



Creating A Single Global Electronic Market

1

OASIS/ebXML Registry Services Specification v1.02 DRAFT

OASIS/ebXML Registry Technical Committee

26 November 2001

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

3 **1 Status of this Document**

4

5 Distribution of this document is unlimited.

6

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

8

9 ***This version:***

10 <http://www.oasis-open.org/committees/regrep/document/rsV1-01.pdf>

11

12 ***Latest version:***

13 <http://www.oasis-open.org/committees/regrep/documents/rsV1-01.pdf>

14 **2 OASIS/ebXML Registry Technical Committee**

15 The OASIS/ebXML Registry Technical Committee has approved this document, as a DRAFT
16 Specification, in its current form. At the time of this approval the following were members of
17 the OASIS/ebXML Registry Technical Committee.

18 Lisa Carnahan, US NIST - Chair
19 Len Gallagher, US NIST
20 Nikola Stojanovic, Encoda Systems
21 Suresh Damodaran, Sterling Commerce
22 Farrukh Najmi, SUN
23 Bruce Bargmeyer, EPA
24 Kathryn Breininger, Boeing
25 Dan Chang, IBM
26 Joseph M. Chiusano, LMI
27 Suresh Damodaran, Sterling Commerce
28 Mike DeNicola, Fujitsu
29 John Evdemon, Vitria Technologies
30 Anne Fischer, Drummond Group
31 Sally Fuger, AIAG
32 Len Gallagher, NIST
33 Michael Joya, XMLGlobal
34 Chaemee Kim, KTNET
35 Jong Kim, InnoDigital
36 Kyu-Chul Lee, Chungnam National University
37 Joel Munter, Intel
38 Farrukh Najmi, Sun Microsystems
39 Joel Neu, Vitria Technologies
40 Sanjay Patil, IONA
41 Waqar Sadiq, EDS
42 Neal Smith, ChevronTexaco
43 Nikola Stojanovic, Encoda Systems Inc.
44 David Webber, XMLGlobal
45 Prasad Yendluri, webmethods
46 Yutaka Yoshida, Sun Microsystems

47	Table of Contents	
48	1 Status of this Document	3
49	2 OASIS/ebXML Registry Technical Committee	4
50	Table of Contents	5
51	Table of Figures	9
52	Table of Tables	10
53	3 Introduction	11
54	3.1 Summary of Contents of Document.....	11
55	3.2 General Conventions.....	11
56	3.3 Audience.....	11
57	4 Design Objectives	12
58	4.1 Goals.....	12
59	4.2 Caveats and Assumptions.....	12
60	5 System Overview	13
61	5.1 What The ebXML Registry Does.....	13
62	5.2 How The ebXML Registry Works.....	13
63	5.2.1 Schema Documents Are Submitted.....	13
64	5.2.2 Business Process Documents Are Submitted.....	13
65	5.2.3 Seller's Collaboration Protocol Profile Is Submitted.....	13
66	5.2.4 Buyer Discovers The Seller.....	13
67	5.2.5 CPA Is Established.....	14
68	5.3 Registry Users.....	14
69	5.4 Where the Registry Services May Be Implemented.....	15
70	5.5 Implementation Conformance.....	15
71	5.5.1 Conformance as an ebXML Registry.....	15
72	5.5.2 Conformance as an ebXML Registry Client.....	16
73	6 ebXML Registry Architecture	17
74	6.1 Registry Service Description.....	17
75	6.2 Abstract Registry Service.....	17
76	6.3 Concrete Registry Services.....	18
77	6.3.1 SOAP Binding.....	19
78	6.3.2 ebXML Message Service Binding.....	19
79	6.4 LifeCycleManager Interface.....	20
80	6.5 QueryManager Interface.....	21
81	6.6 Registry Clients.....	21
82	6.6.1 Registry Client Description.....	21
83	6.6.2 Registry Communication Bootstrapping.....	22
84	6.6.3 RegistryClient Interface.....	23
85	6.6.4 Registry Response Class Hierarchy.....	23
86	6.7 Interoperability Requirements.....	24
87	6.7.1 Client Interoperability.....	24
88	6.7.2 Inter-Registry Cooperation.....	24
89	7 Life Cycle Management Service	25
90	7.1 Life Cycle of a Repository Item.....	25
91	7.2 RegistryObject Attributes.....	25
92	7.3 The Submit Objects Protocol.....	25
93	7.3.1 Universally Unique ID Generation.....	26

94	7.3.2	ID Attribute And Object References.....	26
95	7.3.3	Audit Trail.....	27
96	7.3.4	Submitting Organization.....	27
97	7.3.5	Error Handling.....	27
98	7.3.6	Sample SubmitObjectsRequest.....	28
99	7.4	The Update Objects Protocol.....	31
100	7.4.1	Audit Trail.....	31
101	7.4.2	Submitting Organization.....	32
102	7.4.3	Error Handling.....	32
103	7.5	The Add Slots Protocol.....	32
104	7.6	The Remove Slots Protocol.....	33
105	7.7	The Approve Objects Protocol.....	33
106	7.7.1	Audit Trail.....	34
107	7.7.2	Submitting Organization.....	34
108	7.7.3	Error Handling.....	34
109	7.8	The Deprecate Objects Protocol.....	35
110	7.8.1	Audit Trail.....	35
111	7.8.2	Submitting Organization.....	35
112	7.8.3	Error Handling.....	35
113	7.9	The Remove Objects Protocol.....	36
114	7.9.1	Deletion Scope DeleteRepositoryItemOnly.....	36
115	7.9.2	Deletion Scope DeleteAll.....	36
116	7.9.3	Error Handling.....	37
117	8	Query Management Service.....	38
118	8.1	Ad Hoc Query Request/Response.....	38
119	8.1.1	Query Response Options.....	39
120	8.2	Filter Query Support.....	40
121	8.2.1	FilterQuery.....	42
122	8.2.2	RegistryObjectQuery.....	43
123	1.1.1	56	
124	8.2.3	RegistryEntryQuery.....	56
125	8.2.4	AuditableEventQuery.....	59
126	8.2.5	ClassificationNodeQuery.....	62
127	8.2.6	ClassificationSchemeQuery.....	67
128	8.2.7	RegistryPackageQuery.....	68
129	8.2.8	ExtrinsicObjectQuery.....	70
130	8.2.9	OrganizationQuery.....	71
131	8.2.10	ServiceQuery.....	74
132	8.2.11	Registry Filters.....	76
133	8.2.12	XML Clause Constraint Representation.....	79
134	8.3	SQL Query Support.....	83
135	8.3.1	SQL Query Syntax Binding To [ebRIM].....	83
136	8.3.2	Semantic Constraints On Query Syntax.....	85
137	8.3.3	SQL Query Results.....	85
138	8.3.4	Simple Metadata Based Queries.....	85
139	8.3.5	RegistryObject Queries.....	85
140	8.3.6	RegistryEntry Queries.....	86
141	8.3.7	Classification Queries.....	86
142	8.3.8	Association Queries.....	87

143	8.3.9 Package Queries.....	88
144	8.3.10 ExternalLink Queries.....	88
145	8.3.11 Audit Trail Queries.....	88
146	8.4 Content Retrieval.....	88
147	8.4.1 Identification Of Content Payloads.....	89
148	8.4.2 GetContentResponse Message Structure.....	89
149	9 Registry Security.....	91
150	9.1 Security Concerns.....	91
151	9.2 Integrity of Registry Content.....	92
152	9.2.1 Message Payload Signature.....	92
153	9.2.2 Payload Signature Requirements.....	93
154	9.3 Authentication.....	94
155	9.3.1 Message Header Signature.....	95
156	9.4 Key Distribution and KeyInfo Element.....	97
157	9.4.1 Using ds:KeyInfo Field.....	97
158	9.5 Confidentiality.....	98
159	9.5.1 On-the-wire Message Confidentiality.....	98
160	9.5.2 Confidentiality of Registry Content.....	98
161	9.6 Authorization.....	98
162	9.6.1 Actions.....	99
163	9.7 Access Control.....	Error! Bookmark not defined.
164	Appendix A Web Service Architecture.....	100
165	A.1 Registry Service Abstract Specification.....	100
166	A.2 Registry Service SOAP Binding.....	101
167	Appendix B ebXML Registry Schema Definitions.....	104
168	B.1 RIM Schema.....	104
169	B.2 Query Schema.....	112
170	B.3 Registry Services Interface Schema.....	122
171	Appendix C Interpretation of UML Diagrams.....	127
172	C.1 UML Class Diagram.....	127
173	C.2 UML Sequence Diagram.....	127
174	Appendix D SQL Query.....	128
175	D.1 SQL Query Syntax Specification.....	128
176	D.2 Non-Normative BNF for Query Syntax Grammar.....	128
177	D.3 Relational Schema For SQL Queries.....	129
178	Appendix E Non-normative Content Based Ad Hoc Queries.....	139
179	E.1.1 Automatic Classification of XML Content.....	139
180	E.1.2 Index Definition.....	139
181	E.1.3 Example Of Index Definition.....	139
182	E.1.4 Proposed XML Definition.....	140
183	E.1.5 Example of Automatic Classification.....	140
184	Appendix F Security Implementation Guideline.....	141
185	F.1 Authentication.....	141
186	F.2 Authorization.....	141
187	F.3 Registry Bootstrap.....	141
188	F.4 Content Submission – Client Responsibility.....	141
189	F.5 Content Submission – Registry Responsibility.....	141
190	F.6 Content Delete/Deprecate – Client Responsibility.....	142

