the ebXML CPA and bootstrapping process.
- 372 2. Supports the syntax and the semantics of the Registry Client Interfaces.
- 373 3. Supports the defined ebXML Error Message DTD.
- 374 4. Supports the defined ebXML Registry Schema (Appendix B).

375



## 376 6 ebXML Registry Architecture

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



380  
 381

**Figure 2: ebXML Registry Service Architecture**

### 382 6.1 Registry Service Described

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

- 386 • A Life Cycle Management interface that provides a collection of methods for managing  
 387 objects within the Registry.
- 388 • A Query Management Interface that controls the discovery and retrieval of information from  
 389 the Registry.

390 A registry client program utilizes the services of the registry by invoking methods on one of the  
 391 above interfaces defined by the Registry Service. This specification defines the interfaces  
 392 exposed by the Registry Service (Sections 6.4 and 6.5) as well as the interface for the Registry  
 393 Client (Section 6.6).

## 394 6.2 Abstract Registry Service

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

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

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

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



405  
406

Figure 3: The Abstract ebXML Registry Service

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

## 409 6.3 Concrete Registry Services

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

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

414 This specification describes two concrete bindings for the abstract registry service:

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

417 A registry may implement one or both of the concrete bindings for the abstract registry service as  
418 shown in Figure 4.

419

---

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

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

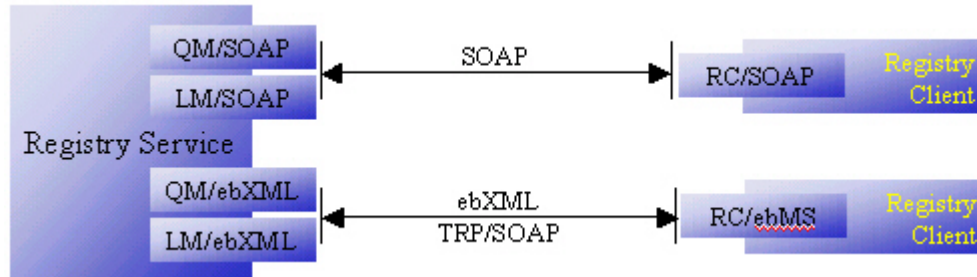


Figure 4: A Concrete ebXML Registry Service

420

421

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

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

## 429 6.3.1 SOAP Binding

### 430 6.3.1.1 WSDL Terminology Primer

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

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

### 443 6.3.1.2 Concrete Binding for SOAP

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

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

```
453 <service name = "RegistryService">
```

454

```

455     <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
456         <soap:address location = "http://your_URL_to_your_QueryManager"/>
457     </port>
458
459     <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
460         <soap:address location = "http://your_URL_to_your_QueryManager"/>
461     </port>
462 </service>
463

```

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

## 466 6.3.2 ebXML Message Service Binding

### 467 6.3.2.1 Service and Action Elements

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

- 470 • The value of the Service element in the MessageHeader is an ebXML Registry Service  
471 interface name (e.g., “LifeCycleManager”). The type attribute of the Service element should  
472 have a value of “ebXMLRegistry”.
- 473 • The value of the Action element in the MessageHeader is an ebXML Registry Service  
474 method name (e.g., “submitObjects”).

```

475
476 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
477 <eb:Action>submitObjects</eb:Action>
478

```

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

### 482 6.3.2.2 Synchronous and Asynchronous Responses

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

#### 484 Asynchronous response

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

- 488 • MessageHeader;
- 489 • RegistryResponse element with empty content (e.g., **NO** AdHocQueryResponse);  
490 – status attribute with value **Unavailable**.

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

- 495 • MessageHeader;
- 496 • RegistryResponse element including;  
497 – Status attribute (Success, Failure);

498 – Optional RegistryErrorList.

499 **Synchronous response**

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

- 502 • MessageHeader;
- 503 • RegistryResponse element including:
  - 504 – Status attribute (Success, Failure);
  - 505 – Optional RegistryErrorList.

506 **6.3.2.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

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

518 **6.4 LifeCycleManager Interface**

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

524 **Table 2: LifeCycle Manager Summary**

<b>Method Summary of LifeCycleManager</b>	
RegistryResponse	<a href="#">approveObjects</a> ( <a href="#">ApproveObjectsRequest</a> req) Approves one or more previously submitted objects.
RegistryResponse	<a href="#">deprecateObjects</a> ( <a href="#">DeprecateObjectsRequest</a> req) Deprecates one or more previously submitted objects.
RegistryResponse	<a href="#">removeObjects</a> ( <a href="#">RemoveObjectsRequest</a> req) Removes one or more previously submitted objects from the Registry.
RegistryResponse	<a href="#">submitObjects</a> ( <a href="#">SubmitObjectsRequest</a> req) Submits one or more objects and possibly related metadata such as Associations and Classifications.
RegistryResponse	<a href="#">updateObjects</a> ( <a href="#">UpdateObjectsRequest</a> req)

	Updates one or more previously submitted objects.
RegistryResponse	<a href="#">addSlots</a> ( <a href="#">AddSlotsRequest</a> req) Add slots to one or more registry entries.
RegistryResponse	<a href="#">removeSlots</a> ( <a href="#">RemoveSlotsRequest</a> req) Remove specified slots from one or more registry entries.

## 525 6.5 QueryManager Interface

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

529 **Table 3: Query Manager**

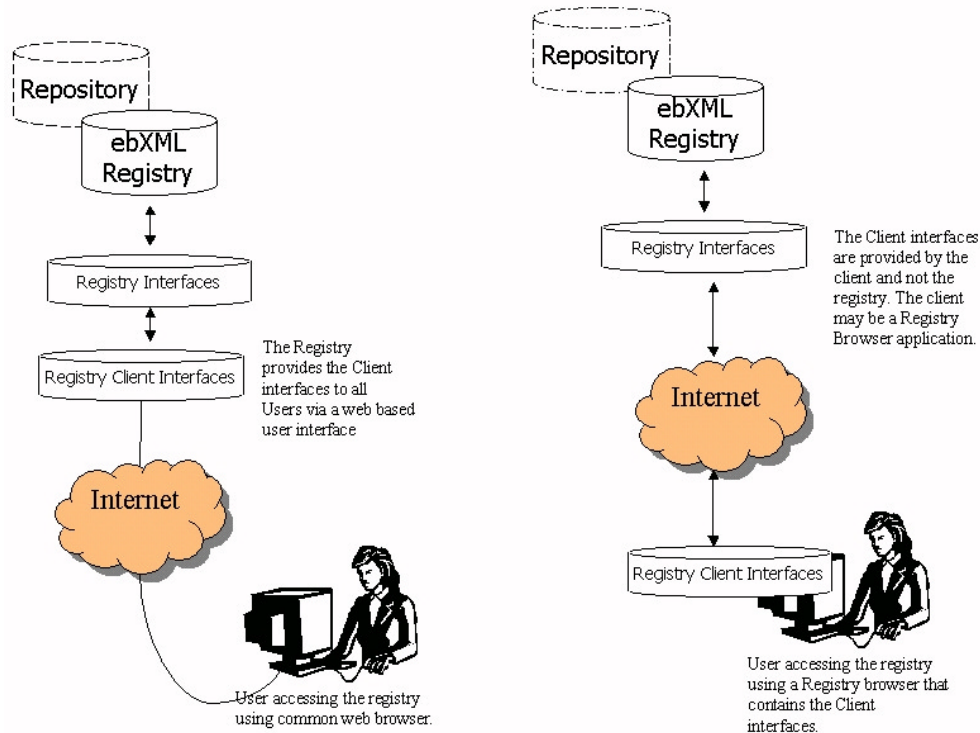
<b>Method Summary of QueryManager</b>	
RegistryResponse	<a href="#">submitAdhocQuery</a> ( <a href="#">AdhocQueryRequest</a> req) Submit an ad hoc query request.

## 530 6.6 Registry Clients

### 531 6.6.1 Registry Client Described

532 The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts  
533 the two possible topologies supported by the registry architecture with respect to the Registry  
534 and Registry Clients. The picture on the left side shows the scenario where the Registry provides  
535 a web based “thin client” application for accessing the Registry that is available to the user using  
536 a common web browser. In this scenario the Registry Client interfaces reside across the Internet  
537 and are local to the Registry from the user’s view. The picture on the right side shows the  
538 scenario where the user is using a “fat client” Registry Browser application to access the registry.  
539 In this scenario the Registry Client interfaces reside within the Registry Browser tool and are  
540 local to the Registry from the user’s view. The Registry Client interfaces communicate with the  
541 Registry over the Internet in this scenario.

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



547  
548

Figure 5: Registry Architecture Supports Flexible Topologies

## 549 6.6.2 Registry Communication Bootstrapping

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

- 556 • In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available  
557 in a WSDL description for the registry. This WSDL conforms to the template WSDL  
558 description in Appendix A.1. This WSDL description may be discovered in a public registry  
559 such as UDDI.
- 560 • In case of ebMS binding, the information exchange between the client and the registry may  
561 be accomplished in a registry specific manner, which may involve establishing a CPA  
562 between the client and the registry. Once the information exchange has occurred the Registry  
563 and the client will have addressing information (e.g. URLs) for the other party.

### 564 6.6.2.1 Communication Bootstrapping for SOAP Binding

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

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

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

#### 574 **6.6.2.2 Communication Bootstrapping for ebXML Message Service**

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

### 585 **6.6.3 RegistryClient Interface**

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

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

592 **Table 4: RegistryClient Summary**

<b>Method Summary of RegistryClient</b>	
void	<b>onResponse</b> ( <a href="#">RegistryResponse</a> resp) Notifies client of the response sent by registry to previously submitted request.

### 593 **6.6.4 Registry Response**

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

## 596 **6.7 Interoperability Requirements**

### 597 **6.7.1 Client Interoperability**

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



## 602 **6.7.2 Inter-Registry Cooperation**

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

606 Examples include:

- 607 • An ebXML Registry that serves as a registry of ebXML Registries.
- 608 • A non-ebXML Registry that serves as a registry of ebXML Registries.
- 609 • Cooperative ebXML Registries, where multiple ebXML registries register with each other in  
610 order to form a federation.

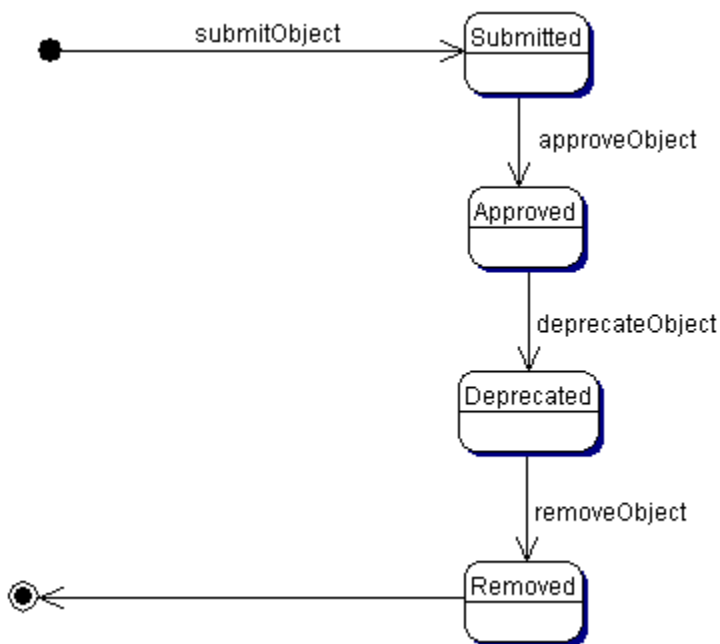
## 611 7 Life Cycle Management Service

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

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

### 621 7.1 Life Cycle of a Repository Item

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



626  
627

Figure 6: Life Cycle of a Repository Item

### 628 7.2 RegistryObject Attributes

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

## 634 7.3 The Submit Objects Protocol

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



638  
 639

**Figure 7: Submit Objects Sequence Diagram**

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

641 The SubmitObjectRequest message includes a LeafRegistryObjectList element.

642 The LeafRegistryObjectList element specifies one or more ExtrinsicObjects or other  
 643 RegistryEntries such as Classifications, Associations, ExternalLinks, or Packages.

644 An ExtrinsicObject element provides required metadata about the content being submitted to the  
 645 Registry as defined by [ebRIM]. Note that these standard ExtrinsicObject attributes are separate  
 646 from the repository item itself, thus allowing the ebXML Registry to catalog objects of any  
 647 object type.

### 648 7.3.1 Universally Unique ID Generation

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

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

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

659 If the client does not supply an id for a submitted object then the registry must generate a

660 universally unique id. Whether the client generates the id or whether the registry generates it, it  
661 must be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

### 662 **7.3.2 ID Attribute And Object References**

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

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

### 678 **7.3.3 Audit Trail**

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

### 681 **7.3.4 Submitting Organization**

682 The RS must create an Association of type SubmitterOf between the submitting organization and  
683 each RegistryObject created via a SubmitObjects request. (Submitting organization is  
684 determined from the organization attribute of the User who submits a SubmitObjects request.)

### 685 **7.3.5 Error Handling**

686 A SubmitObjects request is atomic and either succeeds or fails in total. In the event of success,  
687 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event  
688 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In  
689 the event of an immediate response for an asynchronous request, the registry sends a  
690 RegistryResponse with a status of "Unavailable" back to the client. Failure occurs when one or  
691 more Error conditions are raised in the processing of the submitted objects. Warning messages  
692 do not result in failure of the request. The following business rules apply:

693 **Table 5 Submit Objects Error Handling**

<b>Business Rule</b>	<b>Applies To</b>	<b>Error/Warning</b>
ID not unique	All Classes	Error
Not authorized	All Classes	Error

Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion are set by the RS, and ignored if supplied.	All Classes	Warning

### 694 7.3.6 Sample SubmitObjectsRequest

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

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

```

701 <?xml version = "1.0" encoding = "UTF-8"?>
702 <SubmitObjectsRequest
703   xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
704   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
705   xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
706 spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
707 file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
708   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
709   xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
710 >
711   <rim:LeafRegistryObjectList>
712
713     <!--
714     The following 3 objects package specified ExtrinsicObject in specified
715     RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
716     being submitted
717     -->
718     <!--
719     -->
720
721     <rim:RegistryPackage id = "acmePackage1" >
722       <rim:Name>
723         <rim:LocalizedString value = "RegistryPackage #1"/>
724       </rim:Name>
725       <rim:Description>
726         <rim:LocalizedString value = "ACME's package #1"/>
727       </rim:Description>
728     </rim:RegistryPackage>
729
730     <rim:ExtrinsicObject id = "acmeCPP1" >
731       <rim:Name>
732         <rim:LocalizedString value = "Widget Profile" />
733       </rim:Name>
734       <rim:Description>
735         <rim:LocalizedString value = "ACME's profile for selling widgets" />
736       </rim:Description>
737     </rim:ExtrinsicObject>
738
739     <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
740 = "acmePackage1" targetObject = "acmeCPP1" />
741
742     <!--
743     The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
744     where the RegistryPackage is being submitted and the ExtrinsicObject is
745     already in registry
746     -->

```

```

747
748 <rim:RegistryPackage id = "acmePackage2" >
749   <rim:Name>
750     <rim:LocalizedString value = "RegistryPackage #2"/>
751   </rim:Name>
752   <rim:Description>
753     <rim:LocalizedString value = "ACME's package #2"/>
754   </rim:Description>
755 </rim:RegistryPackage>
756
757 <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
758
759 <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
760 sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
761
762 <!--
763   The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
764   where the RegistryPackage and the ExtrinsicObject are already in registry
765   -->
766
767 <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
768 <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
769
770 <!-- id is unspecified implying that registry must create a uuid for this object -->
771
772 <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
773 123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
774
775 <!--
776   The following 3 objects externally link specified ExtrinsicObject using
777   specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
778   are being submitted
779   -->
780
781 <rim:ExternalLink id = "acmeLink1" >
782   <rim:Name>
783     <rim:LocalizedString value = "Link #1"/>
784   </rim:Name>
785   <rim:Description>
786     <rim:LocalizedString value = "ACME's Link #1"/>
787   </rim:Description>
788 </rim:ExternalLink>
789
790 <rim:ExtrinsicObject id = "acmeCPP2" >
791   <rim:Name>
792     <rim:LocalizedString value = "Sprockets Profile" />
793   </rim:Name>
794   <rim:Description>
795     <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
796   </rim:Description>
797 </rim:ExtrinsicObject>
798
799 <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
800 sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
801
802 <!--
803   The following 2 objects externally link specified ExtrinsicObject using specified
804   ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
805   is already in registry. Note that the targetObject points to an ObjectRef in a
806   previous line
807   -->
808
809 <rim:ExternalLink id = "acmeLink2">
810   <rim:Name>
811     <rim:LocalizedString value = "Link #2"/>
812   </rim:Name>
813   <rim:Description>
814     <rim:LocalizedString value = "ACME's Link #2"/>
815   </rim:Description>
816 </rim:ExternalLink>
817

```

```

818 <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
819 "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
820 123456789012"/>
821
822 <!--
823 The following 3 objects externally identify specified ExtrinsicObject using specified
824 ExternalIdentifier, where the ExternalIdentifier is being submitted and the
825 ExtrinsicObject is already in registry. Note that the targetObject points to an
826 ObjectRef in a previous line
827 -->
828
829 <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
830 <rim:Name>
831 <rim:LocalizedString value = "DUNS"/>
832 </rim:Name>
833
834 <rim:Description>
835 <rim:LocalizedString value = "This is the DUNS scheme"/>
836 </rim:Description>
837 </rim:ClassificationScheme>
838
839 <rim:ExternalIdentifier id = "acmeDUNSID" identificationScheme="DUNS-id" value =
840 "13456789012">
841 <rim:Name>
842 <rim:LocalizedString value = "DUNS" />
843 </rim:Name>
844 <rim:Description>
845 <rim:LocalizedString value = "DUNS ID for ACME"/>
846 </rim:Description>
847 </rim:ExternalIdentifier>
848
849 <rim:Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc" associationType =
850 "ExternallyIdentifies" sourceObject = "acmeDUNSID" targetObject = "urn:uuid:a2345678-1234-1234-
851 123456789012"/>
852
853 <!--
854 The following show submission of a brand new classification scheme in its entirety
855 -->
856 <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
857 <rim:Name>
858 <rim:LocalizedString value = "Geography"/>
859 </rim:Name>
860
861 <rim:Description>
862 <rim:LocalizedString value = "This is a sample Geography scheme"/>
863 </rim:Description>
864
865 <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
866 "NorthAmerica" >
867 <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
868 "UnitedStates" />
869 <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
870 </rim:ClassificationNode>
871
872 <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
873 <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
874 <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
875 </rim:ClassificationNode>
876 </rim:ClassificationNode>
877 </rim:ClassificationScheme>
878
879
880 <!--
881 The following show submission of a Automotive sub-tree of ClassificationNodes that
882 gets added to an existing classification scheme named 'Industry'
883 that is already in the registry
884 -->
885
886 <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
887 <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
888 123456789012">
889 <rim:Name>
890 <rim:LocalizedString value = "Automotive" />

```

```

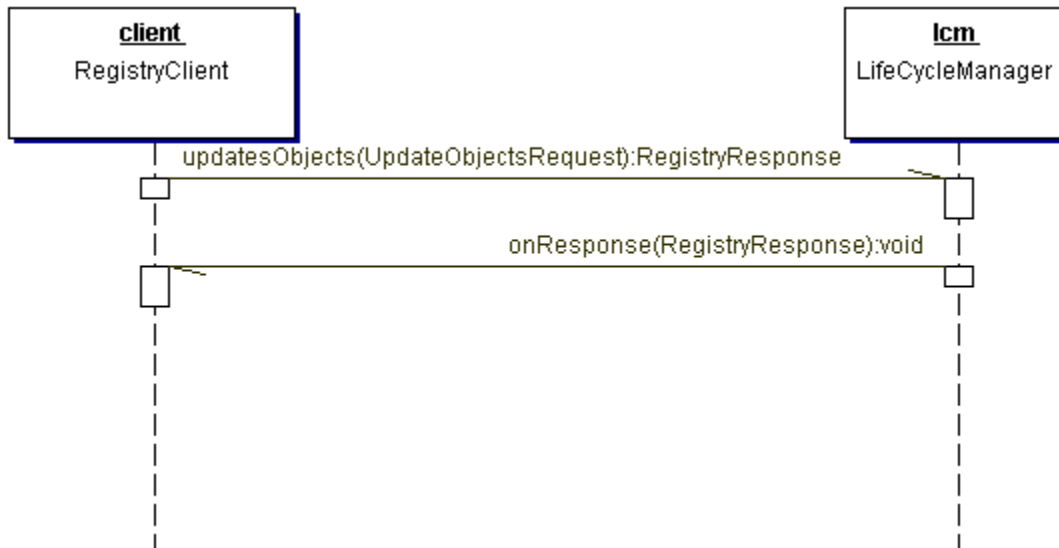
891     </rim:Name>
892     <rim:Description>
893       <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
894     </rim:Description>
895   </rim:ClassificationNode>
896
897   <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
898     <rim:Name>
899       <rim:LocalizedString value = "Parts Supplier" />
900     </rim:Name>
901     <rim:Description>
902       <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
903     </rim:Description>
904   </rim:ClassificationNode>
905
906   <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
907     <rim:Name>
908       <rim:LocalizedString value = "Engine Supplier" />
909     </rim:Name>
910     <rim:Description>
911       <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
912     </rim:Description>
913   </rim:ClassificationNode>
914
915   <!--
916     The following show submission of 2 Classifications of an object that is already in
917     the registry using 2 ClassificationNodes. One ClassificationNode
918     is being submitted in this request (Japan) while the other is already in the registry.
919   -->
920
921   <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
922 1234-123456789012" classificationNode = "Japan-id">
923     <rim:Description>
924       <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
925     </rim:Description>
926   </rim:Classification>
927
928   <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
929 "urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
930 123456789012">
931     <rim:Description>
932       <rim:LocalizedString value = "Classifies object using a node in the registry" />
933     </rim:Description>
934   </rim:Classification>
935
936   <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
937 </rim:LeafRegistryObjectList>
938 </SubmitObjectsRequest>
939

```

## 940 7.4 The Update Objects Protocol

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





944  
945

**Figure 8: Update Objects Sequence Diagram**

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

#### 952 **7.4.1 Audit Trail**

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

#### 955 **7.4.2 Submitting Organization**

956 The RS must maintain an Association of type SubmitterOf between the submitting organization  
957 and each RegistryObject updated via an UpdateObjects request. If an UpdateObjects request is  
958 accepted from a different submitting organization, then the RS must delete the original  
959 association object and create a new one. Of course, the AccessControlPolicy may prohibit this  
960 sort of update in the first place. (Submitting organization is determined from the organization  
961 attribute of the User who submits an UpdateObjects request.)

#### 962 **7.4.3 Error Handling**

963 An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success,  
964 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event  
965 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In  
966 the event of an immediate response for an asynchronous request, the registry sends a  
967 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or  
968 more Error conditions are raised in the processing of the updated objects. Warning messages do  
969 not result in failure of the request. The following business rules apply:

970

**Table 6: Update Objects Error Handling**

<b>Business Rule</b>	<b>Applies To</b>	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	All Classes	Error
Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion cannot be changed via the UpdateObjects protocol, ignored if supplied.	All Classes	Warning
RegistryEntries with stability = "Stable" should not be updated.	All Classes	Warning

971 **7.5 The Add Slots Protocol**

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



975  
976

**Figure 9: Add Slots Sequence Diagram**

977 In the event of success, the registry sends a RegistryResponse with a status of “success” back to  
 978 the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure”  
 979 back to the client.