191	F.7	Content Delete/Deprecate – Registry Responsibility	142
192	Appendix G	Native Language Support (NLS)	143
193	G.1	Definitions.....	143
194	G.1.1	Coded Character Set (CCS):.....	143
195	G.1.2	Character Encoding Scheme (CES):.....	143
196	G.1.3	Character Set (charset):.....	143
197	G.2	NLS And Request / Response Messages	143
198	G.3	NLS And Storing of RegistryObject.....	143
199	G.3.1	Character Set of <i>LocalizedString</i>	144
200	G.3.2	Language Information of <i>LocalizedString</i>	144
201	G.4	NLS And Storing of Repository Items.....	144
202	G.4.1	Character Set of Repository Items	144
203	G.4.2	Language information of repository item	144
204	Appendix H	Terminology Mapping.....	145
205	References	146	
206	Disclaimer	147	
207	Contact Information	148	
208	Copyright Statement	149	
209			

210 **Table of Figures**

211	Figure 1: Actor Relationships	15
212	Figure 2: ebXML Registry Service Architecture	17
213	Figure 3: The Abstract ebXML Registry Service	18
214	Figure 4: A Concrete ebXML Registry Service	18
215	Figure 5: Registry Architecture Supports Flexible Topologies	22
216	Figure 6: Registry Response Class Hierarchy	24
217	Figure 7: Life Cycle of a Repository Item	25
218	Figure 8: Submit Objects Sequence Diagram	26
219	Figure 9: Update Objects Sequence Diagram	31
220	Figure 10: Add Slots Sequence Diagram	33
221	Figure 11: Remove Slots Sequence Diagram	33
222	Figure 12: Approve Objects Sequence Diagram	34
223	Figure 13: Deprecate Objects Sequence Diagram	35
224	Figure 14: Remove Objects Sequence Diagram	37
225	Figure 15: Submit Ad Hoc Query Sequence Diagram	39
226	Figure 16: Example ebRIM Binding	41
227	Figure 17: ebRim Binding for RegistryObjectQuery	44
228	Figure 18: ebRIM Binding for RegistryEntryQuery	56
229	Figure 19: ebRim binding for AuditableEventQuery	59
230	Figure 20: ebRim binding for ClassificationNodeQuery	62
231	Figure 21: ebRIM Binding for ClassificationSchemeQuery	67
232	Figure 22: ebRim binding for RegistryPackageQuery	68
233	Figure 23: ebRIM Binding for ExtrinsicObjectQuery	70
234	Figure 24: ebRim Binding for OrganizationQuery	71
235	Figure 25: ebRIM Binding for ServiceQuery	75
236	Figure 26: The Clause base structure	80
237		

238	Table of Tables	
239	Table 1: Registry Users	14
240	Table 2: LifeCycle Manager Summary	21
241	Table 3: Query Manager	21
242	Table 4: RegistryClient Summary	23
243	Table 5 Submit Objects Error Handling.....	27
244	Table 6: Update Objects Error Handling.....	32
245	Table 7: Approve Objects Error Handling	34
246	Table 8: Deprecate Objects Error Handling.....	36
247	Table 9: Remove Objects Error Handling.....	37
248	Table 10: Path Filter Expressions for Use Cases	65
249	Table 11: Default Access Control Policies	99
250	Table 12: Terminology Mapping Table	145
251		

252 **3 Introduction**

253 **3.1 Summary of Contents of Document**

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

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

259 **3.2 General Conventions**

260 The following conventions are used throughout this document:

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

263 The term "*repository item*" is used to refer to an object that has been submitted to a Registry for
264 storage and safekeeping (e.g. an XML document or a DTD). Every repository item is described
265 by a RegistryEntry instance.

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

268 *Capitalized Italic* words are defined in the ebXML Glossary.

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

272 **3.3 Audience**

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

- 274 • Implementers of ebXML Registry Services
- 275 • Implementers of ebXML Registry Clients

276 **3.3.1.1 Related Documents**

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

- 278 a) *ebXML Registry Information Model* [ebRIM]
- 279 b) *ebXML Message Service Specification* [ebMS]
- 280 c) *ebXML Business Process Specification Schema* [ebBPM]
- 281 d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

282 **4 Design Objectives**

283 **4.1 Goals**

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

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

289 **4.2 Caveats and Assumptions**

290 The Registry Services specification is first in a series of phased deliverables. Later versions of
291 the document will include additional functionality planned for future development. It is
292 assumed that:

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

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

300 **5 System Overview**

301 **5.1 What The ebXML Registry Does**

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

306 **5.2 How The ebXML Registry Works**

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

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

316 **5.2.1 Schema Documents Are Submitted**

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

320 **5.2.2 Business Process Documents Are Submitted**

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

324 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

329 **5.2.4 Buyer Discovers The Seller**

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

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

335 **5.2.5 CPA Is Established**

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

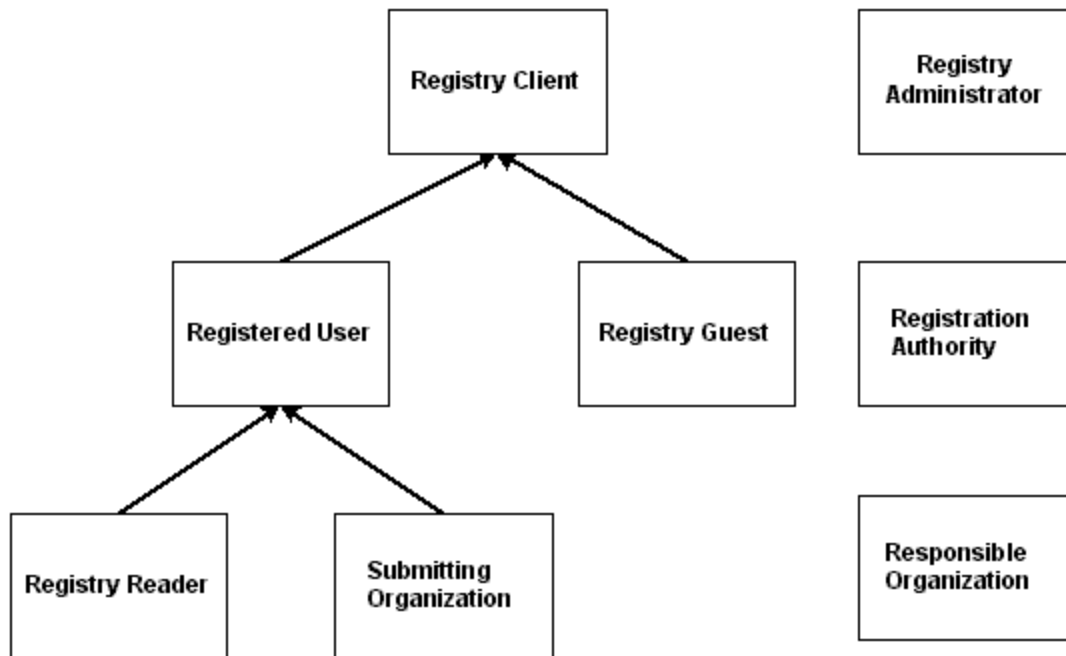
340 Once the seller accepts the CPA, the parties may begin to conduct B2B transactions as defined
 341 by [ebMS].

342 **5.3 Registry Users**

343 We describe the actors who use the registry from the point of view of security and analyze the
 344 security concerns of the registry below. This analysis leads up to the security requirements for
 345 V2. Some of the actors are defined in Section 9.4.1 of [ebRS]. Note that same entity may take on
 346 multiple roles. For example, a Registration Authority and Registry Administrator may have the
 347 same identity.
 348

Table 1: Registry Users

Actor	Function	ISO/IEC 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY be the same as Registration Authority
Registered User	Has a <i>contract</i> 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 <i>contract</i> with Registration Authority. Does not have to be authenticated for Registry access. Cannot change contents of the Registry (MAY be permitted to <i>read</i> some RegistryObjects.)		Note that a Registry Guest is <i>not</i> a Registry Reader.
Submitting Organization	A Registered User who does lifecycle operations on permitted RegistryObjects.	Submitting Organization (SO)	
Registry Reader	A Registered User who has only <i>read</i> access		
Responsible Organization	Creates Registry Objects	Responsible Organization (RO)	RO MAY have the same identity as SO
Registry Client	Registered User or Registered Guest		



349
350

Figure 1: Actor Relationships

351 **Note:**

352 In V2, we are not distinguishing between Submitting Organization and Responsible
353 Organization.

354 Registration of a user happens out-of-band for V2.

355 For V2 we do not distinguish between Registry Administrator and Registration Authority.

356 **5.4 Where the Registry Services May Be Implemented**

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

359 **5.5 Implementation Conformance**

360 An implementation is a *conforming* ebXML Registry if the implementation meets the conditions
361 in Section 5.4.1. An implementation is a conforming ebXML Registry Client if the
362 implementation meets the conditions in Section 5.4.2. An implementation is a conforming
363 ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to
364 the conditions of Section 5.4.1 and Section 5.4.2. An implementation shall be a conforming
365 ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and
366 Registry Client.

367 **5.5.1 Conformance as an ebXML Registry**

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

- 370 1. Conforms to *the ebXML Registry Information Model [ebRIM]*.
371 2. Supports the syntax and semantics of the Registry Interfaces and Security Model.
372 3. Supports the defined ebXML Registry DTD (Appendix A)
373 4. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

374 **5.5.2 Conformance as an ebXML Registry Client**

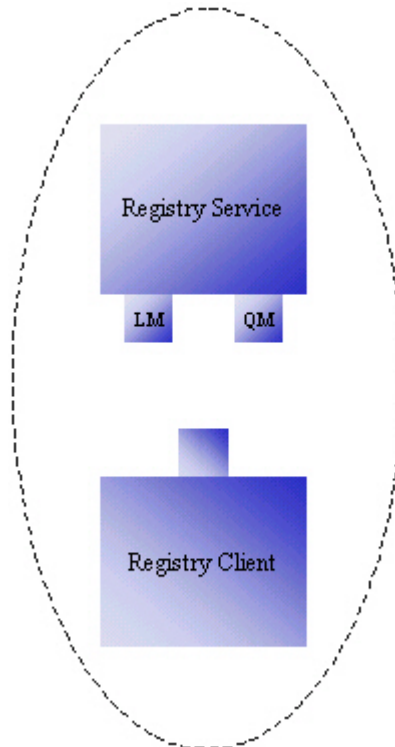
375 An implementation conforms to this specification, as an ebXML Registry Client if it meets the
376 following conditions:

- 377 1. Supports the ebXML CPA and bootstrapping process.
- 378 2. Supports the syntax and the semantics of the Registry Client Interfaces.
- 379 3. Supports the defined ebXML Error Message DTD.
- 380 4. Supports the defined ebXML Registry DTD.

381

382 **6 ebXML Registry Architecture**

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



386
387

Figure 2: ebXML Registry Service Architecture

388 **6.1 Registry Service Description**

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

- 392 • A Life Cycle Management interface that provides a collection of methods for managing
393 objects within the Registry.
- 394 • A Query Management Interface that controls the discovery and retrieval of information from
395 the Registry.

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

400 **6.2 Abstract Registry Service**

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

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

405 The abstract registry service neither defines any specific implementation for the ebXML
 406 Registry, nor does it specify any specific protocols used by the registry. Such implementation
 407 details are described by concrete registry services that realize the abstract registry service.
 408 The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide
 409 two key functional interfaces called **QueryManager**¹ (QM) and **LifeCycleManager**²
 410 (LM).



411
 412 **Figure 3: The Abstract ebXML Registry Service**

413 Appendix A.1 describes the abstract service definition in the Web Service Description Language
 414 (WSDL) syntax.

415 6.3 Concrete Registry Services

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

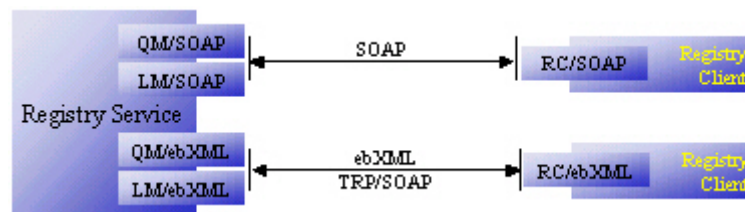
- 418 • Implementations of the interfaces defined by the abstract registry service.
- 419 • Bindings of these concrete interfaces to specific communication protocols.

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

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

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

425



426
 427 **Figure 4: A Concrete ebXML Registry Service**

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

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

¹ Known as ObjectQueryManager in V1.0

² Known as ObjectManager in V1.0

435 6.3.1 SOAP Binding

436 6.3.1.1 WSDL Terminology Primer

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

444 An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract
 445 service is bound to a specific protocol by providing a **binding** definition for each abstract port
 446 type that defines additional protocols specific details. Finally, a concrete **service** definition is
 447 defined as a collection of **ports**, where each port simply adds address information such as a
 448 URL for each concrete port.

449 6.3.1.2 Concrete Binding for SOAP

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

- 452 • A single service element with name “RegistryService” defines the concrete SOAP binding
 453 for the registry service.
- 454 • The service element includes two port definitions, where each port corresponds with one of
 455 the interfaces defined for the abstract registry service. Each port includes an HTTP URL for
 456 accessing that port.
- 457 • Each port definition also references a binding element, one for each interface defined in the
 458 WSDL for the abstract registry service.

```

459
460 <service name = "RegistryService">
461     <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
462         <soap:address location = "http://your_URL_to_your_QueryManager"/>
463     </port>
464
465     <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
466         <soap:address location = "http://your_URL_to_your_QueryManager"/>
467     </port>
468 </service>
469
  
```

470 The complete WSDL description for the SOAP binding is described in Appendix A.2

471 6.3.2 ebXML Message Service Binding

472 6.3.2.1 Service and Action Elements

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

- 475 • The value of the Service element in the MessageHeader is an ebXML Registry Service
 476 interface name (e.g., “LifeCycleManager”). The type attribute of the Service element should
 477 have a value of “ebXMLRegistry”.
- 478 • The value of the Action element in the MessageHeader is an ebXML Registry Service
 479 method name (e.g., “submitObjects”).

```

480
481 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
482 <eb:Action>submitObjects</eb:Action>
  
```

483

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

487 **6.3.2.2 Synchronous and Asynchronous Responses**

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

489 **Asynchronous response**

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

- 493 • MessageHeader;
- 494 • RegistryResponse element with empty content (e.g., **NO AdHocQueryResponse**);
 - 495 – status attribute with value **unavailable**.

496 The Registry delivers the actual Registry response element with non-empty content
497 asynchronously at a later time. The delivery is accomplished by the Registry invoking the
498 onResponse method on the RegistryClient interface as implemented by the registry client
499 application. The onResponse method includes a RegistryResponse element which has a complete
500 as defined by the Synchronous response section below. The Registry response includes:

- 501 • MessageHeader;
- 502 • RegistryResponse element including;
 - 503 – Status attribute (success, failure, warning);
 - 504 – Optional RegistryErrorList.

505 **Synchronous response**

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

- 508 • MessageHeader;
- 509 • RegistryResponse element including;
 - 510 – Status attribute (success, failure, warning);
 - 511 – Optional RegistryErrorList.

512 **6.3.2.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

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

524 **6.4 LifeCycleManager Interface**

525 This is the interface exposed by the Registry Service that implements the object life cycle

526 management functionality of the Registry. Its' methods are invoked by the Registry Client. For
 527 example, the client may use this interface to submit objects, to classify and associate objects and
 528 to deprecate and remove objects. For this specification the semantic meaning of submit, classify,
 529 associate, deprecate and remove is found in [ebRIM].
 530

531

Table 2: LifeCycle Manager Summary

Method Summary of LifeCycleManager	
RegistryResponse	approveObjects (ApproveObjectsRequest req) Approves one or more previously submitted objects.
RegistryResponse	deprecateObjects (DeprecateObjectsRequest req) Deprecates one or more previously submitted objects.
RegistryResponse	removeObjects (RemoveObjectsRequest req) Removes one or more previously submitted objects from the Registry.
RegistryResponse	submitObjects (SubmitObjectsRequest req) Submits one or more objects and possibly related metadata such as Associations and Classifications.
RegistryResponse	updateObjects (UpdateObjectsRequest req) Updates one or more previously submitted objects.
RegistryResponse	addSlots (AddSlotsRequest req) Add slots to one or more registry entries.
RegistryResponse	removeSlots (RemoveSlotsRequest req) Remove specified slots from one or more registry entries.

532 6.5 QueryManager Interface

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

536

537

Table 3: Query Manager

Method Summary of QueryManager	
RegistryResponse	submitAdhocQuery (AdhocQueryRequest req) Submit an ad hoc query request.