980 **7.6 The Remove Slots Protocol**

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



983  
984

Figure 10: Remove Slots Sequence Diagram

985 **7.7 The Approve Objects Protocol**

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



990  
991

Figure 11: Approve Objects Sequence Diagram

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

993 **7.7.1 Audit Trail**

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

## 996 7.7.2 Submitting Organization

997 The RS must maintain an Association of type SubmitterOf between the submitting organization  
 998 and each RegistryObject updated via an ApproveObjects request. If an ApproveObjects request  
 999 is accepted from a different submitting organization, then the RS must delete the original  
 1000 association object and create a new one. Of course, the AccessControlPolicy may prohibit this  
 1001 sort of ApproveObjects request in the first place. (Submitting organization is determined from  
 1002 the organization attribute of the User who submits an ApproveObjects request.)

## 1003 7.7.3 Error Handling

1004 An ApproveObjects request is atomic and either succeeds or fails in total. In the event of success,  
 1005 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event  
 1006 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In  
 1007 the event of an immediate response for an asynchronous request, the registry sends a  
 1008 RegistryResponse with a status of "Unavailable" back to the client. Failure occurs when one or  
 1009 more Error conditions are raised in the processing of the object reference list. Warning messages  
 1010 do not result in failure of the request. The following business rules apply:

1011 **Table 7: Approve Objects Error Handling**

<b>Business Rule</b>	<b>Applies To</b>	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "approved".	All Classes other than RegistryEntry classes	Error
Object status is already "Approved".	RegistryEntry Classes	Warning

## 1012 7.8 The Deprecate Objects Protocol

1013 This section describes the protocol of the Registry Service that allows a client to deprecate one or  
 1014 more previously submitted repository items using the LifeCycleManager. Once an object is  
 1015 deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that  
 1016 object can be submitted. However, existing references to a deprecated object continue to function  
 1017 normally.



1018  
1019

**Figure 12: Deprecate Objects Sequence Diagram**

1020 For details on the schema for the business documents shown in this process refer to Appendix B.

1021 **7.8.1 Audit Trail**

1022 The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject  
1023 deprecated via a Deprecate Objects request.

1024 **7.8.2 Submitting Organization**

1025 The RS must maintain an Association of type SubmitterOf between the submitting organization  
1026 and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request  
1027 is accepted from a different submitting organization, then the RS must delete the original  
1028 association object and create a new one. Of course, the AccessControlPolicy may prohibit this  
1029 sort of Deprecate Objects request in the first place. (Submitting organization is determined from  
1030 the organization attribute of the User who submits a Deprecate Objects request.)

1031 **7.8.3 Error Handling**

1032 A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of  
1033 success, the registry sends a RegistryResponse with a status of “Success” back to the client. In  
1034 the event of failure, the registry sends a RegistryResponse with a status of “Failure” back to the  
1035 client. In the event of an immediate response for an asynchronous request, the registry sends a  
1036 RegistryResponse with a status of “Unavailable” back to the client. Failure occurs when one or  
1037 more Error conditions are raised in the processing of the object reference list. Warning messages  
1038 do not result in failure of the request. The following business rules apply:

1039

**Table 8: Deprecate Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistrvEntrv	Error

	Classes	
Only RegistryEntries may be "deprecated".	All Classes other than RegistryEntry classes	Error
Object status is already "Deprecated".	RegistryEntry Classes	Warning

1040 **7.9 The Remove Objects Protocol**

1041 This section describes the protocol of the Registry Service that allows a client to remove one or  
 1042 more RegistryObject instances and/or repository items using the LifeCycleManager.

1043 The RemoveObjectsRequest message is sent by a client to remove RegistryObject instances  
 1044 and/or repository items. The RemoveObjectsRequest element includes an XML attribute called  
 1045 deletionScope which is an enumeration that can have the values as defined by the following  
 1046 sections.

1047 **7.9.1 Deletion Scope DeleteRepositoryItemOnly**

1048 This deletionScope specifies that the request should delete the repository items for the specified  
 1049 registry entries but not delete the specified registry entries. This is useful in keeping references to  
 1050 the registry entries valid.

1051 **7.9.2 Deletion Scope DeleteAll**

1052 This deletionScope specifies that the request should delete both the RegistryObject and the  
 1053 repository item for the specified registry entries. Only if all references (e.g. Associations,  
 1054 Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject  
 1055 then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to  
 1056 remove a RegistryObject while it still has references raises an error condition:  
 1057 InvalidRequestError.

1058 The remove object protocol is expressed in UML notation as described in Appendix C.



1059

1060 **Figure 13: Remove Objects Sequence Diagram**

1061 For details on the schema for the business documents shown in this process refer to Appendix B.

1062 **7.9.3 Error Handling**

1063 A Remove Objects request is atomic and either succeeds or fails in total. In the event of success,  
 1064 the registry sends a RegistryResponse with a status of “Success” back to the client. In the event  
 1065 of failure, the registry sends a RegistryResponse with a status of “Failure” back to the client. In  
 1066 the event of an immediate response for an asynchronous request, the registry sends a  
 1067 RegistryResponse with a status of “Unavailable” back to the client. Failure occurs when one or  
 1068 more Error conditions are raised in the processing of the object reference list. Warning messages  
 1069 do not result in failure of the request. The following business rules apply:

1070 **Table 9: Remove Objects Error Handling**

<b>Business Rule</b>	<b>Applies To</b>	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	RegistryObject Classes	Error

1071

## 1072 **8 Query Management Service**

1073 This section describes the capabilities of the Registry Service that allow a client  
1074 (QueryManagerClient) to search for or query different kind of registry objects in the ebXML  
1075 Registry using the QueryManager interface of the Registry. The Registry supports the following  
1076 query capabilities:

- 1077 • Filter Query
- 1078 • SQL Query

1079 The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry  
1080 implementation. The SQL Query mechanism is an optional feature and MAY be provided by a  
1081 registry implementation. However, if a vendor provides an SQL query capability to an ebXML  
1082 Registry it SHALL conform to this document. As such this capability is a normative yet optional  
1083 capability.

1084 In a future version of this specification, the W3C XQuery syntax may be considered as another  
1085 query syntax.

1086 The Registry will hold a self-describing capability profile that identifies all supported  
1087 AdhocQuery options. This profile is described in Appendix H.

### 1088 **8.1 Ad Hoc Query Request/Response**

1089 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The  
1090 AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry  
1091 query mechanisms.

1092 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously  
1093 back to the client. The AdhocQueryResponse returns a collection of objects whose element type  
1094 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects  
1095 representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate  
1096 classes in [ebRIM] such as RegistryObject and RegistryEntry.

1097 Any errors in the query request messages are indicated in the corresponding query response  
1098 message.





1099

1100

**Figure 14: Submit Ad Hoc Query Sequence Diagram**

1101 For details on the schema for the business documents shown in this process refer to Appendix  
 1102 B.2.

### 1103 Definition

1104

```

1105 <element name="AdhocQueryRequest">
1106   <complexType>
1107     <sequence>
1108       <element ref="tns:ResponseOption" minOccurs="1" maxOccurs="1" />
1109       <choice minOccurs="1" maxOccurs="1">
1110         <element ref="tns:FilterQuery" />
1111         <element ref="tns:SQLQuery" />
1112       </choice>
1113     </sequence>
1114   </complexType>
1115 </element>
1116
1117 <element name="AdhocQueryResponse">
1118   <complexType>
1119     <choice minOccurs="1" maxOccurs="1">
1120       <element ref="tns:FilterQueryResult" />
1121       <element ref="tns:SQLQueryResult" />
1122     </choice>
1123   </complexType>
1124 </element>
1125
  
```

## 1126 8.1.1 Query Response Options

### 1127 Purpose

1128 A QueryManagerClient may specify what an ad hoc query must return within an  
 1129 AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest.  
 1130 ResponseOption element has an attribute "returnType" and its values are:

- 1131 • ObjectRef - This option specifies that the AdhocQueryResponse may contain a collection of  
1132 ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return  
1133 just the identifiers of the registry objects.
- 1134 • RegistryObject - This option specifies that the AdhocQueryResponse may contain a  
1135 collection of RegistryObject XML elements as defined in [ebRIM Schema]. In this case all  
1136 attributes of the registry objects are returned (objectType, name, description, ...) in addition  
1137 to id attribute.
- 1138 • RegistryEntry - This option specifies that the AdhocQueryResponse may contain a collection  
1139 of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which  
1140 correspond to RegistryEntry or RegistryObject attributes.
- 1141 • LeafClass - This option specifies that the AdhocQueryResponse may contain a collection of  
1142 XML elements that correspond to leaf classes as defined in [ebRIM Schema].
- 1143 • LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse may  
1144 contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema]  
1145 accompanied with their repository items or RegistryEntry or RegistryObject and their  
1146 attributes. Linking of ExtrinsicObject and its repository item is done via contentURI as  
1147 explained in Section 8.4 -Content Retrieval.

1148 ResponseOption element also has an attribute "returnComposedObjects". It specifies whether or  
1149 not the whole hierarchy of composed objects are returned with the registry objects.

1150 If "returnType" is higher then the RegistryObject option, then the highest option that satisfies the  
1151 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return  
1152 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass  
1153 option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then  
1154 RegistryObject metadata will be returned.

#### 1155 Definition

```

1156
1157 <complexType name="ResponseOptionType">
1158   <attribute name="returnType" default="RegistryObject">
1159     <simpleType>
1160       <restriction base="NMTOKEN">
1161         <enumeration value="ObjectRef" />
1162         <enumeration value="RegistryObject" />
1163         <enumeration value="RegistryEntry" />
1164         <enumeration value="LeafClass" />
1165         <enumeration value="LeafClassWithRepositoryItem" />
1166       </restriction>
1167     </simpleType>
1168   </attribute>
1169   <attribute name="returnComposedObjects" type="boolean" default="false" />
1170 </complexType>
1171 <element name="ResponseOption" type="tns:ResponseOptionType" />
1172

```

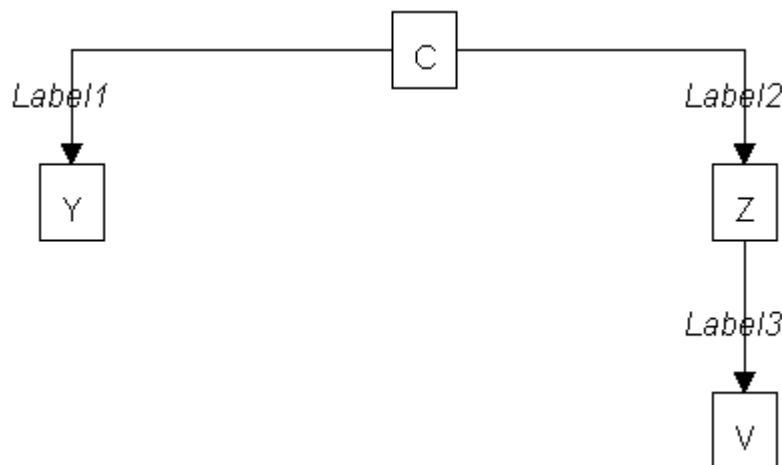
## 1173 8.2 Filter Query Support

1174 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML  
1175 conforming Registry implementation. Each query alternative is directed against a single class  
1176 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter  
1177 queries depending on which classes are queried on.

- 1178 • Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic  
 1179 queries that might return different subclasses of the class that is queried on. The result of  
 1180 such a query is a set of XML elements that correspond to instances of any class that satisfies  
 1181 the responseOption defined previously in Section 8.1.1. An example might be that  
 1182 RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances  
 1183 that satisfy the query. This implies that response might return XML elements that correspond  
 1184 to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- 1185 • Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact  
 1186 traversals of these classes. Responses to these queries are accordingly constrained.

1187 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an  
 1188 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified  
 1189 herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified  
 1190 in Section 8.1.

1191 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of  
 1192 classes derived from a single class and its associations with other classes as defined by ebRIM.  
 1193 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For  
 1194 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a  
 1195 class that is associated with Z. The ebRIM Binding for C might be as in Figure 15



1196  
 1197 **Figure 15: Example ebRIM Binding**

1198 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and  
 1199 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to  
 1200 which ebRIM association is intended. The name of the query is determined by the root class, i.e.  
 1201 this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances  
 1202 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are  
 1203 limited to instances that are linked to their parent node by the identified association.

1204 Each FilterQuery alternative depends upon one or more class filters, where a class filter is a  
 1205 restricted predicate clause over the attributes of a single class. Class methods that are defined in  
 1206 ebRIM and that return simple types constitute “visible attributes” that are valid choices for  
 1207 predicate clauses. Names of those attributes will be same as name of the corresponding method  
 1208 just without the prefix ‘get’. For example, in case of “getLevelNumber” method the  
 1209 corresponding visible attribute is “levelNumber”. The supported class filters are specified in  
 1210 Section 8.2.13 and the supported predicate clauses are defined in Section 8.2.14. A FilterQuery

1211 will be composed of elements that traverse the tree to determine which branches satisfy the  
 1212 designated class filters, and the query result will be the set of instances that support such a  
 1213 branch.

1214 In the above example, the CQuery element will have three subelements, one a CFilter on the C  
 1215 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on  
 1216 the Y class to eliminate branches from C to Y where the target of the association does not satisfy  
 1217 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third  
 1218 element is called a branch element because it allows class filters on each class along the path  
 1219 from C to V. In general, a branch element will have subelements that are themselves class filters,  
 1220 other branch elements, or a full-blown query on the class in the path.

1221 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one  
 1222 branch, filter or query element on Y is allowed. However, if the association is one-to-many, then  
 1223 multiple branch, filter or query elements are allowed. This allows one to specify that an instance  
 1224 of C must have associations with multiple instances of Y before the instance of C is said to  
 1225 satisfy the branch element.

1226 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be  
 1227 stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then  
 1228 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax  
 1229 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their  
 1230 respective superclass queries. Structures of XML elements that match the ebRIM classes are  
 1231 explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in  
 1232 ebRIM whenever possible.

1233 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual  
 1234 hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify  
 1235 the effect of that binding on query semantics.

## 1236 8.2.1 FilterQuery

### 1237 Purpose

1238 To identify a set of queries that traverse specific registry class. Each alternative assumes a  
 1239 specific binding to ebRIM. The status is a success indication or a collection of warnings and/or  
 1240 exceptions.

### 1241 Definition

```

1242 <element name="FilterQuery">
1243   <complexType>
1244     <choice minOccurs="1" maxOccurs="1">
1245       <element ref="tns:RegistryObjectQuery" />
1246       <element ref="tns:RegistryEntryQuery" />
1247       <element ref="tns:AssociationQuery" />
1248       <element ref="tns:AuditableEventQuery" />
1249       <element ref="tns:ClassificationQuery" />
1250       <element ref="tns:ClassificationNodeQuery" />
1251       <element ref="tns:ClassificationSchemeQuery" />
1252       <element ref="tns:RegistryPackageQuery" />
1253       <element ref="tns:ExtrinsicObjectQuery" />
1254       <element ref="tns:OrganizationQuery" />
1255       <element ref="tns:ServiceQuery" />
1256     </choice>
  
```

```

1257     </choice>
1258     </complexType>
1259 </element>
1260
1261 <element name="FilterQueryResult">
1262     <complexType>
1263         <choice minOccurs="1" maxOccurs="1">
1264             <element ref="tns:RegistryObjectQueryResult" />
1265             <element ref="tns:RegistryEntryQueryResult" />
1266             <element ref="tns:AssociationQueryResult" />
1267             <element ref="tns:AuditableEventQueryResult" />
1268             <element ref="tns:ClassificationQueryResult" />
1269             <element ref="tns:ClassificationNodeQueryResult" />
1270             <element ref="tns:ClassificationSchemeQueryResult" />
1271             <element ref="tns:RegistryPackageQueryResult" />
1272             <element ref="tns:ExtrinsicObjectQueryResult" />
1273             <element ref="tns:OrganizationQueryResult" />
1274             <element ref="tns:ServiceQueryResult" />
1275         </choice>
1276     </complexType>
1277 </element>
1278

```

## 1279 Semantic Rules

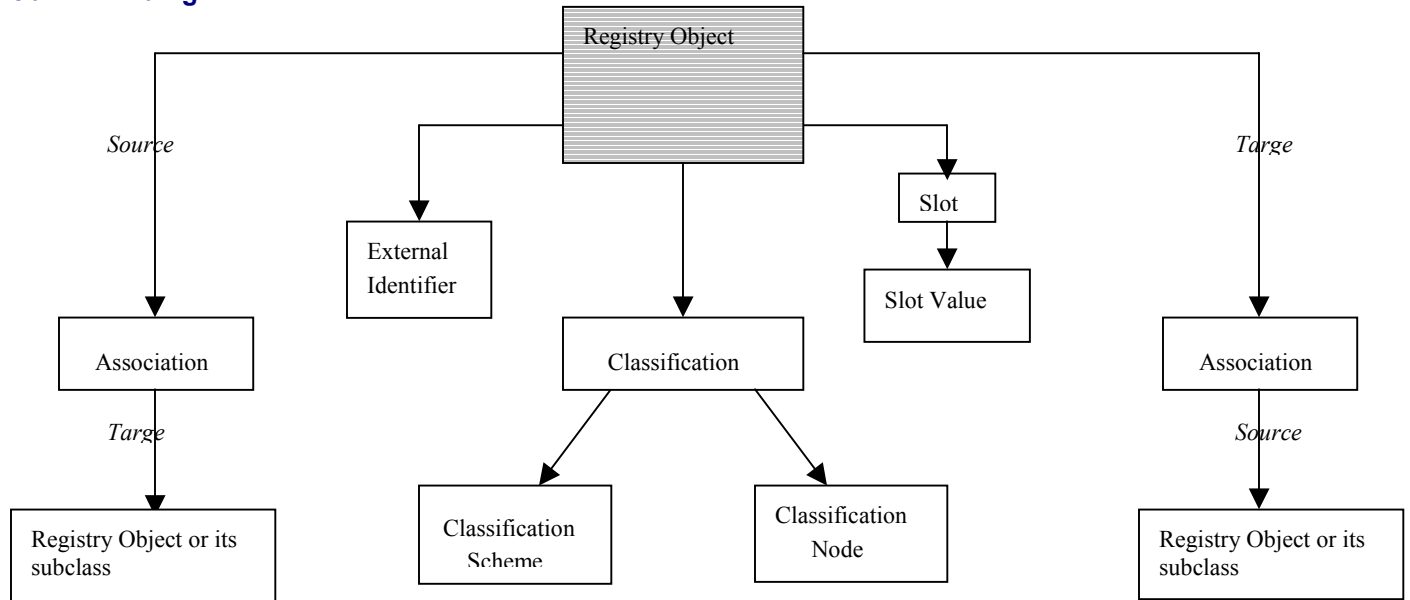
- 1280 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 1281 2. Semantic rules specify the procedure for implementing the evaluation of Filter Queries.  
1282 Implementations do not necessarily have to follow the same procedure provided that the  
1283 same effect is achieved.
- 1284 3. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.  
1285 Each XML attribute carries a value derived from the value of an attribute specified in the  
1286 Registry Information Model [ebRIM Schema].
- 1287 4. For each FilterQuery subelement there is only one corresponding FilterQueryResult  
1288 subelement that must be returned as a response. Class name of the FilterQueryResult  
1289 subelement has to match the class name of the FilterQuery subelement.
- 1290 5. If a Filter, Branch or Query element for a class has no sub-elements then every persistent  
1291 instance of that class satisfies the Filter, Branch or Query.
- 1292 6. If an error condition is raised during any part of the execution of a FilterQuery, then the  
1293 status attribute of the XML RegistryResult is set to "failure" and no AdHocQueryResult  
1294 element is returned; instead, a RegistryErrorList element must be returned with its  
1295 highestSeverity element set to "error". At least one of the RegistryError elements in the  
1296 RegistryErrorList will have its severity attribute set to "error".
- 1297 7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of  
1298 the XML RegistryResult is set to "success" and an appropriate FilterQueryResult element  
1299 must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of  
1300 the RegistryErrorList is set to "warning" and the severity attribute of each RegistryError is  
1301 set to "warning".

1302 **8.2.2 RegistryObjectQuery**

1303 **Purpose**

1304 To identify a set of registry object instances as the result of a query over selected registry  
 1305 metadata.

1306 **ebRIM Binding**



1307 **Figure 16: ebRIM Binding for RegistryObjectQuery**

1308 **Definition**

```

1309 <complexType name="RegistryObjectQueryType">
1310   <sequence>
1311     <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
1312     <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
1313     <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
1314     <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
1315     <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
1316     <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
1317     <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
1318     <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1319     <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1320   </sequence>
1321 </complexType>
1322 <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
1323
1324 <complexType name="LeafRegistryObjectListType">
1325   <choice minOccurs="0" maxOccurs="unbounded">
1326     <element ref="tns:ObjectRef" />
1327     <element ref="tns:Association" />
1328     <element ref="tns:AuditableEvent" />
1329     <element ref="tns:Classification" />
1330     <element ref="tns:ClassificationNode" />
1331     <element ref="tns:ClassificationScheme" />
1332     <element ref="tns:ExternalIdentifier" />
1333     <element ref="tns:ExternalLink" />
1334     <element ref="tns:ExtrinsicObject" />
    
```

```

1335     <element ref="tns:Organization" />
1336     <element ref="tns:RegistryPackage" />
1337     <element ref="tns:Service" />
1338     <element ref="tns:ServiceBinding" />
1339     <element ref="tns:SpecificationLink" />
1340     <element ref="tns:User" />
1341 </choice>
1342 </complexType>
1343
1344 <complexType name="RegistryObjectListType">
1345   <complexContent>
1346     <extension base="tns:LeafRegistryObjectListType">
1347       <choice minOccurs="0" maxOccurs="unbounded">
1348         <element ref="tns:RegistryEntry" />
1349         <element ref="tns:RegistryObject" />
1350       </choice>
1351     </extension>
1352   </complexContent>
1353 </complexType>
1354 <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1355
1356 <complexType name="InternationalStringBranchType">
1357   <sequence>
1358     <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1359   </sequence>
1360 </complexType>
1361
1362 <complexType name="AssociationBranchType">
1363   <sequence>
1364     <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1365     <choice minOccurs="0" maxOccurs="1">
1366       <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1367       <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1368       <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1369       <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1370       <element ref="tns:AssociationQuery" minOccurs="0" maxOccurs="1" />
1371       <element ref="tns:ClassificationQuery" minOccurs="0" maxOccurs="1" />
1372       <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1373       <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1374       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1375       <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1376       <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1377       <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1378       <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1379       <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
1380       <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1381       <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1382     </choice>
1383   </sequence>
1384 </complexType>
1385 <element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
1386 <element name="TargetAssociationBranch" type="tns:AssociationBranchType" />
1387
1388 <element name="ClassifiedByBranch">
1389   <complexType>
1390     <sequence>
1391       <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />

```

```

1392     <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1393     <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1394   </sequence>
1395 </complexType>
1396 </element>
1397
1398 <element name="SlotBranch">
1399   <complexType>
1400     <sequence>
1401       <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
1402       <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
1403     </sequence>
1404   </complexType>
1405 </element>
1406
1407 <element name="UserBranch">
1408   <complexType>
1409     <sequence>
1410       <element ref="tns:UserFilter" minOccurs="0" maxOccurs="1"/>
1411       <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1"/>
1412       <element ref="tns:TelephoneFilter" minOccurs="0" maxOccurs="unbounded"/>
1413       <element ref="tns:EmailAddressFilter" minOccurs="0" maxOccurs="unbounded"/>
1414       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1"/>
1415     </sequence>
1416   </complexType>
1417 </element>
1418
1419 <complexType name="ServiceBindingBranchType">
1420   <sequence>
1421     <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
1422     <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
1423     <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
1424   </sequence>
1425 </complexType>
1426 <element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType" />
1427 <element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType" />
1428
1429 <element name="SpecificationLinkBranch">
1430   <complexType>
1431     <sequence>
1432       <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
1433       <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1434       <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1435     </sequence>
1436   </complexType>
1437 </element>
1438

```

### 1439 **Semantic Rules**

- 1440 1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The  
 1441 following steps will eliminate instances in RO that do not satisfy the conditions of the  
 1442 specified filters.
- 1443 a) If RO is empty then go to number 2 below.