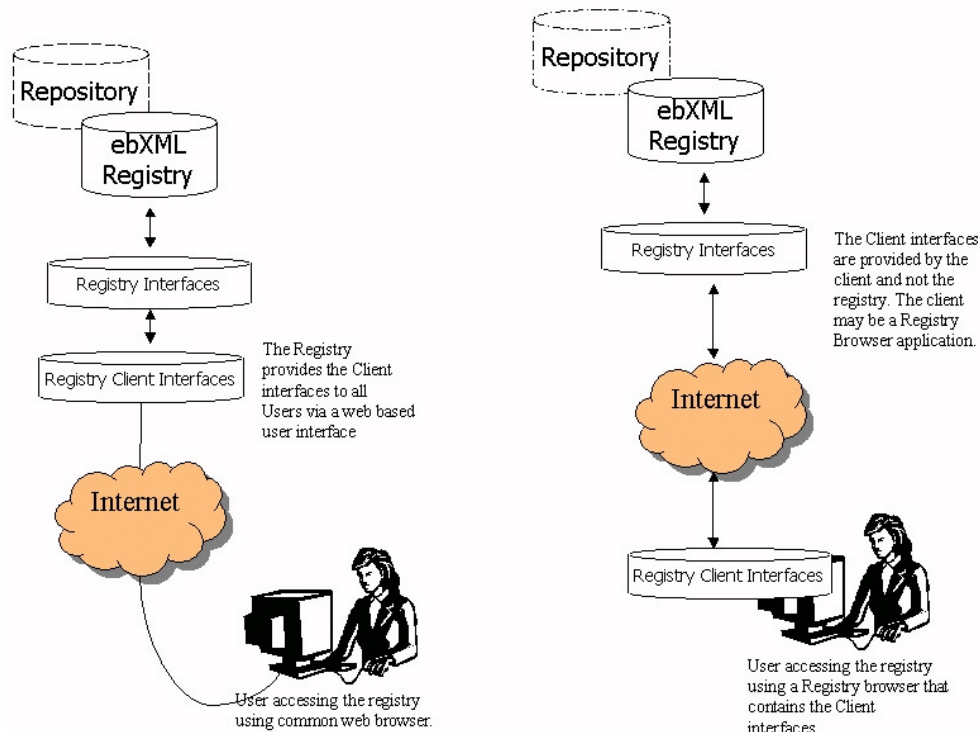
538 6.6 Registry Clients

539 6.6.1 Registry Client Description

540 The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts
 541 the two possible topologies supported by the registry architecture with respect to the Registry
 542 and Registry Clients. The picture on the left side shows the scenario where the Registry provides
 543 a web based "thin client" application for accessing the Registry that is available to the user using

544 a common web browser. In this scenario the Registry Client interfaces reside across the Internet
 545 and are local to the Registry from the user's view. The picture on the right side shows the
 546 scenario where the user is using a "fat client" Registry Browser application to access the registry.
 547 In this scenario the Registry Client interfaces reside within the Registry Browser tool and are
 548 local to the Registry from the user's view. The Registry Client interfaces communicate with the
 549 Registry over the Internet in this scenario.

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



555
 556

Figure 5: Registry Architecture Supports Flexible Topologies

557 6.6.2 Registry Communication Bootstrapping

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

- 564 • In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available
 565 in a WSDL description for the registry. This WSDL conforms to the template WSDL
 566 description in Appendix A.2. This WSDL description may be discovered in a public registry
 567 such as UDDI.

- 568 • In case of ebMS binding, the information exchange between the client and the registry may
 569 be accomplished in a registry specific manner, which may involve establishing a CPA
 570 between the client and the registry. Once the information exchange has occurred the Registry
 571 and the client will have addressing information (e.g. URLs) for the other party.

572 **6.6.2.1 Communication Bootstrapping for SOAP Binding**

573 Each ebXML Registry must provide a WSDL description for its RegistryService as defined by
 574 Appendix A.2. A client uses the WSDL description to determine the address information of the
 575 RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the
 576 RegistryService may be accessed via a URL specified in the WSDL for the registry.

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

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

582 **6.6.2.2 Communication Bootstrapping for ebXML Message Service**

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

593 **6.6.3 RegistryClient Interface**

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

599 The registry sends all asynchronous responses to operations to the onResponse method.
 600

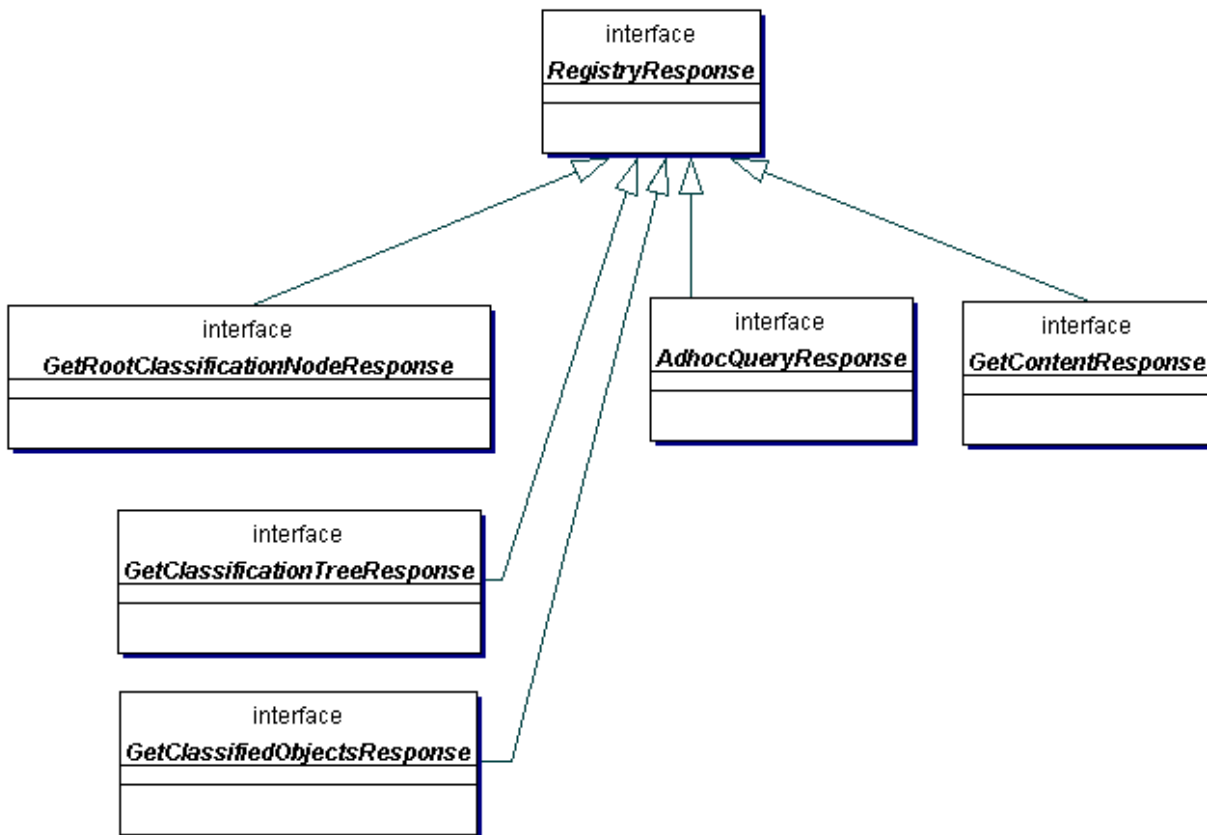
601

Table 4: RegistryClient Summary

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

602 **6.6.4 Registry Response Class Hierarchy**

603 Since many of the responses from the registry have common attributes, they are arranged in a
 604 class hierarchy as shown in . This hierarchy is reflected in the registry schema.



605 **Figure 6: Registry Response Class Hierarchy**

606 **6.7 Interoperability Requirements**

607 **6.7.1 Client Interoperability**

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

612 **6.7.2 Inter-Registry Cooperation**

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

616 Examples include:

- 617 • An ebXML Registry that serves as a registry of ebXML Registries.
- 618 • A non-ebXML Registry that serves as a registry of ebXML Registries.
- 619 • Cooperative ebXML Registries, where multiple ebXML registries register with each other in
 620 order to form a federation.

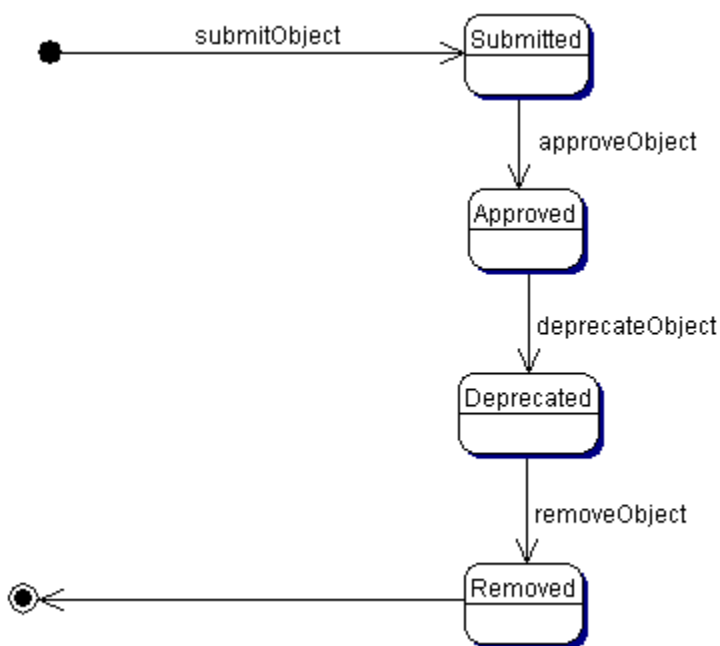
621 7 Life Cycle Management Service

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

628 The minimum-security *policy* for an ebXML registry is to accept content from any client if a
 629 certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the
 630 content. Submitting Organizations do not have to register prior to submitting content.

631 7.1 Life Cycle of a Repository Item

632 The main purpose of the ObjectManagement service is to manage the life cycle of repository
 633 items. Figure 7 shows the typical life cycle of a repository item. Note that the current version of
 634 this specification does not support Object versioning. Object versioning will be added in a future
 635 version of this specification



636
637

Figure 7: Life Cycle of a Repository Item

638 7.2 RegistryObject Attributes

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

644 7.3 The Submit Objects Protocol

645 This section describes the protocol of the Registry Service that allows a RegistryClient to submit

646 one or more repository items to the repository using the *ObjectManager* on behalf of a
 647 Submitting Organization. It is expressed in UML notation as described in Appendix C.



648
 649

Figure 8: Submit Objects Sequence Diagram

650 For details on the schema for the *Business documents* shown in this process refer to [Appendix B](#).

651 The SubmitObjectRequest message includes a RegistryEntryList element.

652 The RegistryEntryList element specifies one or more ExtrinsicObjects or other RegistryEntries
 653 such as Classifications, Associations, ExternalLinks, or Packages.

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

658 7.3.1 Universally Unique ID Generation

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

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

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

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

672 7.3.2 ID Attribute And Object References

673 The id attribute of an object may be used by other objects to reference the first object. Such
 674 references are common both within the SubmitObjectsRequest as well as within the registry.

675 Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the
 676 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the
 677 SubmitObjectsRequest that needs to be referred to within the request document may be assigned
 678 an id by the submitter so that it can be referenced within the request. The submitter may give the
 679 object a proper uuid URN, in which case the id is permanently assigned to the object within the
 680 registry. Alternatively, the submitter may assign an arbitrary id (not a proper uuid URN) as long
 681 as the id is unique within the request document. In this case the id serves as a linkage mechanism
 682 within the request document but must be ignored by the registry and replaced with a registry
 683 generated id upon submission.

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

688 **7.3.3 Audit Trail**

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

691 **7.3.4 Submitting Organization**

692 The RS must create an Association of type SubmittedBy between the submitting organization
 693 and each RegistryObject created via a SubmitObjects request. (Submitting organization is
 694 determined from the organization attribute of the User who submits a SubmitObjects request.)

695 **7.3.5 Error Handling**

696 A SubmitObjects request is atomic and either succeeds or fails in total. In the event of success,
 697 the registry sends a RegistryResponse with a status of “success” back to the client. In the event
 698 of failure, the registry sends a RegistryResponse with a status of “failure” back to the client.
 699 Failure occurs when one or more Error conditions are raised in the processing of the submitted
 700 objects. Warning messages do not result in failure of the request. The following business rules
 701 apply:

702 **Table 5 Submit Objects Error Handling**

Business Rule	Applies To	Error/Warning
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

7.3.6 Sample SubmitObjectsRequest

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

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

```

711 <?xml version = "1.0" encoding = "UTF-8"?>
712 <SubmitObjectsRequest
713   xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
714   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
715   xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
716 spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
717 file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
718   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
719   xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
720 >
721
722 <RegistryObjectList>
723
724   <!--
725   The following 3 objects package specified ExtrinsicObject in specified
726   RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
727   being submitted
728   -->
729
730   <rim:RegistryPackage id = "acmePackage1" >
731     <Name xmlns = "">
732       <LocalizedString value = "RegistryPackage #1"/>
733     </Name>
734     <Description xmlns = "">
735       <LocalizedString value = "ACME's package #1"/>
736     </Description>
737   </rim:RegistryPackage>
738
739   <rim:ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1" >
740     <Name xmlns = "">
741       <LocalizedString value = "Widget Profile" />
742     </Name>
743     <Description xmlns = "">
744       <LocalizedString value = "ACME's profile for selling widgets" />
745     </Description>
746   </rim:ExtrinsicObject>
747
748   <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
749   = "acmePackage1" targetObject = "acmeCPP1" />
750
751   <!--
752   The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
753   Where the RegistryPackage is being submitted and the ExtrinsicObject is
754   already in registry
755   -->
756
757   <rim:RegistryPackage id = "acmePackage2" >
758     <Name xmlns = "">
759       <LocalizedString value = "RegistryPackage #2"/>
760     </Name>
761     <Description xmlns = "">
762       <LocalizedString value = "ACME's package #2"/>
763     </Description>
764   </rim:RegistryPackage>
765
766   <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
767
768   <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
769   sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
770
771   <!--
772   The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
773   where the RegistryPackage and the ExtrinsicObject are already in registry
  
```

```
774     -->
775
776     <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
777     <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
778
779     <!-- id is unspecified implying that registry must create a uuid for this object -->
780
781     <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
782 123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
783
784     <!--
785     The following 3 objects externally link specified ExtrinsicObject using
786     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
787     are being submitted
788     -->
789
790     <rim:ExternalLink id = "acmeLink1" >
791       <Name xmlns = "">
792         <LocalizedString value = "Link #1"/>
793       </Name>
794       <Description xmlns = "">
795         <LocalizedString value = "ACME's Link #1"/>
796       </Description>
797     </rim:ExternalLink>
798
799     <rim:ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" >
800       <Name xmlns = "">
801         <LocalizedString value = "Sprockets Profile" />
802       </Name>
803       <Description xmlns = "">
804         <LocalizedString value = "ACME's profile for selling sprockets"/>
805       </Description>
806     </rim:ExtrinsicObject>
807
808     <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
809 sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
810
811     <!--
812     The following 2 objects externally link specified ExtrinsicObject using specified
813     ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
814     is already in registry. Note that the targetObject points to an ObjectRef in a
815     previous line
816     -->
817
818     <rim:ExternalLink id = "acmeLink2">
819       <Name xmlns = "">
820         <LocalizedString value = "Link #2"/>
821       </Name>
822       <Description xmlns = "">
823         <LocalizedString value = "ACME's Link #2"/>
824       </Description>
825     </rim:ExternalLink>
826
827     <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
828 "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
829 123456789012"/>
830
831     <!--
832     The following 3 objects externally identify specified ExtrinsicObject using specified
833     ExternalIdentifier, where the ExternalIdentifier is being submitted and the
834     ExtrinsicObject is already in registry. Note that the targetObject points to an
835     ObjectRef in a previous line
836     -->
837
838     <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
839       <Name xmlns = "">
840         <LocalizedString value = "DUNS"/>
841       </Name>
842
843       <Description xmlns = "">
844         <LocalizedString value = "This is the DUNS scheme"/>
845       </Description>
846     </rim:ClassificationScheme>
847
```

```

848 <rim:ExternalIdentifier id = "acmeDUNSId" identificationScheme="DUNS-id" value =
849 "13456789012">
850 <Name xmlns = "">
851 <LocalizedString value = "DUNS" />
852 </Name>
853 <Description xmlns = "">
854 <LocalizedString value = "DUNS ID for ACME"/>
855 </Description>
856 </rim:ExternalIdentifier>
857
858 <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType =
859 "ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
860 123456789012"/>
861
862 <!--
863 The following show submission of a brand new classification scheme in its entirety
864 -->
865 <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
866 <Name xmlns = "">
867 <LocalizedString value = "Geography"/>
868 </Name>
869
870 <Description xmlns = "">
871 <LocalizedString value = "This is a sample Geography scheme"/>
872 </Description>
873
874 <ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code = "NorthAmerica" >
875 <ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
876 "UnitedStates" />
877 <ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
878 </ClassificationNode>
879
880 <ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
881 <ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
882 <ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
883 </ClassificationNode>
884 </ClassificationNode>
885 </rim:ClassificationScheme>
886
887
888 <!--
889 The following show submission of a Automotive sub-tree of ClassificationNodes that
890 gets added to an existing classification scheme named 'Industry'
891 that is already in the registry
892 -->
893
894 <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
895 <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
896 123456789012">
897 <Name xmlns = "">
898 <LocalizedString value = "Automotive" />
899 </Name>
900 <Description xmlns = "">
901 <LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
902 </Description>
903 </rim:ClassificationNode>
904
905 <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
906 <Name xmlns = "">
907 <LocalizedString value = "Parts Supplier" />
908 </Name>
909 <Description xmlns = "">
910 <LocalizedString value = "The Parts Supplier node under the Automotive node" />
911 </Description>
912 </rim:ClassificationNode>
913
914 <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
915 <Name xmlns = "">
916 <LocalizedString value = "Engine Supplier" />
917 </Name>
918 <Description xmlns = "">
919 <LocalizedString value = "The Engine Supplier node under the Automotive node" />
920 </Description>
921 </rim:ClassificationNode>
922