- 1444 b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a  
1445 registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from  
1446 RO. If RO is empty then continue to the next numbered rule.
- 1447 c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise,  
1448 let x be a remaining registry object in RO. If x is not linked to at least one  
1449 ExternalIdentifier instance, then remove x from RO; otherwise, treat each  
1450 ExternalIdentifierFilter element separately as follows: Let EI be the set of  
1451 ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If  
1452 EI is empty, then remove x from RO. If RO is empty then continue to the next numbered  
1453 rule.
- 1454 d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a  
1455 remaining registry object in RO. If x doesn't have an auditable event that satisfy  
1456 AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is  
1457 empty then continue to the next numbered rule.
- 1458 e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining  
1459 registry object in RO. If x does not have a name then remove x from RO. If RO is empty  
1460 then continue to the next numbered rule; otherwise treat NameBranch as follows: If any  
1461 LocalizedStringFilter that is specified is not satisfied by at least one of the  
1462 LocalizedStrings that constitute the name of the registry object then remove x from RO.  
1463 If RO is empty then continue to the next numbered rule.
- 1464 f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a  
1465 remaining registry object in RO. If x does not have a name then remove x from RO. If  
1466 RO is empty then continue to the next numbered rule; otherwise treat DescriptionBranch  
1467 as follows: If any LocalizedStringFilter that is specified is not satisfied by some of the  
1468 LocalizedStrings that constitute the description of the registry object then remove x from  
1469 RO. If RO is empty then continue to the next numbered rule.
- 1470 g) If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x  
1471 be a remaining registry object in RO. If x is not the classifiedObject of at least one  
1472 Classification instance, then remove x from RO; otherwise, treat each  
1473 ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified  
1474 within the ClassifiedByBranch, then let CL be the set of all Classification instances that  
1475 have x as the classifiedObject; otherwise, let CL be the set of Classification instances that  
1476 satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then  
1477 remove x from RO and continue to the next numbered rule. Otherwise, if CL is not  
1478 empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of  
1479 remaining Classification instances in CL whose defining classification scheme satisfies  
1480 the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and  
1481 continue to the next numbered rule. Otherwise, if CL remains not empty, and if a  
1482 ClassificationNodeQuery is specified, then replace CL by the set of remaining  
1483 Classification instances in CL for which a classification node exists and for which that  
1484 classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then  
1485 remove x from RO. If RO is empty then continue to the next numbered rule.

- 1486 h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a  
1487 remaining registry object in RO. If x is not linked to at least one Slot instance, then  
1488 remove x from RO. If RO is empty then continue to the next numbered rule; otherwise,  
1489 treat each SlotBranch element separately as follows: If a SlotFilter is not specified within  
1490 the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the  
1491 set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,  
1492 then remove x from RO and continue to the next numbered rule. Otherwise, if SL  
1493 remains not empty, and if a SlotValueFilter is specified, replace SL by the set of  
1494 remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is  
1495 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
- 1496 i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,  
1497 let x be a remaining registry object in RO. If x is not the source object of at least one  
1498 Association instance, then remove x from RO. If RO is empty then continue to the next  
1499 numbered rule; otherwise, treat each SourceAssociationBranch element separately as  
1500 follows:  
1501 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be  
1502 the set of all Association instances that have x as a source object; otherwise, let AF be the  
1503 set of Association instances that satisfy the AssociationFilter and have x as the source  
1504 object. If AF is empty, then remove x from RO.  
1505  
1506 If RO is empty then continue to the next numbered rule.  
1507  
1508 If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT  
1509 be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target  
1510 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty  
1511 then continue to the next numbered rule.  
1512  
1513 If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let  
1514 ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and  
1515 are the target object of some element of AF. If ROT is empty, then remove x from RO. If  
1516 RO is empty then continue to the next numbered rule.  
1517  
1518 If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT  
1519 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the  
1520 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
1521 empty then continue to the next numbered rule.  
1522  
1523 If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT  
1524 be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the  
1525 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
1526 empty then continue to the next numbered rule.  
1527  
1528 If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let  
1529 ROT be the set of ClassificationScheme instances that satisfy the  
1530 ClassificationSchemeQuery and are the target object of some element of AF. If ROT is  
1531 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562

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

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

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

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

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

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

1563 If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set  
1564 of User instances that are the target object of some element of AF. If ROT is empty, then  
1565 remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the  
1566 member of ROT. If a UserFilter element is specified within the UserBranch, and if u does  
1567 not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from  
1568 RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter  
1569 element is specified within the UserBranch, and if the postal address of u does not satisfy  
1570 that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is  
1571 empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are  
1572 specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied  
1573 by at least one of the telephone numbers of u then remove u from ROT. If ROT is empty,  
1574 then remove x from RO. If RO is empty then continue to the next numbered rule. If an  
1575 OrganizationQuery element is specified within the UserBranch, then let o be the  
1576 Organization instance that is identified by the organization that u is affiliated with. If o  
1577 doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROT.  
1578 If ROT is empty, then remove x from RO. If RO is empty then continue to the next  
1579 numbered rule.

1580

1581 If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT  
1582 be the set of Classification instances that satisfy the ClassificationQuery and are the  
1583 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is  
1584 empty then continue to the next numbered rule (Rule 2).

1585

1586 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let  
1587 ROT be the set of ServiceBinding instances that are the target object of some element of  
1588 AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next  
1589 numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is  
1590 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then  
1591 remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then  
1592 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the  
1593 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as  
1594 follows:

1595 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link  
1596 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
1597 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
1598 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then  
1599 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
1600 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl  
1601 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:  
1602 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is  
1603 not a specification link for at least one registry object in RO, then remove sl from SL. If  
1604 SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO  
1605 is empty then continue to the next numbered rule. If a RegistryEntryQuery element is  
1606 specified within the SpecificationLinkBranch then let sl be a remaining specification link  
1607 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the  
1608 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least  
1609 one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from  
1610 ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next  
1611 numbered rule. If a ServiceBindingTargetBranch is specified within the  
1612 ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy  
1613 the ServiceBindingTargetBranch and are the target service binding of some element of  
1614 ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from  
1615 RO. If RO is empty then continue to the next numbered rule.

1616  
1617 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let  
1618 ROT be the set of SpecificationLink instances that are the target object of some element  
1619 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the  
1620 next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is  
1621 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
1622 remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then  
1623 continue to the next numbered rule. If a RegistryObjectQuery element is specified within  
1624 the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat  
1625 RegistryObjectQuery element as follows: Let RO be the result set of the  
1626 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some  
1627 registry object in RO, then remove sl from ROT. If ROT is empty then remove x from  
1628 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery  
1629 element is specified within the SpecificationLinkBranch then let sl be a remaining  
1630 specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the  
1631 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification  
1632 link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then  
1633 remove x from RO. If RO is empty then continue to the next numbered rule.

1634  
1635 If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be  
1636 the set of Association instances that satisfy the AssociationQuery and are the target object  
1637 of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then  
1638 continue to the next numbered rule (Rule 2).

1639

1640 j) If a TargetAssociationBranch element is not specified then go to the next step; otherwise,  
1641 let x be a remaining registry object in RO. If x is not the target object of some  
1642 Association instance, then remove x from RO. If RO is empty then continue to the next  
1643 numbered rule; otherwise, treat each TargetAssociationBranch element separately as  
1644 follows:

1645  
1646 If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be  
1647 the set of all Association instances that have x as a target object; otherwise, let AF be the  
1648 set of Association instances that satisfy the AssociationFilter and have x as the target  
1649 object. If AF is empty, then remove x from RO. If RO is empty then continue to the next  
1650 numbered rule.

1651  
1652 If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be  
1653 the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source  
1654 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
1655 then continue to the next numbered rule.

1656  
1657 If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let  
1658 ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and  
1659 are the source object of some element of AF. If ROS is empty, then remove x from RO. If  
1660 RO is empty then continue to the next numbered rule.

1661  
1662 If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS  
1663 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the  
1664 source object of some element of AF. If ROS is empty, then remove x from RO. If RO is  
1665 empty then continue to the next numbered rule.

1666  
1667 If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS  
1668 be the set of  
1669 RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of  
1670 some element of AF. If ROS is empty, then remove x from RO. If RO is empty then  
1671 continue to the next numbered rule.

1672  
1673 If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let  
1674 ROS be the set of ClassificationScheme instances that satisfy the  
1675 ClassificationSchemeQuery and are the source object of some element of AF. If ROS is  
1676 empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1677  
1678 If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let  
1679 ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery  
1680 and are the source object of some element of AF. If ROS is empty, then remove x from  
1681 RO. If RO is empty then continue to the next numbered rule.

1682

1683 If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS  
1684 be the set of Organization instances that satisfy the OrganizationQuery and are the source  
1685 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
1686 then continue to the next numbered rule.

1687

1688 If an AuditableEventQuery is specified within the TargetAssociationBranch, then let  
1689 ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are  
1690 the source object of some element of AF. If ROS is empty, then remove x from RO. If  
1691 RO is empty then continue to the next numbered rule.

1692

1693 If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let  
1694 ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and  
1695 are the source object of some element of AF. If ROS is empty, then remove x from RO. If  
1696 RO is empty then continue to the next numbered rule.

1697

1698 If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let  
1699 ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are  
1700 the source object of some element of AF. If ROS is empty, then remove x from RO. If  
1701 RO is empty then continue to the next numbered rule.

1702

1703 If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the  
1704 set of Service instances that satisfy the ServiceQuery and are the source object of some  
1705 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue  
1706 to the next numbered rule.

1707

1708 If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set  
1709 of User instances that are the source object of some element of AF. If ROS is empty, then  
1710 remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the  
1711 member of ROS. If a UserFilter element is specified within the UserBranch, and if u does  
1712 not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from  
1713 RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter  
1714 element is specified within the UserBranch, and if the postal address of u does not satisfy  
1715 that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is  
1716 empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are  
1717 specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied  
1718 by some of the telephone numbers of u then remove u from ROS. If ROS is empty, then  
1719 remove x from RO. If RO is empty then continue to the next numbered rule. If an  
1720 OrganizationQuery element is specified within the UserBranch, then let o be the  
1721 Organization instance that is identified by the organization that u is affiliated with. If o  
1722 doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROS.  
1723 If ROS is empty, then remove x from RO. If RO is empty then continue to the next  
1724 numbered rule.

1725

1726 If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be  
1727 the set of Classification instances that satisfy the ClassificationQuery and are the source  
1728 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
1729 then continue to the next numbered rule (Rule 2).

1730

1731 If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS  
1732 be the set of ServiceBinding instances that are the source object of some element of AF.  
1733 If ROS is empty, then remove x from RO. If RO is empty then continue to the next  
1734 numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is  
1735 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then  
1736 remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then  
1737 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the  
1738 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as  
1739 follows:

1740 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link  
1741 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
1742 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
1743 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then  
1744 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
1745 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl  
1746 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:  
1747 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is  
1748 not a specification link for some registry object in RO, then remove sl from SL. If SL is  
1749 empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is  
1750 empty then continue to the next numbered rule. If a RegistryEntryQuery element is  
1751 specified within the SpecificationLinkBranch then let sl be a remaining specification link  
1752 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the  
1753 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some  
1754 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.  
1755 If ROS is empty then remove x from RO. If RO is empty then continue to the next  
1756 numbered rule.

1757



1758 If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let  
 1759 ROS be the set of SpecificationLink instances that are the source object of some element  
 1760 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the  
 1761 next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is  
 1762 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
 1763 remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then  
 1764 continue to the next numbered rule. If a RegistryObjectQuery element is specified within  
 1765 the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat  
 1766 RegistryObjectQuery element as follows: Let RO be the result set of the  
 1767 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some  
 1768 registry object in RO, then remove sl from ROS. If ROS is empty then remove x from  
 1769 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery  
 1770 element is specified within the SpecificationLinkBranch then let sl be a remaining  
 1771 specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the  
 1772 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification  
 1773 link for some registry entry in RE, then remove sl from ROS. If ROS is empty then  
 1774 remove x from RO. If RO is empty then continue to the next numbered rule. If a  
 1775 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT  
 1776 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and  
 1777 are the target service binding of some element of ROT. If SBT is empty then remove sb  
 1778 from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the  
 1779 next numbered rule.

1780  
 1781 If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be  
 1782 the set of Association instances that satisfy the AssociationQuery and are the source  
 1783 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty  
 1784 then continue to the next numbered rule (Rule 2).

- 1785 2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set  
 1786 RO to be the result of the RegistryObjectQuery.
- 1787 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 1788 within the RegistryResponse.

### 1789 Examples

1790 A client application needs all items that are classified by two different classification schemes,  
 1791 one based on "Industry" and another based on "Geography". Both schemes have been defined by  
 1792 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",  
 1793 respectively. The following query identifies registry entries for all registered items that are  
 1794 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of  
 1795 "Asia/Japan".

```
1796
1797 <AdhocQueryRequest>
1798   <ResponseOption returnType = "RegistryEntry"/>
1799   <FilterQuery>
1800     <RegistryObjectQuery>
1801       <ClassifiedByBranch>
1802         <ClassificationFilter>
1803           <Clause>
```

```

1804     <SimpleClause leftArgument = "path">
1805         <StringClause stringPredicate = "Equal">//Automotive</StringClause>
1806     </SimpleClause>
1807 </Clause>
1808 </ClassificationFilter>
1809 <ClassificationSchemeQuery>
1810     <NameBranch>
1811         <LocalizedStringFilter>
1812             <Clause>
1813                 <SimpleClause leftArgument = "value">
1814                     <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
1815                 </SimpleClause>
1816             </Clause>
1817         </LocalizedStringFilter>
1818     </NameBranch>
1819 </ClassificationSchemeQuery>
1820 </ClassifiedByBranch>
1821 <ClassifiedByBranch>
1822     <ClassificationFilter>
1823         <Clause>
1824             <SimpleClause leftArgument = "path">
1825                 <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
1826             </SimpleClause>
1827         </Clause>
1828     </ClassificationFilter>
1829 <ClassificationSchemeQuery>
1830     <NameBranch>
1831         <LocalizedStringFilter>
1832             <Clause>
1833                 <SimpleClause leftArgument = "value">
1834                     <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
1835                 </SimpleClause>
1836             </Clause>
1837         </LocalizedStringFilter>
1838     </NameBranch>
1839 </ClassificationSchemeQuery>
1840 </ClassifiedByBranch>
1841 </RegistryObjectQuery>
1842 </FilterQuery>
1843 </AdhocQueryRequest>
1844

```

1845 A client application wishes to identify all RegistryObject instances that are classified by some  
1846 internal classification scheme and have some given keyword as part of the description of one of  
1847 the classification nodes of that classification scheme. The following query identifies all such  
1848 RegistryObject instances. The query takes advantage of the knowledge that the classification  
1849 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode  
1850 instances.

```

1851
1852 <AdhocQueryRequest>
1853   <ResponseOption returnType = "RegistryObject"/>
1854   <FilterQuery>
1855     <RegistryObjectQuery>
1856       <ClassifiedByBranch>
1857         <ClassificationNodeQuery>
1858           <DescriptionBranch>
1859             <LocalizedStringFilter>
1860               <Clause>
1861                 <SimpleClause leftArgument = "value">
1862                   <StringClause stringPredicate = "Equal">transistor</StringClause>
1863                 </SimpleClause>
1864               </Clause>
1865             </LocalizedStringFilter>
1866           </DescriptionBranch>
1867         </ClassificationNodeQuery>
1868       </ClassifiedByBranch>
1869     </RegistryObjectQuery>
1870   </FilterQuery>
1871 </AdhocQueryRequest>
1872

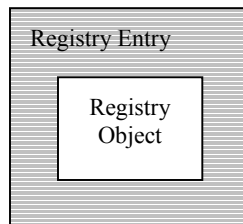
```

### 1873 8.2.3 RegistryEntryQuery

#### 1874 Purpose

1875 To identify a set of registry entry instances as the result of a query over selected registry  
 1876 metadata.

1877



#### 1878 ebRIM Binding

1879

Figure 17: ebRIM Binding for RegistryEntryQuery

#### 1880 Definition

```

1881
1882 <complexType name="RegistryEntryQueryType">
1883   <complexContent>
1884     <extension base="tns:RegistryObjectQueryType">
1885       <sequence>
1886         <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
1887       </sequence>

```

```

1888     </extension>
1889     </complexContent>
1890 </complexType>
1891 <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
1892
1893 <element name="RegistryEntryQueryResult">
1894     <complexType>
1895         <choice minOccurs="0" maxOccurs="unbounded">
1896             <element ref="rim:ObjectRef" />
1897             <element ref="rim:ClassificationScheme" />
1898             <element ref="rim:ExtrinsicObject" />
1899             <element ref="rim:RegistryEntry" />
1900             <element ref="rim:RegistryObject" />
1901             <element ref="rim:RegistryPackage" />
1902         </choice>
1903     </complexType>
1904 </element>
1905

```

## 1906 Semantic Rules

- 1907 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following  
1908 steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
- 1909 a) If RE is empty then continue to the next numbered rule.
- 1910 b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a  
1911 registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.  
1912 If RE is empty then continue to the next numbered rule.
- 1913 c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited  
1914 RegistryObjectQuery over RE as explained in Section 8.2.2.
- 1915 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, set RE  
1916 to be the result of the RegistryEntryQuery.
- 1917 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
1918 within the RegistryResponse.

## 1919 Examples

1920 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if  
1921 they have registered any of their business documents in the Registry. The following query  
1922 returns a set of registry entry identifiers for currently registered items submitted by any  
1923 organization whose name includes the string "XYZ". It does not return any registry entry  
1924 identifiers for superseded, replaced, deprecated, or withdrawn items.

```

1925
1926 <AdhocQueryRequest>
1927     <ResponseOption returnType = "ObjectRef"/>
1928     <FilterQuery>
1929         <RegistryEntryQuery>
1930             <TargetAssociationBranch>
1931                 <AssociationFilter>
1932                     <Clause>
1933                         <SimpleClause leftArgument = "associationType">
1934                             <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>

```

```

1935     </SimpleClause>
1936     </Clause>
1937   </AssociationFilter>
1938   <OrganizationQuery>
1939     <NameBranch>
1940       <LocalizedStringFilter>
1941         <Clause>
1942           <SimpleClause leftArgument = "value">
1943             <StringClause stringPredicate = "Contains">XYZ</StringClause>
1944           </SimpleClause>
1945         </Clause>
1946       </LocalizedStringFilter>
1947     </NameBranch>
1948   </OrganizationQuery>
1949 </TargetAssociationBranch>
1950 <RegistryEntryFilter>
1951   <Clause>
1952     <SimpleClause leftArgument = "status">
1953       <StringClause stringPredicate = "Equal">Approved</StringClause>
1954     </SimpleClause>
1955   </Clause>
1956 </RegistryEntryFilter>
1957 </RegistryEntryQuery>
1958 </FilterQuery>
1959 </AdhocQueryRequest>
1960

```

1961 A client is using the United Nations Standard Product and Services Classification (UNSPSC)
1962 scheme and wants to identify all companies that deal with products classified as "Integrated
1963 circuit components", i.e. UNSPSC code "321118". The client knows that companies have
1964 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
1965 such profile has been classified by UNSPSC according to the products the company deals with.
1966 However, the client does not know if the UNSPSC classification scheme is internal or external to
1967 this registry. The following query returns a set of approved registry entry instances for CPP's of
1968 companies that deal with integrated circuit components.

```

1969
1970 <AdhocQueryRequest>
1971   <ResponseOption returnType = "RegistryEntry"/>
1972   <FilterQuery>
1973     <RegistryEntryQuery>
1974       <ClassifiedByBranch>
1975         <ClassificationFilter>
1976           <Clause>
1977             <SimpleClause leftArgument = "code">
1978               <StringClause stringPredicate = "Equal">321118</StringClause>
1979             </SimpleClause>
1980           </Clause>
1981         </ClassificationFilter>
1982       <ClassificationSchemeQuery>
1983         <NameBranch>
1984           <LocalizedStringFilter>
1985             <Clause>
1986               <SimpleClause leftArgument = "value">
1987                 <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
1988               </SimpleClause>
1989             </Clause>

```

```

1990     </LocalizedStringFilter>
1991     </NameBranch>
1992     </ClassificationSchemeQuery>
1993     </ClassifiedByBranch>
1994     <RegistryEntryFilter>
1995       <Clause>
1996         <CompoundClause connectivePredicate = "And">
1997           <Clause>
1998             <SimpleClause leftArgument = "objectType">
1999               <StringClause stringPredicate = "Equal">CPP</StringClause>
2000             </SimpleClause>
1997           </Clause>
2002           <Clause>
1998             <SimpleClause leftArgument = "status">
1999               <StringClause stringPredicate = "Equal">Approved</StringClause>
2000             </SimpleClause>
1997           </Clause>
1996         </CompoundClause>
1995       </Clause>
1994     </RegistryEntryFilter>
1993     </RegistryEntryQuery>
1992     </FilterQuery>
1991     </AdhocQueryRequest>
2013

```

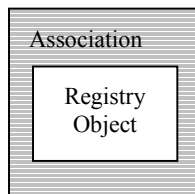
## 2014 8.2.4 AssociationQuery

### 2015 Purpose

2016 To identify a set of association instances as the result of a query over selected registry metadata.

2017

### 2018 ebRIM Binding



2019 **Figure 18: ebRIM Binding for AssociationQuery**

### 2020 Definition

```

2021 <complexType name = "AssociationQueryType">
2022   <complexContent>
2023     <extension base = "tns:RegistryObjectQueryType">
2024       <sequence>
2025         <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
2026       </sequence>
2027     </extension>
2028   </complexContent>
2029 </complexType>
2030 <element name = "AssociationQuery" type = "tns:AssociationQueryType"/>
2031 <element name="AssociationQueryResult">
2032   <complexType>
2033
2034

```

```

2035 <choice minOccurs="0" maxOccurs="unbounded">
2036   <element ref="rim:ObjectRef" />
2037   <element ref="rim:RegistryObject" />
2038   <element ref="rim:Association" />
2039 </choice>
2040 </complexType>
2041 </element>
2042

```

### 2043 Semantic Rules

- 2044 1. Let A denote the set of all persistent Association instances in the Registry. The following
 2045 steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
  - 2046 a) If A is empty then continue to the next numbered rule.
  - 2047 b) If an AssociationFilter element is not directly contained in the AssociationQuery element,
 2048 then go to the next step; otherwise let x be an association instance in A. If x does not
 2049 satisfy the AssociationFilter then remove x from A. If A is empty then continue to the
 2050 next numbered rule.
  - 2051 c) Let A be the set of remaining Association instances. Evaluate inherited
 2052 RegistryObjectQuery over A as explained in Section 8.2.2.
- 2053 2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to
 2054 be the result of the AssociationQuery.
- 2055 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
 2056 within the RegistryResponse.

### 2057 Examples

2058 A client application wishes to identify a set of associations that are 'equivalentTo' a set of other
 2059 associations.

```

2060 <AdhocQueryRequest">
2061   <ResponseOption returnType="LeafClass" />
2062   <FilterQuery>
2063     <AssociationQuery>
2064       <SourceAssociationBranch>
2065         <AssociationFilter>
2066           <Clause>
2067             <SimpleClause leftArgument="associationType">
2068               <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
2069             </SimpleClause>
2070           </Clause>
2071         </AssociationFilter>
2072       </AssociationQuery>
2073     <AssociationQuery>
2074       <AssociationFilter>
2075         <Clause>
2076           <SimpleClause leftArgument="associationType">
2077             <StringClause stringPredicate="StartsWith">Sin</StringClause>
2078           </SimpleClause>
2079         </Clause>
2080       </AssociationFilter>
2081     </AssociationQuery>
2082   </SourceAssociationBranch>
2083   <AssociationFilter>

```

```

2084     <Clause>
2085       <SimpleClause leftArgument="associationType">
2086         <StringClause stringPredicate="StartsWith">Son</StringClause>
2087       </SimpleClause>
2088     </Clause>
2089   </AssociationFilter>
2090 </AssociationQuery>
2091 </FilterQuery>
2092 </AdhocQueryRequest>
2093

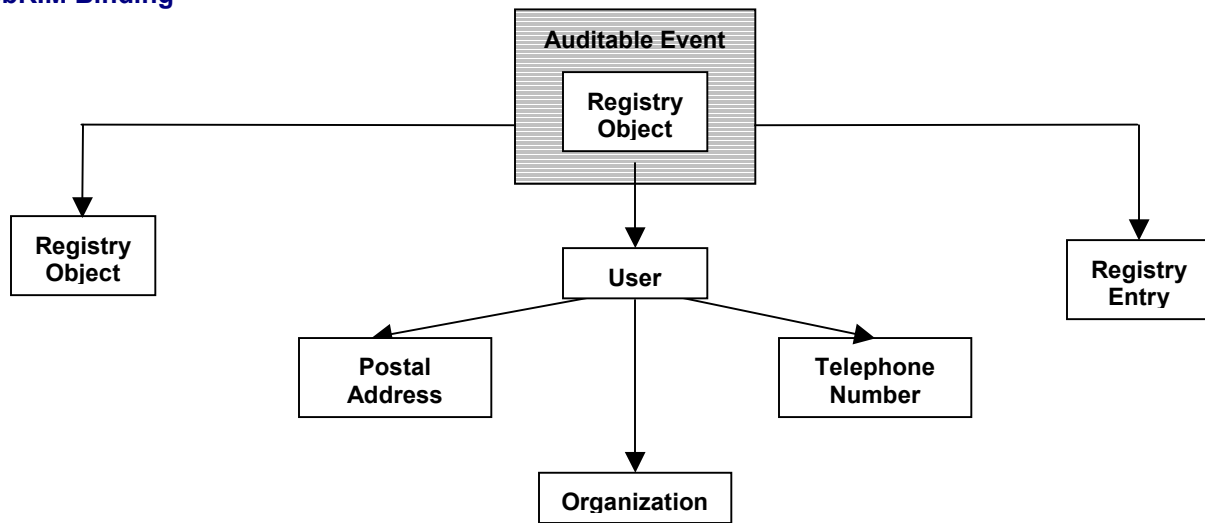
```

## 2094 8.2.5 AuditableEventQuery

### 2095 Purpose

2096 To identify a set of auditable event instances as the result of a query over selected registry  
2097 metadata.

### 2098 ebRIM Binding



2099 **Figure 19: ebRIM Binding for AuditableEventQuery**

### 2100 Definition

```

2101 <complexType name="AuditableEventQueryType">
2102   <complexContent>
2103     <extension base="tns:RegistryObjectQueryType">
2104       <sequence>
2105         <element ref="tns:AuditableEventFilter" minOccurs="0" />
2106         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2107         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2108         <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2109       </sequence>
2110     </extension>
2111   </complexContent>
2112 </complexType>
2113 </complexType>
2114 <element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />
2115 </element>
2116 <complexType>
2117

```



```

2118 <choice minOccurs="0" maxOccurs="unbounded">
2119   <element ref="rim:ObjectRef" />
2120   <element ref="rim:RegistryObject" />
2121   <element ref="rim:AuditableEvent" />
2122 </choice>
2123 </complexType>
2124 </element>
2125

```

## 2126 Semantic Rules

- 2127 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The  
 2128 following steps will eliminate instances in AE that do not satisfy the conditions of the  
 2129 specified filters.
- 2130 a) If AE is empty then continue to the next numbered rule.
- 2131 b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an  
 2132 auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from  
 2133 AE. If AE is empty then continue to the next numbered rule.
- 2134 c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let  
 2135 x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:  
 2136 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is  
 2137 not an auditable event for some registry object in RO, then remove x from AE. If AE is  
 2138 empty then continue to the next numbered rule.
- 2139 d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x  
 2140 be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let  
 2141 RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an  
 2142 auditable event for some registry entry in RE, then remove x from AE. If AE is empty  
 2143 then continue to the next numbered rule.
- 2144 e) If a UserBranch element is not specified then go to the next step; otherwise, let x be a  
 2145 remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter  
 2146 element is specified within the UserBranch, and if u does not satisfy that filter, then  
 2147 remove x from AE. If a PostalAddressFilter element is specified within the UserBranch,  
 2148 and if the postal address of u does not satisfy that filter, then remove x from AE. If  
 2149 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the  
 2150 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then  
 2151 remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if  
 2152 any of the EmailAddressFilters isn't satisfied by some of the email addresses of u then  
 2153 remove x from AE. If an OrganizationQuery element is specified within the UserBranch,  
 2154 then let o be the Organization instance that is identified by the organization that u is  
 2155 affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then  
 2156 remove x from AE. If AE is empty then continue to the next numbered rule.
- 2157 f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited  
 2158 RegistryObjectQuery over AE as explained in Section 8.2.2.
- 2159 2. If AE is empty, then raise the warning: **auditable event query result is empty**; otherwise set  
 2160 AE to be the result of the AuditableEventQuery.
- 2161 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2162 within the RegistryResponse.

2163 **Examples**

2164 A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The  
 2165 client is now interested in all events since the beginning of the year that have impacted that item.  
 2166 The following query will return a set of AuditableEvent instances for all such events.

```

2167 <AdhocQueryRequest>
2168   <ResponseOption returnType = "LeafClass"/>
2169   <FilterQuery>
2170     <AuditableEventQuery>
2171       <AuditableEventFilter>
2172         <Clause>
2173           <SimpleClause leftArgument = "timestamp">
2174             <RationalClause logicalPredicate = "GE">
2175               <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2176             </RationalClause>
2177           </SimpleClause>
2178         </AuditableEventFilter>
2179       <RegistryEntryQuery>
2180         <NameBranch>
2181           <LocalizedStringFilter>
2182             <Clause>
2183               <SimpleClause leftArgument = "value">
2184                 <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
2185               </SimpleClause>
2186             </Clause>
2187           </LocalizedStringFilter>
2188         </NameBranch>
2189       </RegistryEntryQuery>
2190     </AuditableEventQuery>
2191   </FilterQuery>
2192 </AdhocQueryRequest>
2193
2194
2195
  
```

2196 A client company has many registered objects in the Registry. The Registry allows events  
 2197 submitted by other organizations to have an impact on your registered items, e.g. new  
 2198 classifications and new associations. The following query will return a set of identifiers for all  
 2199 auditable events, invoked by some other party, that had an impact on an item submitted by  
 2200 "myorg".

```

2201 <AdhocQueryRequest>
2202   <ResponseOption returnType = "LeafClass"/>
2203   <FilterQuery>
2204     <AuditableEventQuery>
2205       <RegistryEntryQuery>
2206         <TargetAssociationBranch>
2207           <AssociationFilter>
2208             <Clause>
2209               <SimpleClause leftArgument = "associationType">
2210                 <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2211               </SimpleClause>
2212             </Clause>
2213           </AssociationFilter>
2214         <OrganizationQuery>
2215           <NameBranch>
2216             <LocalizedStringFilter>
2217
  
```

```

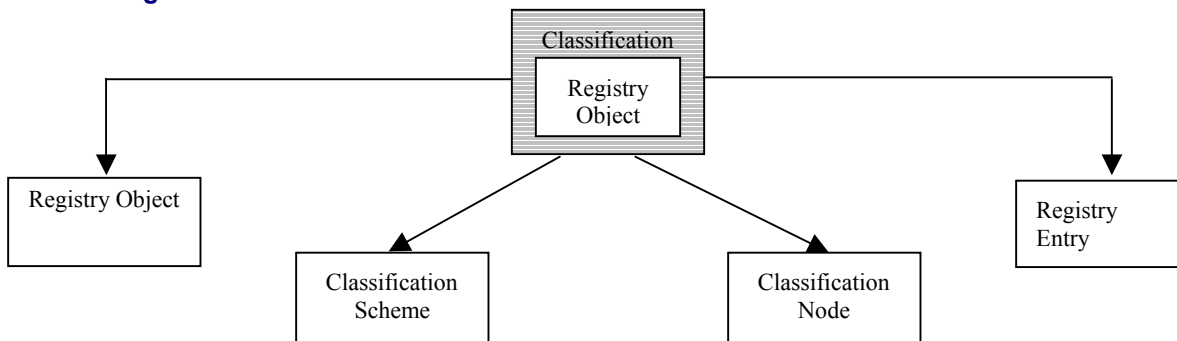
2218     <Clause>
2219         <SimpleClause leftArgument = "value">
2220             <StringClause stringPredicate = "Equal">myorg</StringClause>
2221         </SimpleClause>
2222     </Clause>
2223 </LocalizedStringFilter>
2224 </NameBranch>
2225 </OrganizationQuery>
2226 </TargetAssociationBranch>
2227 </RegistryEntryQuery>
2228 <UserBranch>
2229     <OrganizationQuery>
2230         <NameBranch>
2231             <LocalizedStringFilter>
2232                 <Clause>
2233                     <SimpleClause leftArgument = "value">
2234                         <StringClause stringPredicate = "-Equal">myorg</StringClause>
2235                     </SimpleClause>
2236                 </Clause>
2237             </LocalizedStringFilter>
2238         </NameBranch>
2239     </OrganizationQuery>
2240 </UserBranch>
2241 </AuditableEventQuery>
2242 </FilterQuery>
2243 </AdhocQueryRequest>
2244
    
```

2245 **8.2.6 ClassificationQuery**

2246 **Purpose**

2247 To identify a set of classification instances as the result of a query over selected registry  
 2248 metadata.

2249 **ebRIM Binding**



2250 **Figure 20: ebRIM Binding for ClassificationQuery**

2251 **Definition**

```

2252 <complexType name = "ClassificationQueryType">
2253     <complexContent>
2254         <extension base = "tns:RegistryObjectQueryType">
2255             <sequence>
2256                 <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
2257             </sequence>
2258         </extension>
2259     </complexContent>
2260 </complexType>
    
```

```

2258     <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
2259     <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
2260     <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
2261     <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
2262     </sequence>
2263   </extension>
2264 </complexContent>
2265 </complexType>
2266 <element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>
2267
2268 <element name="ClassificationQueryResult">
2269   <complexType>
2270     <choice minOccurs="0" maxOccurs="unbounded">
2271       <element ref="rim:ObjectRef" />
2272       <element ref="rim:RegistryObject" />
2273       <element ref="rim:Classification" />
2274     </choice>
2275   </complexType>
2276 </element>
2277

```

## 2278 Semantic Rules

- 2279 1. Let C denote the set of all persistent Classification instances in the Registry. The following
 2280 steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
  - 2281 a) If C is empty then continue to the next numbered rule.
  - 2282 b) If a ClassificationFilter element is not directly contained in the ClassificationQuery
 2283 element, then go to the next step; otherwise let x be an classification instance in C. If x
 2284 does not satisfy the ClassificationFilter then remove x from C. If C is empty then
 2285 continue to the next numbered rule.
  - 2286 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
 2287 be a remaining classification in C. If the defining classification scheme of x does not
 2288 satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C.
 2289 If C is empty then continue to the next numbered rule.
  - 2290 d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be
 2291 a remaining classification in C. If the classification node of x does not satisfy the
 2292 ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is
 2293 empty then continue to the next numbered rule.
  - 2294 e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
 2295 x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let
 2296 RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a
 2297 classification of at least one registry object in RO, then remove x from C. If C is empty
 2298 then continue to the next numbered rule.
  - 2299 f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
 2300 be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE
 2301 be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a
 2302 classification of at least one registry entry in RE, then remove x from C. If C is empty
 2303 then continue to the next numbered rule.

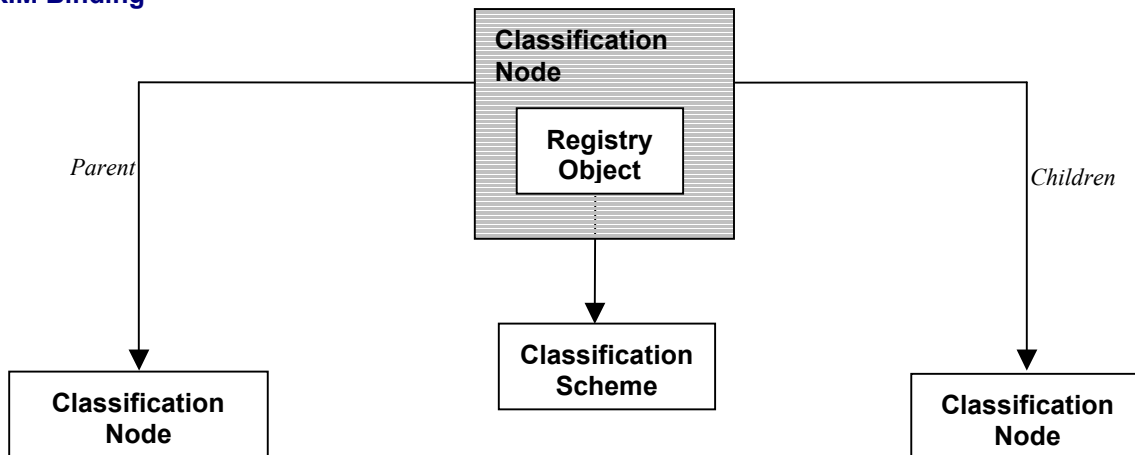
- 2304 2. If C is empty, then raise the warning: *classification query result is empty*; otherwise  
 2305 otherwise, set C to be the result of the ClassificationQuery.  
 2306 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2307 within the RegistryResponse.

## 2308 8.2.7 ClassificationNodeQuery

### 2309 Purpose

2310 To identify a set of classification node instances as the result of a query over selected registry  
 2311 metadata.

### 2312 ebRIM Binding



2313 Figure 21: ebRIM Binding for ClassificationNodeQuery

### 2314 Definition

```

2315 <complexType name="ClassificationNodeQueryType">
2316   <complexContent>
2317     <extension base="tns:RegistryObjectQueryType">
2318       <sequence>
2319         <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
2320         <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
2321         <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
2322           maxOccurs="1" />
2323         <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
2324           minOccurs="0" maxOccurs="unbounded" />
2325       </sequence>
2326     </extension>
2327   </complexContent>
2328 </complexType>
2329 <element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />
2330
2331 <element name="ClassificationNodeQueryResult">
2332   <complexType>
2333     <choice minOccurs="0" maxOccurs="unbounded">
2334       <element ref="rim:ObjectRef" />
2335       <element ref="rim:RegistryObject" />
2336       <element ref="rim:ClassificationNode" />
2337     </choice>
2338   </complexType>
  
```

```
2339 </complexType>  
2340 </element>  
2341
```

## 2342 **Semantic Rules**

- 2343 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The  
2344 following steps will eliminate instances in CN that do not satisfy the conditions of the  
2345 specified filters.
- 2346 a) If CN is empty then continue to the next numbered rule.
- 2347 b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a  
2348 classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove  
2349 x from CN. If CN is empty then continue to the next numbered rule.
- 2350 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x  
2351 be a remaining classification node in CN. If the defining classification scheme of x does  
2352 not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x  
2353 from CN. If CN is empty then continue to the next numbered rule.
- 2354 d) If a ClassificationNodeParentBranch element is not specified, then go to the next step;  
2355 otherwise, let x be a remaining classification node in CN and execute the following  
2356 paragraph with n=x.  
2357 Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base  
2358 level node), then remove x from CN and go to the next step; otherwise, let p be the parent  
2359 node of n. If a ClassificationNodeFilter element is directly contained in the  
2360 ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,  
2361 then remove x from CN. If CN is empty then continue to the next numbered rule. If a  
2362 ClassificationSchemeQuery element is directly contained in the  
2363 ClassificationNodeParentBranch and if defining classification scheme of p does not  
2364 satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then  
2365 continue to the next numbered rule.  
2366 If another ClassificationNodeParentBranch element is directly contained within this  
2367 ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.
- 2368 e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next  
2369 numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the  
2370 parent node of some ClassificationNode instance, then remove x from CN and if CN is  
2371 empty continue to the next numbered rule; otherwise, treat each  
2372 ClassificationNodeChildrenBranch element separately and execute the following  
2373 paragraph with n = x.

- 2374 Let  $n$  be a classification node instance. If a ClassificationNodeFilter element is not  
 2375 specified within the ClassificationNodeChildrenBranch element then let CNC be the set  
 2376 of all classification nodes that have  $n$  as their parent node; otherwise, let CNC be the set  
 2377 of all classification nodes that satisfy the ClassificationNodeFilter and have  $n$  as their  
 2378 parent node. If CNC is empty, then remove  $x$  from CN and if CN is empty continue to the  
 2379 next numbered rule; otherwise, let  $c$  be any member of CNC. If a  
 2380 ClassificationSchemeQuery element is directly contained in the  
 2381 ClassificationNodeChildrenBranch and if the defining classification scheme of  $c$  does not  
 2382 satisfy the ClassificationSchemeQuery then remove  $c$  from CNC. If CNC is empty then  
 2383 remove  $x$  from CN. If CN is empty then continue to the next numbered rule; otherwise,  
 2384 let  $y$  be an element of CNC and continue with the next paragraph.
- 2385 If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly  
 2386 contain another ClassificationNodeChildrenBranch element, then continue to the next  
 2387 numbered rule; otherwise, repeat the previous paragraph with the new  
 2388 ClassificationNodeChildrenBranch element and with  $n = y$ .
- 2389 f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited  
 2390 RegistryObjectQuery over CN as explained in Section 8.2.2.
- 2391 2. If CN is empty, then raise the warning: **classification node query result is empty**; otherwise  
 2392 set CN to be the result of the ClassificationNodeQuery.
- 2393 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2394 within the RegistryResponse.

### 2395 Path Filter Expression usage in ClassificationNodeFilter

2396 The path filter expression is used to match classification nodes in ClassificationNodeFilter  
 2397 elements involving the path attribute of the ClassificationNode class as defined by the getPath  
 2398 method in [ebRIM].

2399 The path filter expressions are based on a very small and proper sub-set of location path syntax  
 2400 of XPath.

2401 The path filter expression syntax includes support for matching multiple nodes by using wild  
 2402 card syntax as follows:

- 2403 • Use of '\*' as a wildcard in place of any path element in the pathFilter
- 2404 • Use of '/' syntax to denote any descendent of a node in the pathFilter

2405 It is defined by the following BNF grammar:

```

2406 pathFilter      ::= '/' schemeId nodePath
2407 nodePath       ::= slashes nodeCode
2408                | slashes '*'
2409                | slashes nodeCode ( nodePath )?
2410 Slashes ::= '/' | '/'
2411
2412 
```

2413 In the above grammar, schemeId is the id attribute of the ClassificationScheme instance. In the  
 2414 above grammar nodeCode is defined by NCName production as defined by  
 2415 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

2416 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a  
 2417 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause  
 2418 is a PATH Filter expression.

2419 This is illustrated in the following example that matches all second level nodes in  
 2420 ClassificationScheme with id ‘Geography-id’ and with code ‘Japan’:

2421  
2422  
2423  
2424  
2425  
2426  
2427  
2428  
2429  
2430  
2431

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

2432 **Use Cases and Examples of Path Filter Expressions**

2433 The following table lists various use cases and examples using the sample Geography scheme  
 2434 below:

2435  
2436  
2437  
2438  
2439  
2440  
2441  
2442  
2443  
2444

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

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

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

2445 **Table 10: 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'

2446 **Examples**

2447 A client application wishes to identify all of the classification nodes in the first three levels of a  
 2448 classification scheme hierarchy. The client knows that the name of the underlying classification



2449 scheme is “urn:ebxml:cs:myscheme”. The following query identifies all nodes at the first three  
 2450 levels.

```

2451 <AdhocQueryRequest>
2452   <ResponseOption returnType = "LeafClass"/>
2453   <FilterQuery>
2454     <ClassificationNodeQuery>
2455       <ClassificationNodeFilter>
2456         <Clause>
2457           <SimpleClause leftArgument = "levelNumber">
2458             <RationalClause logicalPredicate = "LE">
2459               <IntClause>3</IntClause>
2460             </RationalClause>
2461           </SimpleClause>
2462         </Clause>
2463       </ClassificationNodeFilter>
2464     </ClassificationNodeQuery>
2465     <ClassificationSchemeQuery>
2466       <NameBranch>
2467         <LocalizedStringFilter>
2468           <Clause>
2469             <SimpleClause leftArgument = "value">
2470               <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
2471             </SimpleClause>
2472           </Clause>
2473         </LocalizedStringFilter>
2474       </NameBranch>
2475     </ClassificationSchemeQuery>
2476   </FilterQuery>
2477 </AdhocQueryRequest>
2478
2479
```

2480 If, instead, the client wishes all levels returned, they could simply delete the  
 2481 ClassificationNodeFilter element from the query.

2482 The following query finds all children nodes of a first level node whose code is NorthAmerica.

```

2483 <AdhocQueryRequest>
2484   <ResponseOption returnType = "LeafClass"/>
2485   <FilterQuery>
2486     <ClassificationNodeQuery>
2487       <ClassificationNodeFilter>
2488         <Clause>
2489           <SimpleClause leftArgument = "path">
2490             <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
2491           </SimpleClause>
2492         </Clause>
2493       </ClassificationNodeFilter>
2494     </ClassificationNodeQuery>
2495   </FilterQuery>
2496 </AdhocQueryRequest>
2497
2498
```

2499 The following query finds all third level nodes with code of Tokyo.

```

2500 <AdhocQueryRequest>
2501   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2502   <FilterQuery>
2503
```

```

2504 <ClassificationNodeQuery>
2505   <ClassificationNodeFilter>
2506     <Clause>
2507       <SimpleClause leftArgument = "path">
2508         <StringClause stringPredicate = "Equal">/Geography-id/*/*Tokyo</StringClause>
2509       </SimpleClause>
2510     </Clause>
2511   </ClassificationNodeFilter>
2512 </ClassificationNodeQuery>
2513 </FilterQuery>
2514 </AdhocQueryRequest>
2515

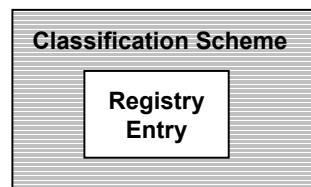
```

## 2516 8.2.8 ClassificationSchemeQuery

### 2517 Purpose

2518 To identify a set of classification scheme instances as the result of a query over selected registry  
 2519 metadata.

### 2520 ebRIM Binding



2521 **Figure 22: ebRIM Binding for ClassificationSchemeQuery**

### 2522 Definition

```

2523 <complexType name="ClassificationSchemeQueryType">
2524   <complexContent>
2525     <extension base="tns:RegistryEntryQueryType">
2526       <sequence>
2527         <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
2528       </sequence>
2529     </extension>
2530   </complexContent>
2531 </complexType>
2532 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
2533
2534

```

### 2535 Semantic Rules

- 2536 1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The  
 2537 following steps will eliminate instances in CS that do not satisfy the conditions of the  
 2538 specified filters.
  - 2539 a) If CS is empty then continue to the next numbered rule.
  - 2540 b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x  
 2541 be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,  
 2542 then remove x from CS. If CS is empty then continue to the next numbered rule.

- 2543 c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited  
 2544 RegistryEntryQuery over CS as explained in Section 8.2.3.
- 2545 2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,  
 2546 set CS to be the result of the ClassificationSchemeQuery.
- 2547 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2548 within the RegistryResponse.