```

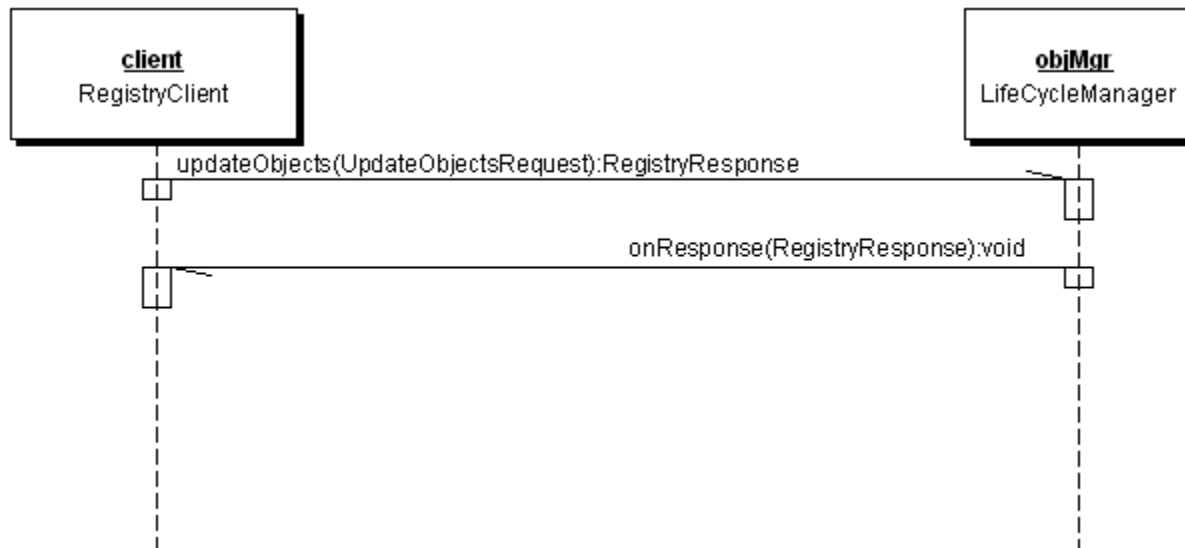
```

923 <!--
924 The following show submission of 2 Classifications of an object that is already in
925 the registry using 2 ClassificationNodes. One ClassificationNode
926 is being submitted in this request (Japan) while the other is already in the registry.
927 -->
928
929 <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
930 1234-123456789012" classificationNode = "Japan-id">
931 <Description xmlns = "">
932 <LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
933 </Description>
934 </rim:Classification>
935
936 <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
937 "urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
938 123456789012">
939 <Description xmlns = "">
940 <LocalizedString value = "Classifies object using a node in the registry" />
941 </Description>
942 </rim:Classification>
943
944 <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
945 </RegistryObjectList>
946 </SubmitObjectsRequest>
947

```

948 7.4 The Update Objects Protocol

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



952
 953 **Figure 9: Update Objects Sequence Diagram**

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

960 7.4.1 Audit Trail

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

963 7.4.2 Submitting Organization

964 The RS must maintain an Association of type SubmittedBy between the submitting organization
 965 and each RegistryObject updated via an UpdateObjects request. If an UpdateObjects request is
 966 accepted from a different submitting organization, then the RS must delete the original
 967 association object and create a new one. Of course, the AccessControlPolicy may prohibit this
 968 sort of update in the first place. (Submitting organization is determined from the organization
 969 attribute of the User who submits an UpdateObjects request.)

970 7.4.3 Error Handling

971 An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success,
 972 the registry sends a RegistryResponse with a status of “success” back to the client. In the event
 973 of failure, the registry sends a RegistryResponse with a status of “failure” back to the client.
 974 Failure occurs when one or more Error conditions are raised in the processing of the updated
 975 objects. Warning messages do not result in failure of the request. The following business rules
 976 apply:

977 **Table 6: Update Objects Error Handling**

Business Rule	Applies To	Error/Warning
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
ContentURI cannot be changed via the UpdateObjects protocol, and is ignored if supplied.	ExtrinsicObject	Warning
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

978 7.5 The Add Slots Protocol

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



982
983

Figure 10: Add Slots Sequence Diagram

984 In the event of success, the registry sends a RegistryResponse with a status of “success” back to
985 the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure”
986 back to the client.

987 7.6 The Remove Slots Protocol

988 This section describes the protocol of the Registry Service that allows a client to remove slots to
989 a previously submitted registry entry using the ObjectManager.



990
991

Figure 11: Remove Slots Sequence Diagram

992 7.7 The Approve Objects Protocol

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



Figure 12: Approve Objects Sequence Diagram

997
998

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

1000 **7.7.1 Audit Trail**

1001 The RS must create AuditableEvents object with eventType Approved for each RegistryObject
1002 approved via an Approve Objects request.

1003 **7.7.2 Submitting Organization**

1004 The RS must maintain an Association of type SubmittedBy between the submitting organization
1005 and each RegistryObject updated via an ApproveObjects request. If an ApproveObjects request
1006 is accepted from a different submitting organization, then the RS must delete the original
1007 association object and create a new one. Of course, the AccessControlPolicy may prohibit this
1008 sort of ApproveObjects request in the first place. (Submitting organization is determined from
1009 the organization attribute of the User who submits an ApproveObjects request.)

1010 **7.7.3 Error Handling**

1011 An ApproveObjects request is atomic and either succeeds or fails in total. In the event of success,
1012 the registry sends a RegistryResponse with a status of “success” back to the client. In the event
1013 of failure, the registry sends a RegistryResponse with a status of “failure” back to the client.
1014 Failure occurs when one or more Error conditions are raised in the processing of the object
1015 reference list. Warning messages do not result in failure of the request. The following business
1016 rules apply:

Table 7: Approve Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "approved".	All Classes other than RegistryEntry classes	Error

1017

Object status is already "Approved".	RegistryEntry Classes	Warning
--------------------------------------	--------------------------	---------

1018 7.8 The Deprecate Objects Protocol

1019 This section describes the protocol of the Registry Service that allows a client to deprecate one or
 1020 more previously submitted repository items using the ObjectManager. Once an object is
 1021 deprecated, no new references (e.g. *new* Associations, Classifications and ExternalLinks) to that
 1022 object can be submitted. However, existing references to a deprecated object continue to function
 1023 normally.



1024
1025

Figure 13: Deprecate Objects Sequence Diagram

1026 For details on the schema for the business documents shown in this process refer to Appendix B.

1027 7.8.1 Audit Trail

1028 The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject
 1029 deprecated via a Deprecate Objects request.

1030 7.8.2 Submitting Organization

1031 The RS must maintain an Association of type SubmittedBy between the submitting organization
 1032 and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request
 1033 is accepted from a different submitting organization, then the RS must delete the original
 1034 association object and create a new one. Of course, the AccessControlPolicy may prohibit this
 1035 sort of Deprecate Objects request in the first place. (Submitting organization is determined from
 1036 the organization attribute of the User who submits a Deprecate Objects request.)

1037 7.8.3 Error Handling

1038 A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of
 1039 success, the registry sends a RegistryResponse with a status of "success" back to the client. In
 1040 the event of failure, the registry sends a RegistryResponse with a status of "failure" back to the
 1041 client. Failure occurs when one or more Error conditions are raised in the processing of the
 1042 object reference list. Warning messages do not result in failure of the request. The following
 1043 business rules apply:

1044

Table 8: Deprecate Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "deprecated".	All Classes other than RegistryEntry classes	Error
Object status is already "Deprecated".	RegistryEntry Classes	Warning

1045 **7.9 The Remove Objects Protocol**

1046 This section describes the protocol of the Registry Service that allows a client to remove one or
1047 more RegistryEntry instances and/or repository items using the ObjectManager.

1048 The RemoveObjectsRequest message is sent by a client to remove RegistryEntry instances
1049 and/or repository items. The RemoveObjectsRequest element includes an XML attribute called
1050 *deletionScope* which is an enumeration that can have the values as defined by the following
1051 sections.

1052 **7.9.1 Deletion Scope DeleteRepositoryItemOnly**

1053 This deletionScope specifies that the request should delete the repository items for the specified
1054 registry entries but not delete the specified registry entries. This is useful in keeping references to
1055 the registry entries valid.

1056 **7.9.2 Deletion Scope DeleteAll**

1057 This deletionScope specifies that the request should delete both the RegistryEntry and the
1058 repository item for the specified registry entries. Only if all references (e.g. Associations,
1059 Classifications, ExternalLinks) to a RegistryEntry have been removed, can that RegistryEntry
1060 then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to
1061 remove a RegistryEntry while it still has references raises an error condition:
1062 InvalidRequestError.

1063 The remove object protocol is expressed in UML notation as described in Appendix C.



1064
1065

Figure 14: Remove Objects Sequence Diagram

1066 For details on the schema for the business documents shown in this process refer to Appendix B.

1067 **7.9.3 Error Handling**

1068 A Remove Objects request is atomic and either succeeds or fails in total. In the event of success,
 1069 the registry sends a RegistryResponse with a status of “success” back to the client. In the event
 1070 of failure, the registry sends a RegistryResponse with a status of “failure” back to the client.
 1071 Failure occurs when one or more Error conditions are raised in the processing of the object
 1072 reference list. Warning messages do not result in failure of the request. The following business
 1073 rules apply:
 1074

Table 9: Remove Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "removed".	All Classes other than RegistryEntry classes	Error

1075

1076 **8 Query Management Service**

1077 This section describes the capabilities of the Registry Service that allow a client
1078 (QueryManagerClient) to search for or query different kind of registry objects in the ebXML
1079 Registry using the QueryManager interface of the Registry. The Registry supports the following
1080 query capabilities:

- 1081 1. Filter Query
- 1082 2. SQL Query

1083 The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry
1084 implementation. The SQL Query mechanism is an optional feature and MAY be provided by a
1085 registry implementation. However, if a vendor provides an SQL query capability to an ebXML
1086 Registry it SHALL conform to this document. As such this capability is a normative yet optional
1087 capability.

1088 In a future version of this specification, the W3C XQuery syntax may be considered as another
1089 query syntax.

1090 The Registry will hold a self-describing **capability profile** that identifies all supported
1091 AdhocQuery options. This profile is described in Section **Error! Reference source not found.**

1092 **8.1 Ad Hoc Query Request/Response**

1093 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
1094 AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry
1095 query mechanisms.

1096 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously
1097 back to the client. The AdhocQueryResponse returns a collection of objects whose element type
1098 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects
1099 representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate
1100 classes in [ebRIM] such as RegistryObject and RegistryEntry.

1101 Any errors in the query request messages are indicated in the corresponding query response
1102 message.

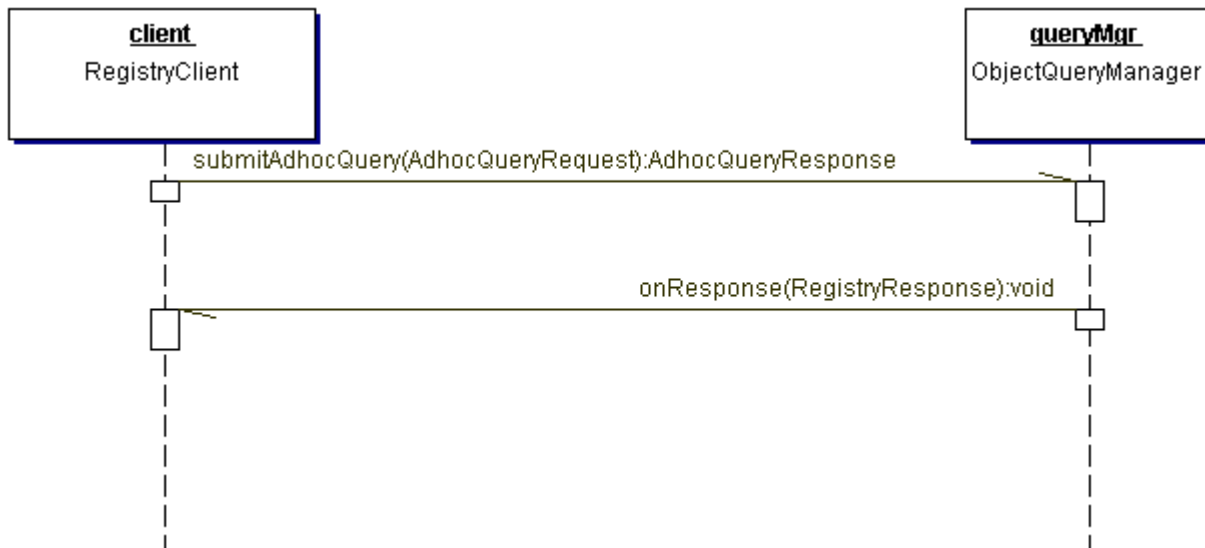


Figure 15: Submit Ad Hoc Query Sequence Diagram

1103

1104 For details on the schema for the business documents shown in this process refer to Appendix

1105 B.2.

1106 Definition

1107

```
1108 <element name="AdhocQueryRequest">
```

```
1109   <complexType>
```

```
1110     <sequence>
```

```
1111       <element ref="tns:ResponseOption" minOccurs="1" maxOccurs="1" />
```

```
1112       <choice minOccurs="1" maxOccurs="1">
```

```
1113         <element ref="FilterQuery" />
```

```
1114         <element ref="SQLQuery" />
```

```
1115       </choice>
```

```
1116     </sequence>
```

```
1117   </complexType>
```

```
1118 </element>
```

1119

```
1120 <element name="AdhocQueryResponse">
```

```
1121   <complexType>
```

```
1122     <choice>
```

```
1123       <element ref="SQLQueryResult" />
```

```
1124       <element ref="FilterQueryResult" />
```

```
1125     </choice>
```

```
1126   </complexType>
```

```
1127 </element>
```

1128

1129 8.1.1 Query Response Options

1130 Purpose

1131 A QueryManagerClient may specify what an ad hoc query must return within an
1132 AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest.

1133 ResponseOption element has an attribute "returnType" and its values are:

- 1134 • ObjectRef - This option specifies that the AdhocQueryResponse must contain a collection of
1135 ObjectRef XML elements as defined in [RIM schema]. Purpose of this option is to return just
1136 the identifiers of the registry objects.

- 1137 • RegistryObject - This option specifies that the AdhocQueryResponse must contain a
 1138 collection of RegistryObject XML elements as defined in [RIM schema]. In this case all
 1139 attributes of the registry objects are returned (objectType, name, description, ...) in addition
 1140 to id attribute.
- 1141 • RegistryEntry - This option specifies that the AdhocQueryResponse must contain a
 1142 collection of RegistryEntry XML elements as defined in [RIM schema], which correspond to
 1143 RegistryEntry attributes.
- 1144 • LeafClass - This option specifies that the AdhocQueryResponse must contain a collection of
 1145 XML elements that correspond to leaf classes as defined in [RIM schema].
- 1146 • LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse must
 1147 contain a collection of ExtrinsicObject XML elements as defined in [RIM schema]
 1148 accompanied with their repository items. Linking of ExtrinsicObject and its repository item
 1149 is done via contentURI as explained in [XXX – Content Retrieval section].
- 1150 ResponseOption element also has an attribute “returnComposedObjects”. It specifies whether or
 1151 not the whole hierarchy of composed objects are returned with the registry objects.
- 1152 If “returnType” is higher then the RegistryObject option, then the highest option that satisfies the
 1153 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return
 1154 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass
 1155 option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then
 1156 RegistryObject metadata will be returned.

1157 Definition

```

1158 <complexType name="ResponseOptionType">
1159   <attribute name="returnType" default="RegistryObject">
1160     <simpleType>
1161       <restriction base="NMTOKEN">
1162         <enumeration value="ObjectRef" />
1163         <enumeration value="RegistryObject" />
1164         <enumeration value="RegistryEntry" />
1165         <enumeration value="LeafClass" />
1166         <enumeration value="LeafClassWithRepositoryItem" />
1167       </restriction>
1168     </simpleType>
1169   </attribute>
1170   <attribute name="returnComposedObjects" type="boolean" default="false" />
1171 </complexType>
1172 <element name="ResponseOption" type="tns:ResponseOptionType" />
1173
1174

```

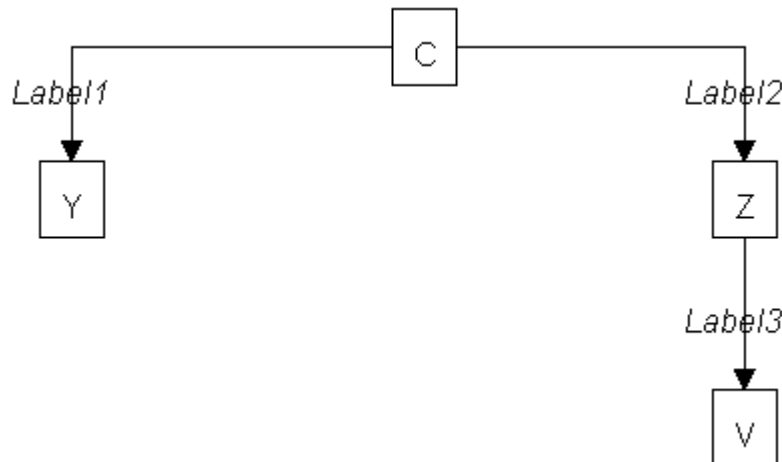
1175 8.2 Filter Query Support

1176 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
 1177 conforming Registry implementation. Each query alternative is directed against a single class
 1178 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter queries
 1179 depending on which classes are queried on.

- 1180 • Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic
 1181 queries that might return different subclasses of the class that is queried on. The result of
 1182 such a query is a set of XML elements that correspond to instances of any class that satisfies
 1183 the responseOption defined previously in [Section 8.1.1](#). An example might be that
 1184 RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances
 1185 that satisfy the query. This implies that response might return XML elements that correspond
 1186 to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- 1187 • Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact
 1188 traversals of these classes. Responses to these queries are accordingly constrained.

1189 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an
 1190 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified
 1191 herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified
 1192 in [Section 8.1](#).

1193 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of
 1194 classes derived from a single class and its associations with other classes as defined by ebRIM.
 1195 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For
 1196 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a
 1197 class that is associated with Z. The ebRIM Binding for C might be as in Figure 16



1198
 1199 **Figure 16: Example ebRIM Binding**

1200 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and
 1201 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to
 1202 which ebRIM association is intended. The name of the query is determined by the root class, i.e.
 1203 this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances
 1204 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are
 1205 limited to instances that are linked to their parent node by the identified association.

1206 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a
 1207 restricted *predicate clause* over the attributes of a single class. Class methods that are defined in
 1208 ebRIM and that return simple types constitute “visible attributes” that are valid choices for
 1209 predicate clauses. Names of those attributes will be same as name of the corresponding method
 1210 just without the prefix ‘get’. For example, in case of “getLevelNumber” method the
 1211 corresponding visible attribute is “levelNumber”. The supported class filters are specified in
 1212 [Section 8.2.11](#) and the supported predicate clauses are defined in [Section 8.2.12](#). A FilterQuery
 1213 will be composed of elements that traverse the tree to determine which branches satisfy the
 1214 designated class filters, and the query result will be the set of instances that support such a
 1215 branch.