### 2549 Examples

2550 A client application wishes to identify all classification scheme instances in the Registry.

```
2551 <AdhocQueryRequest>
2552   <ResponseOption returnType = "LeafClass"/>
2553   <FilterQuery>
2554     <ClassificationSchemeQuery/>
2555   </FilterQuery>
2556 </AdhocQueryRequest>
```

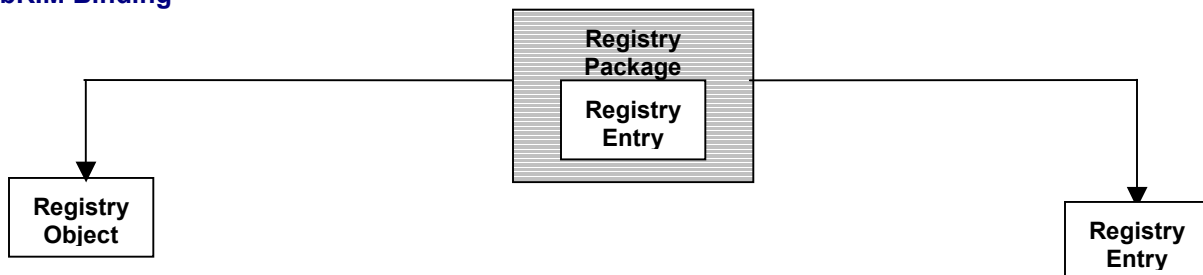
2557

## 2558 8.2.9 RegistryPackageQuery

### 2559 Purpose

2560 To identify a set of registry package instances as the result of a query over selected registry  
 2561 metadata.

### 2562 ebRIM Binding



2563 Figure 23: ebRIM Binding for RegistryPackageQuery

### 2564 Definition

```
2565 <complexType name="RegistryPackageQueryType">
2566   <complexContent>
2567     <extension base="tns:RegistryEntryQueryType">
2568       <sequence>
2569         <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
2570         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
2571         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
2572       </sequence>
2573     </extension>
2574   </complexContent>
2575 </complexType>
2576 <element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />
2577 <element name="RegistryPackageQueryResult">
```

```

2580 <complexType>
2581   <choice minOccurs="0" maxOccurs="unbounded">
2582     <element ref="rim:ObjectRef" />
2583     <element ref="rim:RegistryEntry" />
2584     <element ref="rim:RegistryObject" />
2585     <element ref="rim:RegistryPackage" />
2586   </choice>
2587 </complexType>
2588 </element>
2589

```

## 2590 Semantic Rules

- 2591 1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The  
2592 following steps will eliminate instances in RP that do not satisfy the conditions of the  
2593 specified filters.
  - 2594 a) If RP is empty then continue to the next numbered rule.
  - 2595 b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule;  
2596 otherwise, let x be a registry package instance in RP. If x does not satisfy the  
2597 RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next  
2598 numbered rule.
  - 2599 c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery  
2600 element then treat each RegistryObjectQuery as follows: let RO be the set of  
2601 RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2  
2602 and let PO be the subset of RO that are members of the package x. If PO is empty, then  
2603 remove x from RP. If RP is empty then continue to the next numbered rule. If a  
2604 RegistryEntryQuery element is directly contained in the RegistryPackageQuery element  
2605 then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry  
2606 instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be  
2607 the subset of RE that are members of the package x. If PE is empty, then remove x from  
2608 RP. If RP is empty then continue to the next numbered rule.
  - 2609 d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited  
2610 RegistryEntryQuery over RP as explained in Section 8.2.3.
- 2611 2. If RP is empty, then raise the warning: **registry package query result is empty**; otherwise set  
2612 RP to be the result of the RegistryPackageQuery.
- 2613 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
2614 within the RegistryResponse.

## 2615 Examples

2616 A client application wishes to identify all package instances in the Registry that contain an  
2617 Invoice extrinsic object as a member of the package.

```

2618 <AdhocQueryRequest>
2619   <ResponseOption returnType = "LeafClass"/>
2620   <FilterQuery>
2621     <RegistryPackageQuery>
2622       <RegistryEntryQuery>
2623         <RegistryEntryFilter>
2624           <Clause>
2625

```

```

2626     <SimpleClause leftArgument = "objectType">
2627         <StringClause stringPredicate = "Equal">Invoice</StringClause>
2628     </SimpleClause>
2629 </Clause>
2630 </RegistryEntryFilter>
2631 </RegistryEntryQuery>
2632 </RegistryPackageQuery>
2633 </FilterQuery>
2634 </AdhocQueryRequest>
2635

```

2636 A client application wishes to identify all package instances in the Registry that are not empty.

```

2637 <AdhocQueryRequest>
2638   <ResponseOption returnType = "LeafClass"/>
2639   <FilterQuery>
2640     <RegistryPackageQuery>
2641       <RegistryObjectQuery/>
2642     </RegistryPackageQuery>
2643   </FilterQuery>
2644 </AdhocQueryRequest>
2645
2646

```

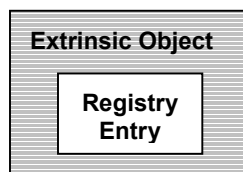
2647 A client application wishes to identify all package instances in the Registry that are empty. Since  
 2648 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate  
 2649 RegistryPackageQuery requests, one to find all packages and another to find all non-empty  
 2650 packages, and then do the set difference themselves. Alternatively, they could do a more  
 2651 complex RegistryEntryQuery and check that the packaging association between the package and  
 2652 its members is non-existent.

2653 Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by  
 2654 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an  
 2655 equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However,  
 2656 the equivalent RegistryEntryQuery is often more complicated to write.

## 2657 8.2.10 ExtrinsicObjectQuery

### 2658 Purpose

2659 To identify a set of extrinsic object instances as the result of a query over selected registry  
 2660 metadata.



### 2661 ebRIM Binding

2662

Figure 24: ebRIM Binding for ExtrinsicObjectQuery

### 2663 Definition

2664

```

2665 <complexType name="ExtrinsicObjectQueryType">
2666   <complexContent>
2667     <extension base="tns:RegistryEntryQueryType">
2668       <sequence>
2669         <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
2670       </sequence>
2671     </extension>
2672   </complexContent>
2673 </complexType>
2674 <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
2675
2676 <element name="ExtrinsicObjectQueryResult">
2677   <complexType>
2678     <choice minOccurs="0" maxOccurs="unbounded">
2679       <element ref="rim:ObjectRef" />
2680       <element ref="rim:RegistryEntry" />
2681       <element ref="rim:RegistryObject" />
2682       <element ref="rim:ExtrinsicObject" />
2683     </choice>
2684   </complexType>
2685 </element>
2686

```

## 2687 **Semantic Rules**

- 2688 1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The  
 2689 following steps will eliminate instances in EO that do not satisfy the conditions of the  
 2690 specified filters.
  - 2691 a) If EO is empty then continue to the next numbered rule.
  - 2692 b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an  
 2693 extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from  
 2694 EO. If EO is empty then continue to the next numbered rule.
  - 2695 c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited  
 2696 RegistryEntryQuery over EO as explained in Section 8.2.3.
- 2697 2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set  
 2698 EO to be the result of the ExtrinsicObjectQuery.
- 2699 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2700 within the RegistryResponse.

## 2701 **8.2.11 OrganizationQuery**

### 2702 **Purpose**

2703 To identify a set of organization instances as the result of a query over selected registry  
 2704 metadata.

### 2705 **ebRIM Binding**

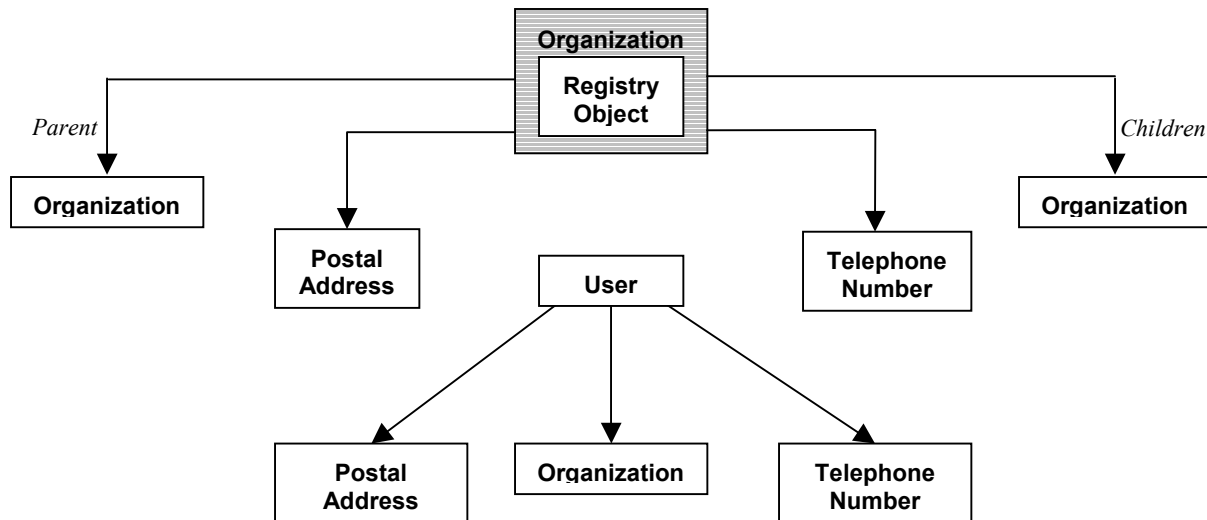


Figure 25: ebRIM Binding for OrganizationQuery

2706

2707

2708

2709

2710

2711

2712

2713

2714

2715

2716

2717

2718

2719

2720

2721

2722

2723

2724

2725

2726

2727

2728

2729

2730

2731

2732

2733

2734

2735

2736

**Definition**

```

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

```

2737

**Semantic Rules**

2738

2739

2740

2741

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.

- 2742        b) If an OrganizationFilter element is not directly contained in the OrganizationQuery  
2743        element, then go to the next step; otherwise let x be an organization instance in ORG. If x  
2744        does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then  
2745        continue to the next numbered rule.
- 2746        c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery  
2747        element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal  
2748        address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is  
2749        empty then continue to the next numbered rule.
- 2750        d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery  
2751        element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of  
2752        the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then  
2753        remove x from ORG. If ORG is empty then continue to the next numbered rule.
- 2754        e) If a UserBranch element is not directly contained in the OrganizationQuery element then  
2755        go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user  
2756        instance that is affiliated with x. If a UserFilter element is specified within the  
2757        UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a  
2758        PostalAddressFilter element is specified within the UserBranch, and if the postal address  
2759        of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s)  
2760        are specified within the UserBranch and if any of the TelephoneNumberFilters isn't  
2761        satisfied by some of the telephone numbers of x then remove x from ORG. If  
2762        EmailAddressFilter(s) are specified within the UserBranch and if any of the  
2763        EmailAddressFilters isn't satisfied by some of the email addresses of x then remove x  
2764        from ORG. If an OrganizationQuery element is specified within the UserBranch, then let  
2765        o be the Organization instance that is identified by the organization that u is affiliated  
2766        with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x  
2767        from ORG. If ORG is empty then continue to the next numbered rule.
- 2768        f) If a OrganizationParentBranch element is not specified within the OrganizationQuery,  
2769        then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the  
2770        following paragraph with o = x:  
2771        Let o be an organization instance. If an OrganizationFilter is not specified within the  
2772        OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the  
2773        Organization hierarchy), then remove x from ORG; otherwise, let p be the parent  
2774        organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.  
2775        If ORG is empty then continue to the next numbered rule.  
2776        If another OrganizationParentBranch element is directly contained within this  
2777        OrganizationParentBranch element, then repeat the previous paragraph with o = p.
- 2778        g) If a OrganizationChildrenBranch element is not specified, then continue to the next  
2779        numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent  
2780        node of some organization instance, then remove x from ORG and if ORG is empty  
2781        continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch  
2782        element separately and execute the following paragraph with n = x.



- 2783 Let  $n$  be an organization instance. If an OrganizationFilter element is not specified within  
 2784 the OrganizationChildrenBranch element then let ORGC be the set of all organizations  
 2785 that have  $n$  as their parent node; otherwise, let ORGC be the set of all organizations that  
 2786 satisfy the OrganizationFilter and have  $n$  as their parent node. If ORGC is empty, then  
 2787 remove  $x$  from ORG and if ORG is empty continue to the next numbered rule; otherwise,  
 2788 let  $c$  be any member of ORGC. If a PostalAddressFilter element is directly contained in  
 2789 the OrganizationChildrenBranch and if the postal address of  $c$  does not satisfy the  
 2790 PostalAddressFilter then remove  $c$  from ORGC. If ORGC is empty then remove  $x$  from  
 2791 ORG. If ORG is empty then continue to the next numbered rule. If no  
 2792 TelephoneNumberFilter element is directly contained in the OrganizationChildrenBranch  
 2793 and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone  
 2794 numbers of  $c$  then remove  $c$  from ORGC. If ORGC is empty then remove  $x$  from ORG. If  
 2795 ORG is empty then continue to the next numbered rule; otherwise, let  $y$  be an element of  
 2796 ORGC and continue with the next paragraph.
- 2797 If the OrganizationChildrenBranch element is terminal, i.e. if it does not directly contain  
 2798 another OrganizationChildrenBranch element, then continue to the next numbered rule;  
 2799 otherwise, repeat the previous paragraph with the new OrganizationChildrenBranch  
 2800 element and with  $n = y$ .
- 2801 h) Let ORG be the set of remaining Organization instances. Evaluate inherited  
 2802 RegistryObjectQuery over ORG as explained in Section 8.2.2.
- 2803 2. If ORG is empty, then raise the warning: *organization query result is empty*; otherwise set  
 2804 ORG to be the result of the OrganizationQuery.
- 2805 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2806 within the RegistryResponse.

## 2807 Examples

2808 A client application wishes to identify a set of organizations, based in France, that have  
 2809 submitted a PartyProfile extrinsic object this year.

```

2810 <AdhocQueryRequest>
2811 <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2812 <FilterQuery>
2813 <OrganizationQuery>
2814 <SourceAssociationBranch>
2815 <AssociationFilter>
2816 <Clause>
2817 <SimpleClause leftArgument = "associationType">
2818 <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2819 </SimpleClause>
2820 </Clause>
2821 </AssociationFilter>
2822 <RegistryObjectQuery>
2823 <RegistryObjectFilter>
2824 <Clause>
2825 <SimpleClause leftArgument = "objectType">
2826 <StringClause stringPredicate = "Equal">CPP</StringClause>
2827 </SimpleClause>
2828 </Clause>
2829 </RegistryObjectFilter>
2830 <AuditableEventQuery>
2831 <AuditableEventFilter>
2832 <Clause>
2833 <SimpleClause leftArgument = "timestamp">
2834 <RationalClause logicalPredicate = "GE">
2835 <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2836 </RationalClause>
2837 </Clause>
  
```

```

2838         </SimpleClause>
2839     </Clause>
2840 </AuditableEventFilter>
2841 </AuditableEventQuery>
2842 </RegistryObjectQuery>
2843 </SourceAssociationBranch>
2844 <PostalAddressFilter>
2845     <Clause>
2846         <SimpleClause leftArgument = "country">
2847             <StringClause stringPredicate = "Equal">France</StringClause>
2848         </SimpleClause>
2849     </Clause>
2850 </PostalAddressFilter>
2851 </OrganizationQuery>
2852 </FilterQuery>
2853 </AdhocQueryRequest>
2854

```

2855 A client application wishes to identify all organizations that have Corporation named XYZ as a  
 2856 parent.

```

2857 <AdhocQueryRequest>
2858 <ResponseOption returnType = "LeafClass"/>
2859 <FilterQuery>
2860     <OrganizationQuery>
2861         <OrganizationParentBranch>
2862             <NameBranch>
2863                 <LocalizedStringFilter>
2864                     <Clause>
2865                         <SimpleClause leftArgument = "value">
2866                             <StringClause stringPredicate = "Equal">XYZ</StringClause>
2867                         </SimpleClause>
2868                     </Clause>
2869                 </LocalizedStringFilter>
2870             </NameBranch>
2871         </OrganizationParentBranch>
2872     </OrganizationQuery>
2873 </FilterQuery>
2874 </AdhocQueryRequest>
2875
2876

```

## 2877 8.2.12 ServiceQuery

### 2878 Purpose

2879

2880 To identify a set of service instances as the result of a query over selected registry metadata.

### 2881 ebRIM Binding

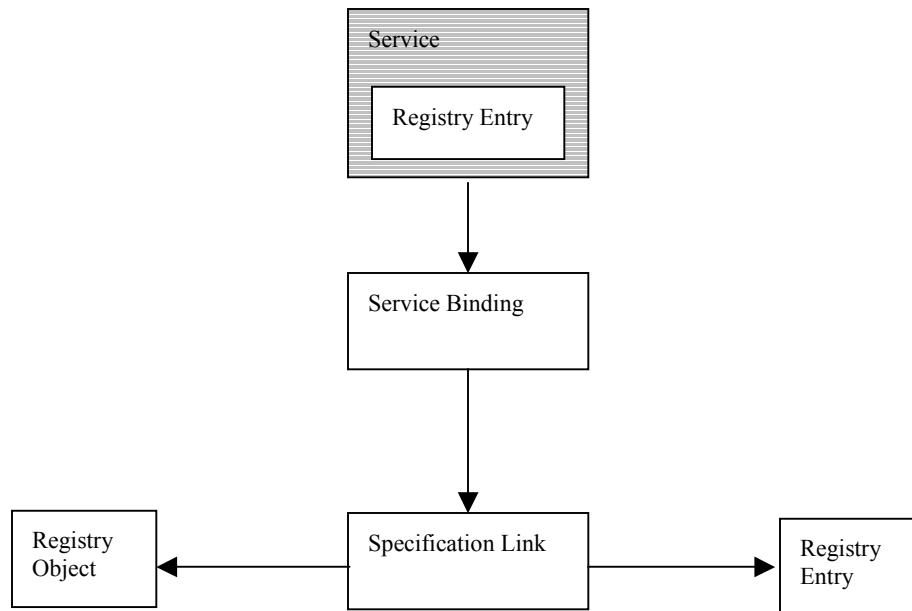


Figure 26: ebRIM Binding for ServiceQuery

2882

### 2883 Definition

```

2884
2885 <complexType name="ServiceQueryType">
2886   <complexContent>
2887     <extension base="tns:RegistryEntryQueryType">
2888       <sequence>
2889         <element ref="tns:ServiceFilter" minOccurs="0"
2890           maxOccurs="1" />
2891         <element ref="tns:ServiceBindingBranch" minOccurs="0"
2892           maxOccurs="unbounded" />
2893       </sequence>
2894     </extension>
2895   </complexContent>
2896 </complexType>
2897 <element name="ServiceQuery" type="tns:ServiceQueryType" />
2898
2899 <element name="ServiceQueryResult">
2900   <complexType>
2901     <choice minOccurs="0" maxOccurs="unbounded">
2902       <element ref="rim:ObjectRef" />
2903       <element ref="rim:RegistryObject" />
2904       <element ref="rim:Service" />
2905     </choice>
2906   </complexType>
2907 </element>
2908
  
```

### 2909 Semantic Rules

- 2910 1. Let S denote the set of all persistent Service instances in the Registry. The following steps
- 2911 will eliminate instances in S that do not satisfy the conditions of the specified filters.
- 2912 a) If S is empty then continue to the next numbered rule.

- 2913 b) If a ServiceFilter is not specified then go to the next step; otherwise, let x be a service in  
 2914 S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then  
 2915 continue to the next numbered rule.
- 2916 c) If a ServiceBindingBranch is not specified then continue to the next numbered rule;  
 2917 otherwise, consider each ServiceBindingBranch element separately as follows:  
 2918 Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the  
 2919 member of SB. If a ServiceBindingFilter element is specified within the  
 2920 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If  
 2921 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.  
 2922 If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then  
 2923 continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch  
 2924 element separately as follows:  
 2925 Let sb be a remaining service binding in SB. Let SL be the set of all specification link  
 2926 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is  
 2927 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then  
 2928 remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove  
 2929 x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery  
 2930 element is specified within the SpecificationLinkBranch then let sl be a remaining  
 2931 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the  
 2932 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a  
 2933 specification link for some registry object in RO, then remove sl from SL. If SL is empty  
 2934 then remove sb from SB. If SB is empty then remove x from S. If S is empty then  
 2935 continue to the next numbered rule. If a RegistryEntryQuery element is specified within  
 2936 the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat  
 2937 RegistryEntryQuery element as follows: Let RE be the result set of the  
 2938 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some  
 2939 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If  
 2940 SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
- 2941 d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery  
 2942 over AE as explained in Section 8.2.3.
- 2943 2. If S is empty, then raise the warning: *service query result is empty*; otherwise set S to be the  
 2944 result of the ServiceQuery.
- 2945 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)  
 2946 within the RegistryResponse.

## 2947 Examples

2948

## 2949 8.2.13 Registry Filters

### 2950 Purpose

2951 To identify a subset of the set of all persistent instances of a given registry class.

### 2952 Definition

2953 `<complexType name="FilterType">`  
 2954

```

2955     <sequence>
2956         <element ref="tns:Clause" />
2957     </sequence>
2958 </complexType>
2959 <element name="RegistryObjectFilter" type="tns:FilterType" />
2960 <element name="RegistryEntryFilter" type="tns:FilterType" />
2961 <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
2962 <element name="RegistryPackageFilter" type="tns:FilterType" />
2963 <element name="OrganizationFilter" type="tns:FilterType" />
2964 <element name="ClassificationNodeFilter" type="tns:FilterType" />
2965 <element name="AssociationFilter" type="tns:FilterType" />
2966 <element name="ClassificationFilter" type="tns:FilterType" />
2967 <element name="ClassificationSchemeFilter" type="tns:FilterType" />
2968 <element name="ExternalLinkFilter" type="tns:FilterType" />
2969 <element name="ExternalIdentifierFilter" type="tns:FilterType" />
2970 <element name="SlotFilter" type="tns:FilterType" />
2971 <element name="AuditableEventFilter" type="tns:FilterType" />
2972 <element name="UserFilter" type="tns:FilterType" />
2973 <element name="SlotValueFilter" type="tns:FilterType" />
2974 <element name="PostalAddressFilter" type="tns:FilterType" />
2975 <element name="TelephoneNumberFilter" type="tns:FilterType" />
2976 <element name="ServiceFilter" type="tns:FilterType" />
2977 <element name="ServiceBindingFilter" type="tns:FilterType" />
2978 <element name="SpecificationLinkFilter" type="tns:FilterType" />
2979 <element name="LocalizedStringFilter" type="tns:FilterType" />
2980

```

## 2981 Semantic Rules

- 2982 1. The Clause element is defined in Section 8.2.14.
- 2983 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing  
2984 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in  
2985 [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set  
2986 of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the  
2987 Clause predicate.
- 2988 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing  
2989 SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in  
2990 [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter  
2991 returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*  
2992 for the Clause predicate.
- 2993 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing  
2994 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in  
2995 [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter  
2996 returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to  
2997 *True* for the Clause predicate.
- 2998 5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing  
2999 SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in  
3000 [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns  
3001 a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for  
3002 the Clause predicate.

- 3003 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing  
3004 SimpleClause shall identify a public attribute of the Organization or PostalAddress UML  
3005 classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The  
3006 OrganizationFilter returns a set of identifiers for Organization instances whose attribute  
3007 values evaluate to *True* for the Clause predicate.
- 3008 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any  
3009 containing SimpleClause shall identify a public attribute of the ClassificationNode UML  
3010 class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the  
3011 leftAttribute is the visible attribute “path” then if stringPredicate of the StringClause is not  
3012 “Equal” then raise exception: *classification node path attribute error*. The  
3013 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose  
3014 attribute values evaluate to *True* for the Clause predicate.
- 3015 8. For every AssociationFilter XML element, the leftArgument attribute of any containing  
3016 SimpleClause shall identify a public attribute of the Association UML class defined in  
3017 [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a  
3018 set of identifiers for Association instances whose attribute values evaluate to *True* for the  
3019 Clause predicate.
- 3020 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing  
3021 SimpleClause shall identify a public attribute of the Classification UML class defined in  
3022 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter  
3023 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*  
3024 for the Clause predicate.
- 3025 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any  
3026 containing SimpleClause shall identify a public attribute of the ClassificationNode UML  
3027 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The  
3028 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances  
3029 whose attribute values evaluate to *True* for the Clause predicate.
- 3030 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing  
3031 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in  
3032 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns  
3033 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the  
3034 Clause predicate.
- 3035 12. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing  
3036 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in  
3037 [ebRIM]. If not, raise exception: *external identifier attribute error*. The  
3038 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose  
3039 attribute values evaluate to *True* for the Clause predicate.
- 3040 13. For every SlotFilter XML element, the leftArgument attribute of any containing  
3041 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If  
3042 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot  
3043 instances whose attribute values evaluate to *True* for the Clause predicate.

- 3044 14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing  
3045 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in  
3046 [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter  
3047 returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to  
3048 *True* for the Clause predicate.
- 3049 15. For every UserFilter XML element, the leftArgument attribute of any containing  
3050 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If  
3051 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User  
3052 instances whose attribute values evaluate to *True* for the Clause predicate.
- 3053 16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one  
3054 SlotValue instance for each “value” in the “values” list of a Slot instance. The visible  
3055 attribute of SlotValue is “value”. It is a character string. The dynamic instances of SlotValue  
3056 are derived from the “values” attribute defined in ebRIM for a Slot instance. For every  
3057 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause  
3058 shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception:  
3059 *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value”  
3060 attribute evaluates to *True* for the Clause predicate.
- 3061 17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing  
3062 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in  
3063 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter  
3064 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*  
3065 for the Clause predicate.
- 3066 18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any  
3067 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML  
3068 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.  
3069 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances  
3070 whose attribute values evaluate to *True* for the Clause predicate.
- 3071 19. For every ServiceFilter XML element, the leftArgument attribute of any containing  
3072 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].  
3073 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for  
3074 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 3075 20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing  
3076 SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in  
3077 [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter  
3078 returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to  
3079 *True* for the Clause predicate.
- 3080 21. For every SpecificationLinkFilter XML element, the leftArgument attribute of any  
3081 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class  
3082 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The  
3083 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose  
3084 attribute values evaluate to *True* for the Clause predicate.

3085 22. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing  
 3086 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in  
 3087 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter  
 3088 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to  
 3089 *True* for the Clause predicate.

3090 **8.2.14 XML Clause Constraint Representation**

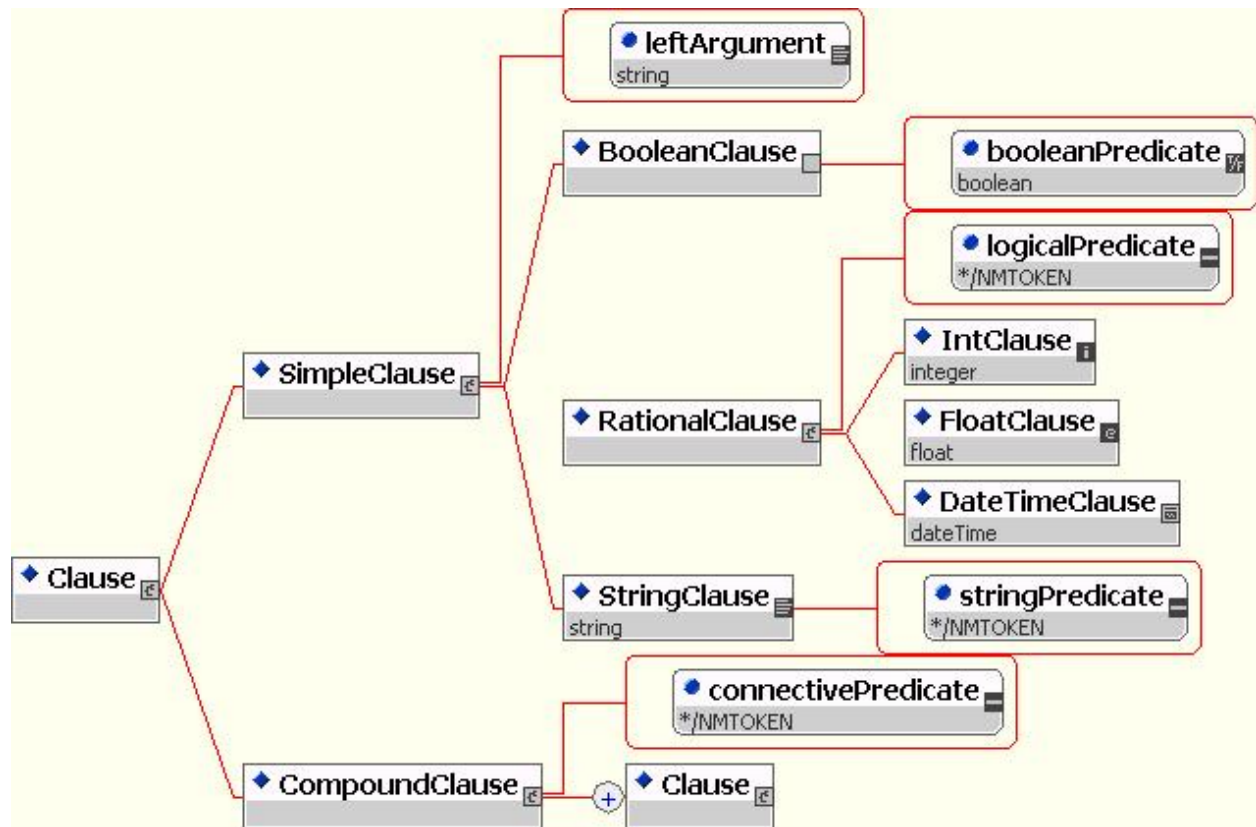
3091 **Purpose**

3092 The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses.  
 3093 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to  
 3094 simply as Clauses in this specification.

3095 **Conceptual Diagram**

3096 The following is a conceptual diagram outlining the Clause structure.

3097



3098  
3099

Figure 27: The Clause Structure

3100 **Semantic Rules**

3101 Predicates and Arguments are combined into a "LeftArgument - Predicate - RightArgument"  
 3102 format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.

3103 **SimpleClauses**

3104 A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the



3105 Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.  
 3106 SimpleClause is extended to support BooleanClause, StringClause, and RationalClause  
 3107 (abstract).

3108 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a  
 3109 boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-  
 3110 compare operations and a right argument as the element's text data. Rational number support is  
 3111 provided through a common RationalClause providing an enumeration of appropriate rational  
 3112 number compare operations, which is further extended to IntClause and FloatClause, each with  
 3113 appropriate signatures for the right argument.

#### 3114 CompoundClauses

3115 A CompoundClause contains two or more Clauses (Simple or Compound) and a connective  
 3116 predicate. This provides for arbitrarily complex Clauses to be formed.

#### 3117 **Definition**

```

3118 <element name = "Clause">
3119   <annotation>
3120     <documentation xml:lang = "en">
3121       The following lines define the XML syntax for Clause.
3122     </documentation>
3123   </annotation>
3124   <complexType>
3125     <choice>
3126       <element ref = "tns:SimpleClause"/>
3127       <element ref = "tns:CompoundClause"/>
3128     </choice>
3129   </complexType>
3130 </element>
3131 <element name = "SimpleClause">
3132   <complexType>
3133     <choice>
3134       <element ref = "tns:BooleanClause"/>
3135       <element ref = "tns:RationalClause"/>
3136       <element ref = "tns:StringClause"/>
3137     </choice>
3138     <attribute name = "leftArgument" use = "required" type =
3139       "string"/>
3140   </complexType>
3141 </element>
3142 <element name = "CompoundClause">
3143   <complexType>
3144     <sequence>
3145       <element ref = "tns:Clause" maxOccurs = "unbounded"/>
3146     </sequence>
3147     <attribute name = "connectivePredicate" use = "required">
3148       <simpleType>
3149         <restriction base = "NMTOKEN">
3150           <enumeration value = "And"/>
3151           <enumeration value = "Or"/>
3152         </restriction>
3153       </simpleType>
3154     </attribute>
3155   </complexType>
3156 </element>
3157 </element>
3158 </element>

```

```

3159     <element name = "BooleanClause">
3160         <complexType>
3161             <attribute name = "booleanPredicate" use = "required" type =
3162 "boolean"/>
3163         </complexType>
3164     </element>
3165     <element name = "RationalClause">
3166         <complexType>
3167             <choice>
3168                 <element ref = "tns:IntClause"/>
3169                 <element ref = "tns:FloatClause"/>
3170                 <element ref = "tns:DateTimeClause"/>
3171             </choice>
3172             <attribute name = "logicalPredicate" use = "required">
3173                 <simpleType>
3174                     <restriction base = "NMTOKEN">
3175                         <enumeration value = "LE"/>
3176                         <enumeration value = "LT"/>
3177                         <enumeration value = "GE"/>
3178                         <enumeration value = "GT"/>
3179                         <enumeration value = "EQ"/>
3180                         <enumeration value = "NE"/>
3181                     </restriction>
3182                 </simpleType>
3183             </attribute>
3184         </complexType>
3185     </element>
3186     <element name = "IntClause" type = "integer"/>
3187     <element name = "FloatClause" type = "float"/>
3188     <element name = "DateTimeClause" type = "dateTime"/>
3189
3190     <element name = "StringClause">
3191         <complexType>
3192             <simpleContent>
3193                 <extension base = "string">
3194                     <attribute name = "stringPredicate" use = "required">
3195                         <simpleType>
3196                             <restriction base = "NMTOKEN">
3197                                 <enumeration value = "Contains"/>
3198                                 <enumeration value = "-Contains"/>
3199                                 <enumeration value = "StartsWith"/>
3200                                 <enumeration value = "-StartsWith"/>
3201                                 <enumeration value = "Equal"/>
3202                                 <enumeration value = "-Equal"/>
3203                                 <enumeration value = "EndsWith"/>
3204                                 <enumeration value = "-EndsWith"/>
3205                             </restriction>
3206                         </simpleType>
3207                     </attribute>
3208                 </extension>
3209             </simpleContent>
3210         </complexType>
3211     </element>
3212

```

### 3213 Examples

3214 Simple BooleanClause: "Smoker" = True

```

3215
3216 <Clause>
3217   <SimpleClause leftArgument="Smoker">
3218     <BooleanClause booleanPredicate="True"/>
3219   </SimpleClause>
3220 </Clause>
3221

```

3222 **Simple StringClause: "Smoker" contains "mo"**

```

3223
3224 <Clause>
3225   <SimpleClause leftArgument = "Smoker">
3226     <StringClause stringPredicate = "Contains">mo</StringClause>
3227   </SimpleClause>
3228 </Clause>

```

3229 **Simple IntClause: "Age" >= 7**

```

3230
3231 <Clause>
3232   <SimpleClause leftArgument="Age">
3233     <RationalClause logicalPredicate="GE">
3234       <IntClause>7</IntClause>
3235     </RationalClause>
3236   </SimpleClause>
3237 </Clause>
3238

```

3239 **Simple FloatClause: "Size" = 4.3**

```

3240
3241 <Clause>
3242   <SimpleClause leftArgument="Size">
3243     <RationalClause logicalPredicate="Equal">
3244       <FloatClause>4.3</FloatClause>
3245     </RationalClause>
3246   </SimpleClause>
3247 </Clause>
3248

```

3249 **Compound with two Simples (( "Smoker" = False)AND("Age" =< 45))**

```

3250
3251 <Clause>
3252   <CompoundClause connectivePredicate="And">
3253     <Clause>
3254       <SimpleClause leftArgument="Smoker">
3255         <BooleanClause booleanPredicate="False"/>
3256       </SimpleClause>
3257     </Clause>
3258     <Clause>
3259       <SimpleClause leftArgument="Age">
3260         <RationalClause logicalPredicate="LE">
3261           <IntClause>45</IntClause>
3262         </RationalClause>
3263       </SimpleClause>
3264     </Clause>
3265   </CompoundClause>
3266 </Clause>
3267

```

## 3268 Compound with one Simple and one Compound

3269 ( ("Smoker" = False)And(("Age" =&lt; 45)Or("American"=True)) )

```

3270
3271 <Clause>
3272   <CompoundClause connectivePredicate="And">
3273     <Clause>
3274       <SimpleClause leftArgument="Smoker">
3275         <BooleanClause booleanPredicate="False"/>
3276       </SimpleClause>
3277     </Clause>
3278     <Clause>
3279       <CompoundClause connectivePredicate="Or">
3280         <Clause>
3281           <SimpleClause leftArgument="Age">
3282             <RationalClause logicalPredicate="LE">
3283               <IntClause>45</IntClause>
3284             </RationalClause>
3285           </SimpleClause>
3286         </Clause>
3287         <Clause>
3288           <SimpleClause leftArgument="American">
3289             <BooleanClause booleanPredicate="True"/>
3290           </SimpleClause>
3291         </Clause>
3292       </CompoundClause>
3293     </Clause>
3294   </CompoundClause>
3295 </Clause>
3296

```

3297 **8.3 SQL Query Support**

3298 The Registry may optionally support an SQL based query capability that is designed for Registry  
 3299 clients that demand more advanced query capability. The optional SQLQuery element in the  
 3300 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query  
 3301 language.

3302 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of  
 3303 the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database  
 3304 Language SQL [SQL], extended to include <sql invoked routines> (also known as  
 3305 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined  
 3306 in template form in Appendix D.3. The syntax of the Registry query language is defined by the  
 3307 BNF grammar in D.1.

3308 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use  
 3309 relational databases in a Registry implementation.

3310 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

3311 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational  
 3312 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as  
 3313 described in the following sections.

### 3314 **8.3.1.1 Class Binding**

3315 A subset of the class names defined in [ebRIM] map to table names that may be queried by an  
3316 SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an  
3317 SQL query.

3318 The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3  
3319 is as follows:

- 3320 • Classes that have concrete instances are mapped to relational tables. In addition entity classes  
3321 (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- 3322 • The intermediate classes in the inheritance hierarchy, namely RegistryObject and  
3323 RegistryEntry, map to relational views.
- 3324 • The names of relational tables and views are the same as the corresponding [ebRIM] class  
3325 name. However, the name binding is case insensitive.
- 3326 • Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in  
3327 Appendix D.3 where the column definitions are based on a subset of attributes defined for  
3328 that class in [ebRIM]. The attributes that map to columns include the inherited attributes for  
3329 the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed  
3330 which column definitions.

3331 An SQLQuery against a table not defined in Appendix D.3 may raise an error condition:  
3332 InvalidQueryException.

3333 The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn  
3334 definitions.

### 3335 **8.3.1.2 Primitive Attributes Binding**

3336 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same  
3337 way as column names in SQL. Again the exact attribute names are defined in the class  
3338 definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is  
3339 therefore valid for a query to contain attribute names that do not exactly match the case defined  
3340 in [ebRIM].

### 3341 **8.3.1.3 Reference Attribute Binding**

3342 A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a  
3343 class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject  
3344 class returns a reference to an instance of an AccessControlPolicy object.

3345 In such cases the reference maps to the i.d attribute for the referenced object. The name of the  
3346 resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data  
3347 type for the column is VARCHAR(64) as defined in Appendix D.3.

3348 When a reference attribute value holds a null reference, it maps to a null value in the SQL  
3349 binding and may be tested with the <null specification> (“IS [NOT] NULL” syntax) as defined  
3350 by [SQL].

3351 Reference attribute binding is a special case of a primitive attribute mapping.

### 3352 **8.3.1.4 Complex Attribute Binding**

3353 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of

3354 a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type  
3355 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

3356 The SQL query schema does not map complex attributes as columns in the table for the class for  
3357 which the attribute is defined. Instead the complex attributes are mapped to columns in the table  
3358 for the domain class that represents the data type for the complex attribute (e.g.  
3359 TelephoneNumber). A column links the row in the domain table to the row in the parent table  
3360 (e.g. User). An additional column named 'attribute\_name' identifies the attribute name in the  
3361 parent class, in case there are multiple attributes with the same complex attribute type.

3362 This mapping also easily allows for attributes that are a collection of a complex type. For  
3363 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the  
3364 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier  
3365 and an attribute\_name.

### 3366 **8.3.1.5 Binding of Methods Returning Collections**

3367 Several of the [ebRIM] classes define methods in addition to attributes, where these methods  
3368 return collections of references to instances of classes defined by [ebRIM]. For example, the  
3369 getPackages method of the ManagedObject class returns a Collection of references to instances  
3370 of Packages that the object is a member of.

3371 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in  
3372 Appendix D.3 such that these stored procedures return a collection of id attribute values. The  
3373 returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

3374 These stored procedures may be used as the right-hand-side of an SQL IN clause to test for  
3375 membership of an object in such collections of references.

### 3376 **8.3.2 Semantic Constraints On Query Syntax**

3377 This section defines simplifying constraints on the query syntax that cannot be expressed in the  
3378 BNF for the query syntax. These constraints must be applied in the semantic analysis of the  
3379 query.

- 3380 1. Class names and attribute names must be processed in a case insensitive manner.
- 3381 2. The syntax used for stored procedure invocation must be consistent with the syntax of an  
3382 SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 3383 3. For this version of the specification, the SQL select column list consists of exactly one  
3384 column, and must always be t.id, where t is a table reference in the FROM clause.
- 3385 4. Join operations must be restricted to simple joins involving only those columns that have an  
3386 index defined within the normative SQL schema. This constraint is to prevent queries that  
3387 may be computationally too expensive.

### 3388 **8.3.3 SQL Query Results**

3389 The result of an SQL query resolves to a collection of objects within the registry. It never  
3390 resolves to partial attributes. The objects related to the result set may be returned as an  
3391 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the  
3392 responseOption parameter specified by the client on the AdHocQueryRequest. The entire result

3393 set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 8.1.

### 3394 **8.3.4 Simple Metadata Based Queries**

3395 The simplest form of an SQL query is based upon metadata attributes specified for a single class  
3396 within [ebRIM]. This section gives some examples of simple metadata based queries.

3397 For example, to get the collection of ExtrinsicObjects whose name contains the word ‘Acme’  
3398 and that have a version greater than 1.3, the following query must be submitted:

```
3399
3400 SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
3401         eo.id = nm.parent AND
3402         eo.majorVersion >= 1 AND
3403         (eo.majorVersion >= 2 OR eo.minorVersion > 3);
3404
```

3405 Note that the query syntax allows for conjugation of simpler predicates into more complex  
3406 queries as shown in the simple example above.

### 3407 **8.3.5 RegistryObject Queries**

3408 The schema for the SQL query defines a special view called RegistryObject that allows doing a  
3409 polymorphic query against all RegistryObject instances regardless of their actual concrete type or  
3410 table name.

3411 The following example is the similar to that in Section 8.3.4 except that it is applied against all  
3412 RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id  
3413 for all qualifying RegistryObject instances whose name contains the word ‘Acme’ and whose  
3414 description contains the word “bicycle”.

```
3415
3416 SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND
3417         d.value LIKE '%bicycle%' AND
3418         ro.id = nm.parent AND ro.id = d.parent;
3419
```

### 3420 **8.3.6 RegistryEntry Queries**

3421 The schema for the SQL query defines a special view called RegistryEntry that allows doing a  
3422 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or  
3423 table name.

3424 The following example is the same as Section 8.3.4 except that it is applied against all  
3425 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id  
3426 for all qualifying RegistryEntry instances whose name contains the word ‘Acme’ and that have a  
3427 version greater than 1.3.

```
3428
3429 SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
3430         re.id = nm.parent AND
3431         re.majorVersion >= 1 AND
3432         (re.majorVersion >= 2 OR re.minorVersion > 3);
3433
```

### 3434 **8.3.7 Classification Queries**

3435 This section describes the various classification related queries that must be supported.

### 3436 **8.3.7.1 Identifying ClassificationNodes**

3437 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may  
 3438 also be identified as a path attribute that specifies an XPATH expression [XPT] from a root  
 3439 classification node to the specified classification node in the XML document that would  
 3440 represent the ClassificationNode tree including the said ClassificationNode.

### 3441 **8.3.7.2 Getting ClassificationSchemes**

3442 To get the collection of ClassificationSchemes the following query predicate must be supported:

```
3443
3444 SELECT scheme.id FROM ClassificationScheme scheme;
3445
```

3446 The above query returns all ClassificationSchemes. Note that the above query may also specify  
 3447 additional predicates (e.g. name, description etc.) if desired.

### 3448 **8.3.7.3 Getting Children of Specified ClassificationNode**

3449 To get the children of a ClassificationNode given the ID of that node the following style of query  
 3450 must be supported:

```
3451
3452 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
3453
```

3454 The above query returns all ClassificationNodes that have the node specified by <id> as their  
 3455 parent attribute.

### 3456 **8.3.7.4 Getting Objects Classified By a ClassificationNode**

3457 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the  
 3458 following style of query must be supported:

```
3459
3460 SELECT id FROM ExtrinsicObject
3461 WHERE
3462     id IN (SELECT classifiedObject FROM Classification
3463           WHERE
3464             classificationNode IN (SELECT id FROM ClassificationNode
3465                                   WHERE path = '/Geography/Asia/Japan'))
3466 AND
3467     id IN (SELECT classifiedObject FROM Classification
3468           WHERE
3469             classificationNode IN (SELECT id FROM ClassificationNode
3470                                   WHERE path = '/Industry/Automotive'))
3471
```

3472 The above query gets the collection of ExtrinsicObjects that are classified by the Automotive  
 3473 Industry and the Japan Geography. Note that according to the semantics defined for  
 3474 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by  
 3475 descendants of the specified ClassificationNodes.

### 3476 **8.3.7.5 Getting Classifications That Classify an Object**

3477 To get the collection of Classifications that classify a specified Object the following style of  
 3478 query must be supported:

```
3479
3480 SELECT id FROM Classification c
3481     WHERE c.classifiedObject = <id>;
3482
```



### 3483 8.3.8 Association Queries

3484 This section describes the various Association related queries that must be supported.

#### 3485 8.3.8.1 Getting All Association With Specified Object As Its Source

3486 To get the collection of Associations that have the specified Object as its source, the following  
3487 query must be supported:

```
3488 SELECT id FROM Association WHERE sourceObject = <id>
3489
3490
```

#### 3491 8.3.8.2 Getting All Association With Specified Object As Its Target

3492 To get the collection of Associations that have the specified Object as its target, the following  
3493 query must be supported:

```
3494 SELECT id FROM Association WHERE targetObject = <id>
3495
3496
```

#### 3497 8.3.8.3 Getting Associated Objects Based On Association Attributes

3498 To get the collection of Associations that have specified Association attributes, the following  
3499 queries must be supported:

3500 Select Associations that have the specified name.

```
3501 SELECT id FROM Association WHERE name = <name>
3502
3503
```

3504 Select Associations that have the specified association type, where association type is a string  
3505 containing the corresponding field name described in [ebRIM].

```
3506 SELECT id FROM Association WHERE
3507     associationType = <associationType>
3508
3509
```

#### 3510 8.3.8.4 Complex Association Queries

3511 The various forms of Association queries may be combined into complex predicates. The  
3512 following query selects Associations that have a specific sourceObject, targetObject and  
3513 associationType:

```
3514 SELECT id FROM Association WHERE
3515     sourceObject = <id1> AND
3516     targetObject = <id2> AND
3517     associationType = <associationType>;
3518
3519
```

### 3520 8.3.9 Package Queries

3521 To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
3522 SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));
3523
3524
```

#### 3525 8.3.9.1 Complex Package Queries

3526 The following query gets all Packages that a specified object belongs to, that are not deprecated  
3527 and where name contains "RosettaNet."

```

3528
3529
3530 SELECT id FROM Package p, Name n WHERE
3531     p.id IN (RegistryObject_packages(<id>)) AND
3532     nm.value LIKE '%RosettaNet%' AND nm.parent = p.id AND
3533     p.status <> 'Deprecated'

```

### 3534 8.3.10 ExternalLink Queries

3535 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is  
3536 specified:

```

3537
3538 SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
3539

```

3540 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is  
3541 specified:

```

3542
3543 SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
3544

```

#### 3545 8.3.10.1 Complex ExternalLink Queries

3546 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that  
3547 contain the word 'legal' in their description and have a URL for their externalURI.

```

3548
3549 SELECT id FROM ExternalLink WHERE
3550     id IN (RegistryObject_externalLinks(<id>)) AND
3551     description LIKE '%legal%' AND
3552     externalURI LIKE '%http://%'
3553

```

### 3554 8.3.11 Audit Trail Queries

3555 To get the complete collection of AuditableEvent objects for a specified ManagedObject, the  
3556 following query is specified:

```

3557
3558 SELECT id FROM AuditableEvent WHERE registryObject = <id>
3559

```

## 3560 8.4 Content Retrieval

3561 A client retrieves content via the Registry by sending the GetContentRequest to the  
3562 QueryManager. The GetContentRequest specifies a list of Object references for Objects that  
3563 need to be retrieved. The QueryManager returns the specified content by sending a  
3564 GetContentResponse message to the RegistryClient interface of the client. If there are no errors  
3565 encountered, the GetContentResponse message includes the specified content as additional  
3566 payloads within the message. In addition to the GetContentResponse payload, there is one  
3567 additional payload for each content that was requested. If there are errors encountered, the  
3568 RegistryResponse payload includes an error and there are no additional content specific  
3569 payloads.

### 3570 8.4.1 Identification Of Content Payloads

3571 Since the GetContentResponse message may include several repository items as additional  
3572 payloads, it is necessary to have a way to identify each payload in the message. To facilitate this

3573 identification, the Registry must do the following:

- 3574 • Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mime-
- 3575 part that contains the corresponding repository item for the ExtrinsicObject
- 3576 • In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the
- 3577 repository item in the Reference element for that object in the Manifest element of the
- 3578 ebXMLHeader.

#### 3579 8.4.2 GetContentResponse Message Structure

3580 The following message fragment illustrates the structure of the GetContentResponse Message  
 3581 that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs  
 3582 for the requested objects.

```

3583 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
3584
3585 --Boundary
3586 Content-ID: <GetContentRequest@example.com>
3587 Content-Type: text/xml
3588
3589 <?xml version="1.0" encoding="UTF-8"?>
3590 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
3591   xmlns:eb='http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
3592 <SOAP-ENV:Header>
3593
3594   ...ebMS header goes here if using ebMS
3595
3596 </SOAP-ENV:Header>
3597 <SOAP-ENV:Body>
3598
3599   ...ebMS manifest goes here if using ebMS
3600
3601 <?xml version="1.0" encoding="UTF-8"?>
3602
3603 <GetContentRequest>
3604   <ObjectRefList>
3605     <ObjectRef id="d8163dfb-f45a-4798-81d9-88aca29c24ff" .../>
3606     <ObjectRef id="212c3a78-1368-45d7-acc9-a935197e1e4f" .../>
3607   </ObjectRefList>
3608 </GetContentRequest>
3609
3610 </SOAP-ENV:Body>
3611 </SOAP-ENV:Envelope>
3612
3613 --Boundary
3614 Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
3615 Content-Type: text/xml
3616
3617 <?xml version="1.0" encoding="UTF-8"?>
3618 <CPP>
3619   .....
3620 </CPP>
3621
3622 --Boundary--
3623 Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f
3624 Content-Type: text/xml
3625
3626 <CPP>
3627   .....
3628 </CPP>
3629
3630 --Boundary--
3631
3632
3633
  
```

## 3634 **9 Registry Security**

3635 This chapter describes the security features of the ebXML Registry. It is assumed that the reader  
3636 is familiar with the security related classes in the Registry information model as described in  
3637 [ebRIM]. Security glossary terms can be referenced from RFC 2828.

### 3638 **9.1 Security Concerns**

3639 In the current version of this specification, we address data integrity and source integrity (item 1  
3640 in Appendix F.1). We have used a minimalist approach to address the access control concern as  
3641 in item 2 of Appendix F.1. Essentially, “any known entity (Submitting Organization) can publish  
3642 content and anyone can view published content.” The Registry information model has been  
3643 designed to allow more sophisticated security policies in future versions of this specification.

### 3644 **9.2 Integrity of Registry Content**

3645 It is assumed that most business registries do not have the resources to validate the veracity of  
3646 the content submitted to them. "The mechanisms described in this section can be used to ensure  
3647 that any tampering with the content submitted by a Submitting Organization can be detected.  
3648 Furthermore, these mechanisms support unambiguous identification of the Responsible  
3649 Organization for any registry content. The Registry Client has to sign the contents before  
3650 submission – otherwise the content will be rejected. Note that in the discussions in this section  
3651 we assume a Submitting Organization to be also the Responsible Organization. Future version of  
3652 this specification may provide more examples and scenarios where a Submitting Organization  
3653 and Responsible Organization are different.

#### 3654 **9.2.1 Message Payload Signature**

3655 The integrity of the Registry content requires that all submitted content be signed by the Registry  
3656 client. The signature on the submitted content ensures that:

- 3657
- 3658 • Any tampering of the content can be detected.
  - 3659 • The content’s veracity can be ascertained by its association with a specific Submitting  
3660 Organization.

3660 This section specifies the requirements for generation, packaging and validation of payload  
3661 signatures. A payload signature is packaged with the payload. Therefore the requirements apply  
3662 regardless of whether the Registry Client and the Registration Authority communicate over  
3663 vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML  
3664 Messaging Service does not specify the generation, validation and packaging of payload  
3665 signatures. The specification of payload signatures is left upto the application (such as Registry).  
3666 So the requirements on the payload signatures augment the [ebMS] specification.

#### 3667 **Use Case**

3668 This Use Case illustrates the use of header and payload signatures (we discuss header signatures  
3669 later).

- 3670
- 3671 • RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the  
3672 content along with the payload signature to the Registry.
  - RC2 (Registry Client 2) retrieves RC1’s content from the Registry.

- 3673 • RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves  
 3674 the content, the response from the Registration Authority to RC2 contains the following:
- 3675 – Payload containing the content that has been published by RC1.
  - 3676 – RC1’s payload signature (represented by a ds:Signature element) over RC1’s published  
 3677 content.
  - 3678 – The public key for validating RC1’s payload signature in ds:Signature element ( using the  
 3679 KeyInfo element as specified in [XMLDSIG] ) so RC2 can obtain the public key for  
 3680 signature (e.g. retrieve a certificate containing the public key for RC1).
  - 3681 – A ds:Signature element containing the header signature. Note that the Registration  
 3682 Authority (not RC1) generates this signature.

## 3683 9.2.2 Payload Signature Requirements

### 3684 9.2.2.1 Payload Signature Packaging Requirements

3685 A payload signature is represented by a ds:Signature element. The payload signature must be  
 3686 packaged with the payload as specified here. This packaging assumes that the payload is always  
 3687 signed.

- 3688 • The payload and its signature must be enclosed in a MIME multipart message with a  
 3689 Content-Type of multipart/Related.
- 3690 • The first body part must contain the XML signature as specified in Section 9.2.2.2, “Payload  
 3691 Signature Generation Requirements”.
- 3692 • The second through n<sup>th</sup> body part must be the content.

3693 The packaging of the payload signature with one payload is as follows:

```

3694 MIME-Version: 1.0
3695 Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
3696 Content-Description: ebXML Message
3697
3698 -- MIME_boundary
3699 Content-Type: text/xml; charset=UTF-8
3700 Content-Transfer-Encoding: 8bit
3701 Content-ID: http://claiming-it.com/claim061400a.xml
3702
3703 <?xml version='1.0' encoding="utf-8"?>
3704 <SOAP-ENV: Envelope>
3705 ...
3706 SOAP-ENV: Envelope>
3707
3708 --MIME_boundary
3709 Content-Type: multipart/Related; boundary=PAYLOAD_boundary
3710
3711 --PAYLOAD_boundary
3712 Content-Type: text/xml; charset=UTF-8
3713 Content-Transfer-Encoding: 8bit
3714 Content-ID: payload1
3715
```

```

3716 <ds:Signature>
3717     ... Payload signature
3718 </ds:Signature>
3719
3720 --PAYLOAD_boundary
3721 Content-Type: text/xml; charset=UTF-8
3722 Content-Transfer-Encoding: 8bit
3723 Content-ID: payload2
3724 <SubmitObjectsRequest>...</SubmitObjectsRequest>
3725 --MIME_boundary
3726

```

### 3727 9.2.2.2 Payload Signature Generation Requirements

3728 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in  
 3729 this section. Note: the “ds” name space reference is to <http://www.w3.org/2000/09/xmlsig#>

- 3730 • ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified  
 3731 using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a  
 3732 client may use any of these attributes. However, signing using the following Algorithm  
 3733 attribute: <http://www.w3.org/2000/09/xmlsig/#dsa-sha1> will allow interoperability with all  
 3734 XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this  
 3735 algorithm.

3736 The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The  
 3737 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported  
 3738 <http://www.w3.org/TR/2001/REC-xml-c14n-2001315>

- 3739 • One ds:Reference element to reference each of the payloads that needs to be signed must be  
 3740 created. The ds:Reference element:
  - 3741 – Must identify the payload to be signed using the URI attribute of the ds:Reference  
 3742 element.
  - 3743 – Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be  
 3744 support the following digest algorithm:  
 3745 <http://www.w3.org/2000/09/xmlsig/#sha1>
  - 3746 – Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

3747 The ds:SignedValue must be generated as specified in [XMLDSIG].

3748 The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject  
 3749 to the requirements stated in Section 9.4, “KeyDistribution and KeyInfo element”.

### 3750 9.2.2.3 Message Payload Signature Validation

3751 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

### 3752 9.2.2.4 Payload Signature Example

3753 The following example shows the format of the payload signature:

```

3754 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
3755 <ds:SignedInfo>
3756     <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmlsig/#dsa-sha1" />
3757

```

```
3758 <ds:CanonicalizationMethod>
3759     Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
3760 </ds:CanonicalizationMethod>
3761 <ds:Reference URI=#Payload1>
3762     <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmldsig#sha1">
3763         <ds:DigestValue> ... </ds:DigestValue>
3764     </ds:Reference>
3765 </ds:SignedInfo>
3766 <ds:SignatureValue> ... </ds:SignatureValue>
3767 </ds:Signature>
3768
```

## 3769 **9.3 Authentication**

3770 The Registry must be able to authenticate the identity of the Principal associated with client  
3771 requests. The identity of the Principal can be identified by verifying the message header  
3772 signature with the certificate of the Principal. The certificate may be in the message itself or  
3773 provided to the registry through means unspecified in this specification. If not provided in the  
3774 message, this specification does not specify how the Registry correlates a specific message with  
3775 a certificate. Authentication of each payload must also be possible by using the signature  
3776 associated with each payload. Authentication is also required to identify the "privileges" a  
3777 Principal is authorized ("authorization") to have with respect to specific objects in the Registry.

3778 The Registry must perform authentication on a per message basis. From a security point of view,  
3779 all messages are independent and there is no concept of a session encompassing multiple  
3780 messages or conversations. Session support may be added as an optimization feature in future  
3781 versions of this specification.

3782 It is important to note that the message header signature can only guarantee data integrity and it  
3783 may be used for Authentication knowing that it is vulnerable to replay types of attacks. True  
3784 support for authentication requires timestamps or nonce (nonrecurring series of numbers to  
3785 identify each message) that are signed.

### 3786 **9.3.1 Message Header Signature**

3787 Message headers are signed to provide data integrity while the message is in transit. Note that the  
3788 signature within the message header also signs the digests of the payloads.

#### 3789 Header Signature Requirements

3790 Message headers can be signed and are referred to as a header signature. This section specifies  
3791 the requirements for generation, packaging and validation of a header signature. These  
3792 requirements apply when the Registry Client and Registration Authority communicate using  
3793 vanilla SOAP with Attachments. When ebXML MS is used for communication, then the  
3794 message handler (i.e. [ebMS]) specifies the generation, packaging and validation of XML  
3795 signatures in the SOAP header. Therefore the header signature requirements do not apply when  
3796 the ebXML MS is used for communication. However, payload signature generation requirements  
3797 (specified elsewhere in this document) do apply whether vanilla SOAP with Attachments or  
3798 ebXML MS is used for communication.

### 3799 9.3.1.1 Packaging Requirements

3800 A header signature is represented by a ds:Signature element. The ds:Signature element generated  
3801 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature  
3802 element in the SOAP header field is shown below.  
3803

```
3804 MIME-Version: 1.0
3805 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3806 Content-Description: ebXML Message
3807
3808 -- MIME_boundary
3809 Content-Type: text/xml; charset=UTF-8
3810 Content-Transfer-Encoding: 8bit
3811 Content-ID: http://claiming-it.com/claim061400a.xml
3812
3813 <?xml version='1.0' encoding="utf-8"?>
3814 <SOAP-ENV:Envelope
3815     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3816     <SOAP-ENV:Header>
3817         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3818             ...signature over soap envelope
3819         </ds:Signature>
3820     </SOAP-ENV:Header>
3821     <SOAP-ENV:Body>
3822         ...
3823     </SOAP-ENV:Body>
3824 </SOAP-ENV:Envelope>
3825
```

### 3826 9.3.1.2 Header Signature Generation Requirements

3827 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in  
3828 this section. A ds:Signature element contains:

- 3829 • ds:SignedInfo
- 3830 • ds:SignatureValue
- 3831 • ds:KeyInfo

3832 The ds:SignedInfo element must be generated as follows:

- 3833 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified  
3834 using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,  
3835 a client must be capable of signing using only the following Algorithm attribute:  
3836 <http://www.w3.org/2000/09/xmldsig/#dsa-sha1> This algorithm is being chosen because all  
3837 XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
- 3838 2. The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The  
3839 following Canonicalization algorithm (specified in [XMLDSIG] ) must be supported:  
3840 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>



- 3841 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.  
 3842 This signs the entire ds:Reference element and:
- 3843 – Must include the following ds:Transform:  
 3844 <http://www.w3.org/2000/09/xmldsig#enveloped-signature>  
 3845 This ensures that the signature (which is embedded in the <SOAP-ENV:Header>  
 3846 element) is not included in the signature calculation.
  - 3847 – Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the  
 3848 ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) .  
 3849 The URI attribute must be “”.
  - 3850 – Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support  
 3851 the following digest algorithm: <http://www.w3.org/2000/09/xmldsig/#sha1>
  - 3852 – Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

3853 The ds:SignedValue must be generated as specified in [XMLDSIG].

3854 The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in  
 3855 Section 9.4, “KeyDistribution and KeyInfo element”.

### 3856 9.3.1.3 Header Signature Validation Requirements

3857 The ds:Signature element for the ebXML message header must be validated by the recipient as  
 3858 specified by [XMLDSIG].

### 3859 9.3.1.4 Header Signature Example

3860 The following example shows the format of a header signature:

```

3861 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3862   <ds:SignedInfo>
3863     <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>
3864     <ds:CanonicalizationMethod>
3865       Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
3866     </ds:CanonicalizationMethod>
3867     <ds:Reference URI= "">
3868       <ds:Transform>
3869         http://www.w3.org/2000/09/xmldsig#enveloped-signature
3870       </ds:Transform>
3871       <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
3872       <ds:DigestValue> ... </ds:DigestValue>
3873     </ds:Reference>
3874   </ds:SignedInfo>
3875   <ds:SignatureValue> ... </ds:SignatureValue>
3876 </ds:Signature>
3877
3878
```

## 3879 9.4 Key Distribution and KeyInfo Element

3880 To validate a signature, the recipient of the signature needs the public key corresponding to the  
 3881 signer’s public key. The participants may use the KeyInfo field of ds:Signature, or distribute the

3882 public keys out-of-band. In this section we consider the case when the public key is sent in the  
3883 KeyInfo field. The following use cases need to be handled:

- 3884 • Registration Authority needs the public key of the Registry Client to validate the signature
- 3885 • Registry Client needs the public key of the Registration Authority to validate the Registry's  
3886 signature.
- 3887 • Registry Client RC1 needs the public key of Registry Client (RC2) to validate the content  
3888 signed by RC1.
- 3889 • [XMLDSIG] provides a *ds:KeyInfo* element that can be used to pass the recipient  
3890 information for retrieving the public key. *ds:KeyInfo* is an optional element as specified in  
3891 [XMLDSIG]. This field together with the procedures outlined in this section is used to  
3892 securely pass the public key to a recipient. *ds:KeyInfo* can be used to pass information such  
3893 as keys, certificates, names etc. The intended usage of KeyInfo field is to send the X509  
3894 Certificate, and subsequently extract the public key from the certificate. Therefore, the  
3895 KeyInfo field must contain a X509 Certificate as specified in [XMLDSIG], if the KeyInfo  
3896 field is present.

3897 The following assumptions are also made:

- 3898 1. A Certificate is associated both with the Registration Authority and a Registry Client.
- 3899 2. A Registry Client registers its certificate with the Registration Authority. The mechanism  
3900 used for this is not specified here.
- 3901 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local  
3902 key store. The mechanism is not specified here.

3903 Couple of scenarios on the use of KeyInfo field is in Appendix F.8.

## 3904 **9.5 Confidentiality**

### 3905 **9.5.1 On-the-wire Message Confidentiality**

3906 It is suggested but not required that message payloads exchanged between clients and the  
3907 Registry be encrypted during transmission. This specification does not specify how payload  
3908 encryption is to be done.

### 3909 **9.5.2 Confidentiality of Registry Content**

3910 In the current version of this specification, there are no provisions for confidentiality of Registry  
3911 content. All content submitted to the Registry may be discovered and read by any client. This  
3912 implies that the Registry and the client need to have an a priori agreement regarding encryption  
3913 algorithm, key exchange agreements, etc. This service is not addressed in this specification.

## 3914 **9.6 Authorization**

3915 The Registry must provide an authorization mechanism based on the information model defined  
3916 in [ebRIM]. In this version of the specification the authorization mechanism is based on a default  
3917 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of  
3918 this specification will allow for custom Access Control Policies to be defined by the Submitting  
3919 Organization. The authorization is going to be applied on a specific set of privileges. A

3920 privilege is the ability to carry a specific action.

### 3921 **9.6.1 Actions**

#### 3922 Life Cycle Actions

3923 submitObjects

3924 updateObjects

3925 addSlots

3926 removeSlots

3927 approveObjects

3928 deprecateObjects

3929 removeObjects

#### 3930 Read Actions

3931 The various getXXX() methods in QueryManagement Service.

## 3932 **9.7 Access Control**

3933 The Registry must create a default AccessControlPolicy object that grants the default  
 3934 permissions to Registry users based upon their assigned role. The following table defines the  
 3935 Permissions granted by the Registry to the various pre-defined roles for Registry users based  
 3936 upon the default AccessControlPolicy. Note that we have "ContentOwner" as a role. This role  
 3937 maps to the Submitting Organization in the current version of the specification.

3938 **Table 11: Default Access Control Policies**

<b>Role</b>	<b>Permissions</b>
ContentOwner	Access to <i>all</i> methods on Registry Objects that are owned by the ContentOwner.
RegistryAdministrator	Access to <i>all</i> methods on <i>all</i> Registry Objects
RegistryGuest	Access to <i>all</i> read-only (getXXX) methods on <i>all</i> Registry Objects (read-only access to all content).

3939 The Registry must implement the default AccessControlPolicy and associate it with all Objects  
 3940 in the Registry. The following list summarizes the default role-based AccessControlPolicy:

- 3941 • Anyone can publish content, but needs to be a Registered User
- 3942 • Anyone can access the content without requiring authentication
- 3943 • The ContentOwner has access to all methods for Registry Objects created by it.
- 3944 • The RegistryAdministrator has access to all methods on all Registry Objects
- 3945 • Unauthenticated clients can access all read-only (getXXX) methods

- 3946 • At the time of content submission, the Registry must assign the default ContentOwner role to  
3947 the Submitting Organization (SO) as authenticated by the credentials in the submission  
3948 message. In the current version of this specification, the Submitting Organization will be the  
3949 DN as identified by the certificate
- 3950 • Clients that browse the Registry need not use certificates. The Registry must assign the  
3951 default RegistryGuest role to such clients.

## 3952 **Appendix A Web Service Architecture**

### 3953 **A.1 Registry Service Abstract Specification**

3954 The normative definition of the Abstract Registry Service in WSDL is defined at the following  
3955 location on the web:

3956 <http://www.oasis-open.org/committees/regrep/documents/2.0/services/Registry.wsdl>

### 3957 **A.2 Registry Service SOAP Binding**

3958 The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined  
3959 at the following location on the web:

3960 <http://www.oasis-open.org/committees/regrep/documents/2.0/services/SOAPBinding.wsdl>

3961

## 3962 **Appendix B ebXML Registry Schema Definitions**

### 3963 **B.1 RIM Schema**

3964 The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the  
3965 following location on the web:

3966 <http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd>

### 3967 **B.2 Query Schema**

3968 The normative XML Schema definition for the XML query syntax for the registry service  
3969 interface can be found at the following location on the web:

3970 <http://www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd>

### 3971 **B.3 Registry Services Interface Schema**

3972 The normative XML Schema definition that defines the XML requests and responses supported  
3973 by the registry service interfaces in this document can be found at the following location on the  
3974 web:

3975 <http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd>

### 3976 **B.4 Examples of Instance Documents**

3977 A growing number of non-normative XML instance documents that conform to the normative  
3978 Schema definitions described earlier may be found at the following location on the web:

3979 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/>

3980

## 3981 **Appendix C Interpretation of UML Diagrams**

3982 This section describes in *abstract terms* the conventions used to define ebXML business process  
3983 description in UML.

### 3984 **C.1 UML Class Diagram**

3985 A UML class diagram is used to describe the Service Interfaces required to implement an  
3986 ebXML Registry Services and clients. The UML class diagram contains:

- 3987
- 3988 1. A collection of UML interfaces where each interface represents a Service Interface for a  
3989 Registry service.
  - 3990 2. Tabular description of methods on each interface where each method represents an  
3991 Action (as defined by [ebCPP]) within the Service Interface representing the UML  
3992 interface.
  - 3993 3. Each method within a UML interface specifies one or more parameters, where the type of  
3994 each method argument represents the ebXML message type that is exchanged as part of  
3995 the Action corresponding to the method. Multiple arguments imply multiple payload  
3996 documents within the body of the corresponding ebXML message.

### 3997 **C.2 UML Sequence Diagram**

3998 A UML sequence diagram is used to specify the business protocol representing the interactions  
3999 between the UML interfaces for a Registry specific ebXML business process. A UML sequence  
4000 diagram provides the necessary information to determine the sequencing of messages, request to  
4001 response association as well as request to error response association.

4002 Each sequence diagram shows the sequence for a specific conversation protocol as method calls  
4003 from the requestor to the responder. Method invocation may be synchronous or asynchronous  
4004 based on the UML notation used on the arrow-head for the link. A half arrow-head represents  
4005 asynchronous communication. A full arrow-head represents synchronous communication.

4006 Each method invocation may be followed by a response method invocation from the responder to  
4007 the requestor to indicate the ResponseName for the previous Request. Possible error response is  
4008 indicated by a conditional response method invocation from the responder to the requestor. See  
4009 Figure 7 on page 27 for an example.

## 4010 Appendix D SQL Query

### 4011 D.1 SQL Query Syntax Specification

4012 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The  
 4013 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax  
 4014 conforms to the <query specification>, modulo the restrictions identified below:

- 4015 1. A <select list> may contain at most one <select sublist>.
- 4016 2. In a <select list> must be is a single column whose data type is UUID, from the table in the  
 4017 <from clause>.
- 4018 3. A <derived column> may not have an <as clause>.
- 4019 4. <table expression> does not contain the optional <group by clause> and <having clause>  
 4020 clauses.
- 4021 5. A <table reference> can only consist of <table name> and <correlation name>.
- 4022 6. A <table reference> does not have the optional AS between <table name> and  
 4023 <correlation name>.
- 4024 7. There can only be one <table reference> in the <from clause>.
- 4025 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows  
 4026 for the right hand side of the <in predicate> to be limited to a restricted <query  
 4027 specification> as defined above.
- 4028 9. A <search condition> within the <where clause> may not include a <query expression>.
- 4029 10. Simple joins are allowed only if they are based on indexed columns within the relational  
 4030 schema.
- 4031 11. The SQL query syntax allows for the use of <sql invoked routines> invocation from  
 4032 [SQL/PSM] as the RHS of the <in predicate>.

### 4033 D.2 Non-Normative BNF for Query Syntax Grammar

4034 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as  
 4035 an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative  
 4036 syntax. For the normative syntax rules see Appendix D.1.

```

4037 /*****
4038  * The Registry Query (Subset of SQL-92) grammar starts here
4039  *****/
4040
4041 RegistryQuery = SQLSelect [ ";" ]
4042
4043 SQLSelect = "SELECT" [ "DISTINCT" ] SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
4044
4045 SQLSelectCols = ID
4046
4047 SQLTableList = SQLTableRef
4048
4049 SQLTableRef = ID
4050
4051 SQLWhere = "WHERE" SQLOrExpr
4052
4053 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
4054
4055
  
```



```

4056 SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*
4057
4058 SQLNotExpr = [ "NOT" ] SQLCompareExpr
4059
4060 SQLCompareExpr =
4061     (SQLColRef "IS") SQLIsClause
4062     | SQLSumExpr [ SQLCompareExprRight ]
4063
4064
4065 SQLCompareExprRight =
4066     SQLLikeClause
4067     | SQLInClause
4068     | SQLCompareOp SQLSumExpr
4069
4070 SQLCompareOp =
4071     "="
4072     | "<>"
4073     | ">"
4074     | ">="
4075     | "<"
4076     | "<="
4077
4078 SQLInClause = [ "NOT" ] "IN" "(" SQLValueList ")"
4079
4080 SQLValueList = SQLValueElement ( "," SQLValueElement )*
4081
4082 SQLValueElement = "NULL" | SQLSelect
4083
4084 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
4085
4086 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
4087
4088 SQLPattern = STRING_LITERAL
4089
4090 SQLLiteral =
4091     STRING_LITERAL
4092     | INTEGER_LITERAL
4093     | FLOATING_POINT_LITERAL
4094
4095 SQLColRef = SQLLvalue
4096
4097 SQLLvalue = SQLLvalueTerm
4098
4099 SQLLvalueTerm = ID ( "." ID )*
4100
4101 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
4102
4103 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr )*
4104
4105 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
4106
4107 SQLTerm = "(" SQLOrExpr ")"
4108     | SQLColRef
4109     | SQLLiteral
4110
4111 INTEGER_LITERAL = ([ "0"-"9" ])+
4112
4113 FLOATING_POINT_LITERAL =
4114     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ (EXPONENT)?
4115     | "." ([ "0"-"9" ])+ (EXPONENT)?
4116     | ([ "0"-"9" ])+ EXPONENT
4117     | ([ "0"-"9" ])+ (EXPONENT)?
4118
4119 EXPONENT = [ "e", "E" ] ( [ "+", "-" ] )? ([ "0"-"9" ])+
4120
4121 STRING_LITERAL: "'" (~[''])* ( "''" (~[''])* )* "'"
4122
4123 ID = ( <LETTER> )+ ( "-" | "$" | "#" | <DIGIT> | <LETTER> )*
4124 LETTER = [ "A"-"Z", "a"-"z" ]
4125 DIGIT = [ "0"-"9" ]

```

### 4126 **D.3 Relational Schema For SQL Queries**

4127 The normative Relational Schema definition for SQL queries can be found at the following  
4128 location on the web:

4129 <http://www.oasis-open.org/committees/regrep/documents/2.0/sql/database.sql>

4130

4131 The stored procedures that must be supported by the SQL query feature are defined at the following  
4132 location on the web:

4133 <http://www.oasis-open.org/committees/regrep/documents/2.0/sql/storedProcedures.sql>

4134

## 4135 **Appendix E Non-normative Content Based Ad Hoc Queries**

4136 The Registry SQL query capability supports the ability to search for content based not only on  
4137 metadata that catalogs the content but also the data contained within the content itself. For  
4138 example it is possible for a client to submit a query that searches for all Collaboration Party  
4139 Profiles that define a role named “seller” within a RoleName element in the CPP document itself.  
4140 Currently content-based query capability is restricted to XML content.

### 4141 **E.1 Automatic Classification of XML Content**

4142 Content-based queries are indirectly supported through the existing classification mechanism  
4143 supported by the Registry.

4144 A submitting organization may define logical indexes on any XML schema or DTD when it is  
4145 submitted. An instance of such a logical index defines a link between a specific attribute or  
4146 element node in an XML document tree and a ClassificationNode in a classification scheme  
4147 within the registry.

4148 The registry utilizes this index to automatically classify documents that are instances of the  
4149 schema at the time the document instance is submitted. Such documents are classified according  
4150 to the data contained within the document itself.

4151 Such automatically classified content may subsequently be discovered by clients using the  
4152 existing classification-based discovery mechanism of the Registry and the query facilities of the  
4153 QueryManager.

4154 [Note] This approach is conceptually similar to the way databases support  
4155 indexed retrieval. DBAs define indexes on tables in the schema. When  
4156 data is added to the table, the data gets automatically indexed.

### 4157 **E.2 Index Definition**

4158 This section describes how the logical indexes are defined in the SubmittedObject element  
4159 defined in the Registry Schema. The complete Registry Schema is available via hyperlinks in  
4160 Appendix B.

4161 A SubmittedObject element for a schema or DTD may define a collection of  
4162 ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList  
4163 is ignored if the content being submitted is not of the SCHEMA objectType.

4164 The ClassificationIndex element inherits the attributes of the base class RegistryObject in  
4165 [ebRIM]. It then defines specialized attributes as follows:

- 4166 1. classificationNode: This attribute references a specific ClassificationNode by its ID.
- 4167 2. contentIdentifier: This attribute identifies a specific data element within the document  
4168 instances of the schema using an XPATH expression as defined by [XPT].

### 4169 **E.3 Example Of Index Definition**

4170 To define an index that automatically classifies a CPP based upon the roles defined within its  
4171 RoleName elements, the following index must be defined on the CPP schema or DTD:  
4172

```

4173 <ClassificationIndex
4174     classificationNode='id-for-role-classification-scheme'
4175     contentIdentifier='/Role//RoleName'
4176 />
4177

```

## 4178 E.4 Proposed XML Definition

```

4179
4180 <!--
4181 A ClassificationIndexList is specified on ExtrinsicObjects of objectType
4182 'Schema' to define an automatic Classification of instance objects of the
4183 schema using the specified classificationNode as parent and a
4184 ClassificationNode created or selected by the object content as selected
4185 by the contentIdentifier
4186 -->
4187 <!ELEMENT ClassificationIndex EMPTY>
4188 <!ATTLIST ClassificationIndex
4189     %ObjectAttributes;
4190     classificationNode IDREF #REQUIRED
4191     contentIdentifier CDATA #REQUIRED
4192 >
4193
4194 <!-- ClassificationIndexList contains new ClassificationIndexes -->
4195 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
4196

```

## 4197 E.5 Example of Automatic Classification

4198 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the CPP is  
4199 submitted it will automatically be classified by two ClassificationNodes named “buyer” and  
4200 “seller” that are both children of the ClassificationNode (e.g. a node named Role) specified in the  
4201 classificationNode attribute of the ClassificationIndex. If either of the two ClassificationNodes  
4202 named “buyer” and “seller” did not previously exist, the LifeCycleManager would automatically  
4203 create these ClassificationNodes.

## 4204 **Appendix F Security Implementation Guideline**

4205 This section provides a suggested blueprint for how security processing may be implemented in  
4206 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have  
4207 different implementations as long as they support the default security roles and authorization  
4208 rules described in this document.

### 4209 **F.1 Security Concerns**

4210 The security risks broadly stem from the following concerns. After a description of these  
4211 concerns and potential solutions, we identify the concerns that we address in the current  
4212 specification

4213 1. Is the content of the registry (data) trustworthy?

4214 a) How to make sure “what is in the registry” is “what is put there” by a submitting  
4215 organization? This concern can be addressed by ensuring that the publisher is  
4216 authenticated using digital signature (Source Integrity), message is not corrupted during  
4217 transfer using digital signature (Data Integrity), and the data is not altered by  
4218 unauthorized subjects based on access control policy (Authorization)

4219 b) How to protect data while in transmission?

4220 Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data  
4221 Confidentiality that can be addressed by encrypting the data in transmission. How to  
4222 protect against a replay attack?

4223 c) Is the content up to date? The versioning as well as any time stamp processing, when  
4224 done securely will ensure the “latest content” is guaranteed to be the latest content.

4225 d) How to ensure only bona fide responsible organizations add contents to registry?  
4226 Ensuring Source Integrity (as in 1a).

4227 e) How to ensure that bona fide publishers add contents to registry only at authorized  
4228 locations? (System Integrity)

4229 f) What if the publishers deny modifying certain content after-the-fact? To prevent this  
4230 (Nonrepudiation) audit trails may be kept which contain signed message digests.

4231 g) What if the reader denies getting information from the registry?

4232 2. How to provide selective access to registry content? The broad answer is, by using an access  
4233 control policy – applies to (a), (b), and (c) directly.

4234 a) How does a submitting organization restrict access to the content to only specific registry  
4235 readers?

4236 b) How can a submitting organization allow some “partners” (fellow publishers) to modify  
4237 content?

4238 c) How to provide selective access to partners the registry usage data?

4239 d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw  
4240 failure of the registry security components? The solution to this problem is by having  
4241 System Integrity.

4242 e) Data confidentiality of RegistryObject

- 4243 3. How do we make “who can see what” policy itself visible to limited parties, even excluding  
4244 the administrator (self & confidential maintenance of access control policy). By making sure  
4245 there is an access control policy for accessing the policies themselves.
- 4246 4. How to transfer credentials? The broad solution is to use credentials assertion (such as being  
4247 worked on in SAML). Currently, Registry does not support the notion of a session.  
4248 Therefore, some of these concerns are not relevant to the current specification.
- 4249 a) How to transfer credentials (authorization/authentication) to federated registries?  
4250 b) How do aggregators get credentials (authorization/authentication) transferred to them?  
4251 c) How to store credentials through a session?

## 4252 **F.2 Authentication**

- 4253 1. As soon as a message is received, the first work is the authentication. A principal object is  
4254 created.
- 4255 2. If the message is signed, it is verified (including the validity of the certificate) and the DN of  
4256 the certificate becomes the identity of the principal. Then the Registry is searched for the  
4257 principal and if found, the roles and groups are filled in.
- 4258 3. If the message is not signed, an empty principal is created with the role RegistryGuest. This  
4259 step is for symmetry and to decouple the rest of the processing.
- 4260 4. Then the message is processed for the command and the objects it will act on.

## 4261 **F.3 Authorization**

4262 For every object, the access controller will iterate through all the AccessControlPolicy objects  
4263 with the object and see if there is a chain through the permission objects to verify that the  
4264 requested method is permitted for the Principal. If any of the permission objects which the object  
4265 is associated with has a common role, or identity, or group with the principal, the action is  
4266 permitted.

## 4267 **F.4 Registry Bootstrap**

4268 When a Registry is newly created, a default Principal object should be created with the identity  
4269 of the Registry Admin’s certificate DN with a role RegistryAdmin. This way, any message  
4270 signed by the Registry Admin will get all the privileges.

4271 When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the  
4272 default AccessControlPolicy. This includes the creation of the necessary Permission instances as  
4273 well as the Privileges and Privilege attributes.

## 4274 **F.5 Content Submission – Client Responsibility**

4275 The Registry client must sign the contents before submission – otherwise the content will be  
4276 rejected.

## 4277 **F.6 Content Submission – Registry Responsibility**

- 4278 1. As with any other request, the client will first be authenticated. In this case, the Principal  
4279 object will get the DN from the certificate.
- 4280 2. As per the request in the message, the RegistryEntry will be created.
- 4281 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 4282 4. If a principal with the identity of the SO is not available, an identity object with the SO's DN  
4283 is created.
- 4284 5. A principal with this identity is created.

## 4285 **F.7 Content Delete/Deprecate – Client Responsibility**

4286 The Registry client must sign the header before submission, for authentication purposes;  
4287 otherwise, the request will be rejected

## 4288 **F.8 Content Delete/Deprecate – Registry Responsibility**

- 4289 1. As with any other request, the client will first be authenticated. In this case, the Principal  
4290 object will get the DN from the certificate. As there will be a principal with this identity in  
4291 the Registry, the Principal object will get all the roles from that object
- 4292 2. As per the request in the message (delete or deprecate), the appropriate method in the  
4293 RegistryObject class will be accessed.
- 4294 3. The access controller performs the authorization by iterating through the Permission objects  
4295 associated with this object via the singleton default AccessControlPolicy.
- 4296 4. If authorization succeeds then the action will be permitted. Otherwise an error response is  
4297 sent back with a suitable AuthorizationException error message.

## 4298 **F.9 Using ds:KeyInfo Field**

4299 Two typical usage scenarios for ds:KeyInfo are described below.

### 4300 **Scenario 1**

- 4301 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
- 4302 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
- 4303 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
- 4304 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature
- 4305 5. Registration Authority validates the header signature using the public key from the  
4306 certificate.
- 4307 6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the  
4308 certificate from the KeyInfo field of the payload signature. Note that this step is not an  
4309 essential one if the onus of validation is that of the eventual user, another Registry Client, of  
4310 the content.

### 4311 **Scenario 2**

- 4312 1. RC1 signs the payload and SOAP envelope using its private key and publishes to the  
4313 Registry.
- 4314 2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
- 4315 3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature.  
4316 This step is required in addition to step 2 because when RC2 retrieves content, it should see  
4317 RC1's signature with the payload.
- 4318 4. RC2 retrieves content from the Registry.
- 4319 5. Registration Authority signs the SOAP envelope using its private key. Registration Authority  
4320 sends RC1's content and the RC1's signature (signed by RC1).
- 4321 6. Registration Authority need not send its certificate in the KeyInfo field since RC2 is assumed  
4322 to have obtained the Registration Authority's certificate out of band and installed it in its  
4323 local key store.
- 4324 7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the  
4325 Registration Authority's signature.
- 4326 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates  
4327 the signature on the payload.



## 4328 **Appendix G Native Language Support (NLS)**

### 4329 **G.1 Definitions**

4330 Although this section discusses only character set and language, the following terms have to be  
4331 defined clearly.

#### 4332 **G.1.1 Coded Character Set (CCS):**

4333 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of  
4334 CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

#### 4335 **G.1.2 Character Encoding Scheme (CES):**

4336 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are  
4337 ISO-2022, UTF-8.

#### 4338 **G.1.3 Character Set (charset):**

- 4339 • charset is a set of rules for mapping from a sequence of octets to a sequence of  
4340 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- 4341 • A list of registered character sets can be found at [IANA].

### 4342 **G.2 NLS And Request / Response Messages**

4343 For the accurate processing of data in both registry client and registry services, it is essential to  
4344 know which character set is used. Although the body part of the transaction may contain the  
4345 charset in xml encoding declaration, registry client and registry services shall specify charset  
4346 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a  
4347 text/xml entity is received with the charset parameter omitted, MIME processors and XML  
4348 processors MUST use the default charset value of "us-ascii". For example:

```
4349 Content-Type: text/xml; charset=ISO-2022-JP  
4350  
4351
```

4352 Also, when an application/xml entity is used, the charset parameter is optional, and registry  
4353 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which  
4354 directly address this contingency.

4355 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

### 4356 **G.3 NLS And Storing of RegistryObject**

4357 This section provides NLS guidelines on how a registry should store RegistryObject instances.

4358 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple  
4359 locales. Thus there is no language or character set associated with a specific RegistryObject  
4360 instance.

4361 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows.  
4362 Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in

4363 RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed  
4364 as LocalizedString sub-elements within the XML element representing the I18N capable  
4365 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a  
4366 specific locale. Each LocalizedString element has a charset and lang attribute as well as a value  
4367 attribute of type string.

### 4368 **G.3.1 Character Set of *LocalizedString***

4369 The character set used by a locale specific String (LocalizedString) is defined by the charset  
4370 attribute. It is highly recommended to use UTF-8 or UTF-16 for maximum inter-operability.

### 4371 **G.3.2 Language Information of *LocalizedString***

4372 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

## 4373 **G.4 NLS And Storing of Repository Items**

4374 This section provides NLS guidelines on how a registry should store repository items.  
4375 While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is  
4376 always associated with a single repository item. The repository item may be in a single locale or  
4377 may be in multiple locales. This specification does not specify the repository item.

### 4378 **G.4.1 Character Set of Repository Items**

4379 The MIME Content-Type mime header for the mime multi-part containing the repository  
4380 item MAY contain a "charset" attribute that specifies the character set used by the repository  
4381 item. For example:

```
4382 Content-Type: text/xml; charset="UTF-8"
```

4385 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset  
4386 of a repository item must be preserved as it is originally specified in the transaction.

### 4387 **G.4.2 Language information of repository item**

4388 The Content-language mime header for the mime bodypart containing the repository item may  
4389 specify the language for a locale specific repository item. The value of the Content-language  
4390 mime header property must conform to [RFC 1766].

4391 This document currently specifies only the method of sending the information of character set  
4392 and language, and how it is stored in a registry. However, the language information may be used  
4393 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a  
4394 language negotiation procedure, like registry client is asking a favorite language for messages  
4395 from registry services, could be another functionality for the future revision of this document.

## 4396 **Appendix H Registry Profile**

4397 Every registry must support exactly one Registry Profile. The Registry Profile is an XML  
4398 document that describes the capabilities of the registry. The profile document must conform to  
4399 the RegistryProfile element as described in the Registry Services Interface schema defined in  
4400 Appendix B. The registry must make the RegistryProfile accessible over HTTP protocol via a  
4401 URL. The URL must conform to the pattern:

4402 <http://<base url>/registryProfile>

4403

## 4404 10 References

- 4405 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 4406 [ebRIM] ebXML Registry Information Model version 2.0
- 4407 <http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf>
- 4408 [ebRIM Schema] ebXML Registry Information Model Schema
- 4409 <http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd>
- 4410 [ebBPSS] ebXML Business Process Specification Schema
- 4411 <http://www.ebxml.org/specs>
- 4412 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 4413 <http://www.ebxml.org/specs/>
- 4414 [ebMS] ebXML Messaging Service Specification, Version 1.0
- 4415 <http://www.ebxml.org/specs/>
- 4416 [XPT] XML Path Language (XPath) Version 1.0
- 4417 <http://www.w3.org/TR/xpath>
- 4418 [SQL] Structured Query Language (FIPS PUB 127-2)
- 4419 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 4420 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 4421 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 4422 [IANA] IANA (Internet Assigned Numbers Authority).
- 4423 Official Names for Character Sets, ed. Keld Simonsen et al.
- 4424 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 4425 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
- 4426 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
- 4427 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 4428 [RFC 2119] IETF (Internet Engineering Task Force). RFC 2119
- 4429 [RFC 2130] IETF (Internet Engineering Task Force). RFC 2130
- 4430 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
- 4431 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
- 4432 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 4433 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
- 4434 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
- 4435 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 4436 [RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
- 4437 Internet Security Glossary, ed. R. Shirey. May 2000.
- 4438 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html>
- 4439 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
- 4440 XML Media Types, ed. M. Murata. 2001.
- 4441 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
- 4442 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
- 4443 <http://www.w3.org/TR/REC-xml>
- 4444 [UUID] DCE 128 bit Universal Unique Identifier
- 4445 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)
- 4446 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
- 4447 [WSDL] W3C Note. Web Services Description Language (WSDL) 1.1

- 4448 <http://www.w3.org/TR/wsdl>
- 4449 [SOAP11] W3C Note. Simple Object Access Protocol, May 2000,
- 4450 <http://www.w3.org/TR/SOAP>
- 4451 [SOAPAt] W3C Note: SOAP with Attachments, Dec 2000,
- 4452 <http://www.w3.org/TR/SOAP-attachments>
- 4453 [XMLDSIG] XML-Signature Syntax and Processing,
- 4454 <http://www.w3.org/TR/2001/PR-xmlsig-core-20010820/>

## 4455 **11 Disclaimer**

4456 The views and specification expressed in this document are those of the authors and are not  
4457 necessarily those of their employers. The authors and their employers specifically disclaim  
4458 responsibility for any problems arising from correct or incorrect implementation or use of this  
4459 design.

**4460 12 Contact Information**

## 4461 Team Leader

4462 Name: Lisa J. Carnahan  
4463 Company: National Institute of Standards and Technology  
4464 Street: 100 Bureau Drive, Stop 8970  
4465 City, State, Postal Code: Gaithersburg, Md. 20899  
4466 Country: USA  
4467 Phone: 301-975-3362  
4468 Email: lisa.carnahan@nist.gov

4469

## 4470 Editor

4471 Name: Anne A. Fischer  
4472 Company: Drummond Group, Inc.  
4473 Street: 4700 Bryant Irvin Ct., Suite 303  
4474 City, State, Postal Code: Fort Worth, Texas 76107-7645  
4475 Country: USA  
4476 Phone: 817-371-2367  
4477 Email: anne@drummondgroup.com

4478

## 4479 Technical Editor

4480 Name: Farrukh S. Najmi  
4481 Company: Sun Microsystems  
4482 Street: 1 Network Dr., MS BUR02-302  
4483 City, State, Postal Code: Burlington, MA 01803-0902  
4484 Country: USA  
4485 Phone: 781-442-0703  
4486 Email: najmi@east.sun.com

4487

## 4488 **13 Copyright Statement**

4489 Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

4490 **Copyright (C) The Organization for the Advancement of Structured Information**  
4491 **Standards [OASIS], 2002. All Rights Reserved.**

4492 This document and translations of it may be copied and furnished to others, and derivative works  
4493 that comment on or otherwise explain it or assist in its implementation may be prepared, copied,  
4494 published and distributed, in whole or in part, without restriction of any kind, provided that the  
4495 above copyright notice and this paragraph are included on all such copies and derivative works.  
4496 However, this document itself may not be modified in any way, such as by removing the  
4497 copyright notice or references to OASIS, except as needed for the purpose of developing OASIS  
4498 specifications, in which case the procedures for copyrights defined in the OASIS Intellectual  
4499 Property Rights document must be followed, or as required to translate it into languages other  
4500 than English.

4501 The limited permissions granted above are perpetual and will not be revoked by OASIS or its  
4502 successors or assigns.

4503 This document and the information contained herein is provided on an "AS IS" basis and OASIS  
4504 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT  
4505 LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN  
4506 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF  
4507 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

4508 OASIS takes no position regarding the validity or scope of any intellectual property or other  
4509 rights that might be claimed to pertain to the implementation or use of the technology described  
4510 in this document or the extent to which any license under such rights might or might not be  
4511 available; neither does it represent that it has made any effort to identify any such rights.  
4512 Information on OASIS's procedures with respect to rights in OASIS specifications can be found  
4513 at the OASIS website. Copies of claims of rights made available for publication and any  
4514 assurances of licenses to be made available, or the result of an attempt made to obtain a general  
4515 license or permission for the use of such proprietary rights by implementors or users of this  
4516 specification, can be obtained from the OASIS Executive Director.

4517 OASIS invites any interested party to bring to its attention any copyrights, patents or patent  
4518 applications, or other proprietary rights which may cover technology that may be required to  
4519 implement this specification. Please address the information to the OASIS Executive Director.