1216 In the above example, the CQuery element will have three subelements, one a CFilter on the C
1217 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on
1218 the Y class to eliminate branches from C to Y where the target of the association does not satisfy
1219 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third
1220 element is called a *branch* element because it allows class filters on each class along the path
1221 from C to V. In general, a branch element will have subelements that are themselves class filters,
1222 other branch elements, or a full-blown query on the class in the path.

1223 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one
1224 branch, filter or query element on Y is allowed. However, if the association is one-to-many, then
1225 multiple branch, filter or query elements are allowed. This allows one to specify that an instance
1226 of C must have associations with multiple instances of Y before the instance of C is said to
1227 satisfy the branch element.

1228 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be
1229 stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then
1230 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax
1231 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their
1232 respective superclass queries. Structures of XML elements that match the ebRIM classes are
1233 explained in [RIM Schema]. Names of Filters, Queries and Branches correspond to names in
1234 ebRIM whenever possible.

1235 **The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.10 below identify the virtual**
1236 **hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative**
1237 **specify the effect of that binding on query semantics.**

1238 8.2.1 FilterQuery

1239 Purpose

1240 To identify a set of queries that traverse specific registry class. Each alternative assumes a
1241 specific binding to ebRIM. The status is a success indication or a collection of warnings and/or
1242 exceptions.

1243 Definition

```
1244 <element name="FilterQuery">
1245   <complexType>
1246     <choice minOccurs="1" maxOccurs="1">
1247       <element ref="tns:RegistryObjectQuery" />
1248       <element ref="tns:RegistryEntryQuery" />
1249       <element ref="tns:AuditableEventQuery" />
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   </complexType>
1258 </element>
1259
1260 <element name="FilterQueryResult">
1261   <complexType>
1262     <choice minOccurs="1" maxOccurs="1">
1263       <element ref="tns:RegistryObjectQueryResult" />
1264       <element ref="tns:RegistryEntryQueryResult" />
1265     </choice>
1266   </complexType>
1267 </element>
```

```
1266     <element ref="tns:AuditableEventQueryResult" />
1267     <element ref="tns:ClassificationNodeQueryResult" />
1268     <element ref="tns:ClassificationSchemeQueryResult" />
1269     <element ref="tns:RegistryPackageQueryResult" />
1270     <element ref="tns:ExtrinsicObjectQueryResult" />
1271     <element ref="tns:OrganizationQueryResult" />
1272     <element ref="tns:ServiceQueryResult" />
1273   </choice>
1274 </complexType>
1275 </element>
1276
```

1277 **Semantic Rules**

- 1278 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 1279 2. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
1280 Each XML attribute carries a value derived from the value of an attribute specified in the
1281 Registry Information Model [RIM Schema].
- 1282 3. For each FilterQuery subelement there is only one corresponding FilterQueryResult
1283 subelement that must be returned as a response. Class name of the FilterQueryResult
1284 subelement has to match the class name of the FilterQuery subelement.
- 1285 4. If an error condition is raised during any part of the execution of a FilterQuery, then the
1286 status attribute of the XML RegistryResult is set to “failure” and no query result element is
1287 returned; instead, a RegistryErrorList element must be returned with its highestSeverity
1288 element set to “error”. At least one of the RegistryError elements in the RegistryErrorList
1289 will have its severity attribute set to “error”.
- 1290 5. If no error conditions are raised during execution of a FilterQuery, then the status attribute of
1291 the XML RegistryResult is set to “success” and an appropriate query result element must be
1292 included. If a RegistryErrorList is also returned, then the highestSeverity attribute of the
1293 RegistryErrorList is set to “warning” and the serverity attribute of each RegistryError is set to
1294 “warning”.

1295 **8.2.2 RegistryObjectQuery**

1296 **Purpose**

1297 To identify a set of registry object instances as the result of a query over selected registry
1298 metadata.

1299 **ebRIM Binding**

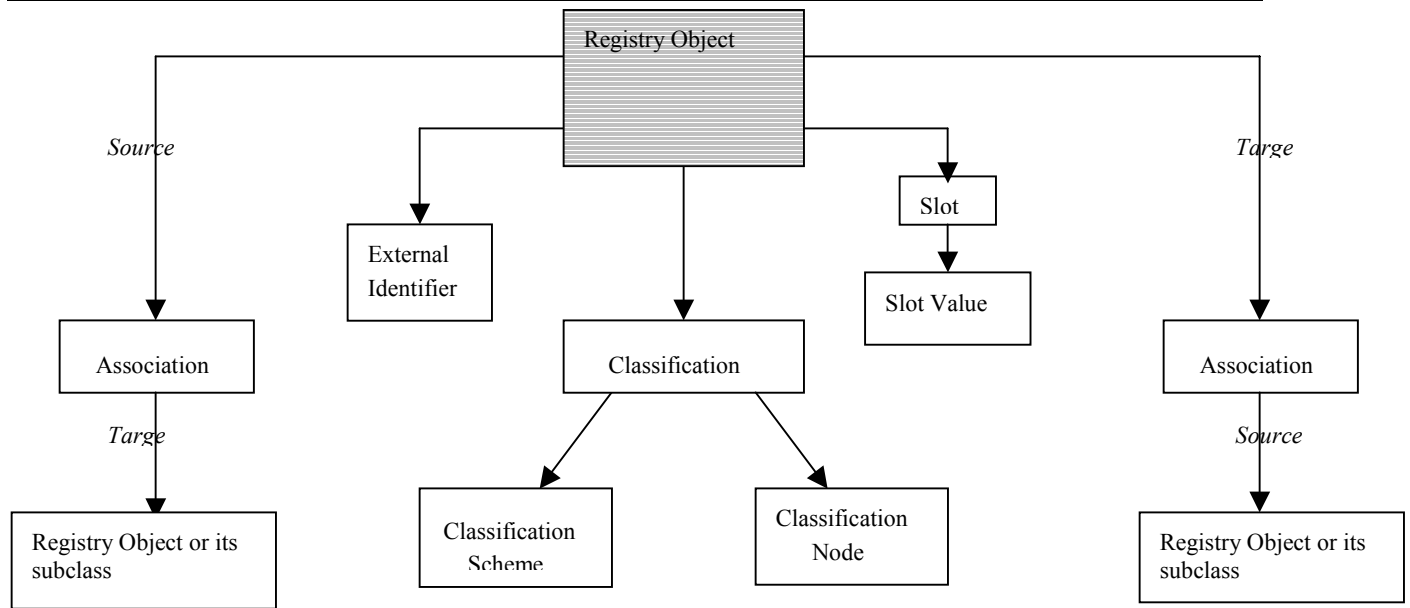


Figure 17: ebRim Binding for RegistryObjectQuery

1300

1301 **Definition**

```

1302 <complexType name="RegistryObjectQueryType">
1303   <sequence>
1304     <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
1305     <element name="NameBranch" type="InternationalStringBranchType" minOccurs="0" maxOccurs="1" />
1306     <element name="DescriptionBranch" type="InternationalStringBranchType" minOccurs="0" maxOccurs="1" />
1307     <element ref="SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1308     <element ref="TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1309     <element ref="tns:ClassificationBranch" minOccurs="0" maxOccurs="unbounded" />
1310     <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
1311     <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
1312   </sequence>
1313 </complexType>
1314 <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
1315
1316 <complexType name="LeafRegistryObjectListType">
1317   <choice minOccurs="0" maxOccurs="unbounded">
1318     <element ref="tns:ObjectRef" />
1319     <element ref="tns:Association" />
1320     <element ref="tns:AuditableEvent" />
1321     <element ref="tns:Classification" />
1322     <element ref="tns:ClassificationNode" />
1323     <element ref="tns:ClassificationScheme" />
1324     <element ref="tns:ExternalIdentifier" />
1325     <element ref="tns:ExternalLink" />
1326     <element ref="tns:ExtrinsicObject" />
1327     <element ref="tns:Organization" />
1328     <element ref="tns:RegistryPackage" />
1329     <element ref="tns:Service" />
1330     <element ref="tns:ServiceBinding" />
1331     <element ref="tns:SpecificationLink" />
1332     <element ref="tns:User" />
1333   </choice>
1334 </complexType>
1335
1336 <complexType name="RegistryObjectListType">
1337   <complexContent>
  
```

```
1338 <extension base="tns:LeafRegistryObjectListType">
1339 <choice minOccurs="0" maxOccurs="unbounded">
1340 <element ref="tns:RegistryEntry" />
1341 <element ref="tns:RegistryObject" />
1342 </choice>
1343 </extension>
1344 </complexContent>
1345 </complexType>
1346 <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1347
1348 <complexType name="InternationalStringBranchType">
1349 <sequence>
1350 <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1351 </sequence>
1352 </complexType>
1353
1354 <complexType name="AssociationBranchType">
1355 <sequence>
1356 <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1357 <choice minOccurs="0" maxOccurs="1">
1358 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1359 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1360 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1361 <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1362 <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1363 <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1364 <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1365 <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1366 <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1367 <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1368 <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1369 <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
1370 <element ref="tns:ClassificationBranch" minOccurs="0" maxOccurs="1" />
1371 <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1372 <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1373 <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="1" />
1374 <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="1" />
1375 </choice>
1376 </sequence>
1377 </complexType>
1378 <element name="SourceAssociationBranch" type="AssociationBranchType" />
1379 <element name="TargetAssociationBranch" type="AssociationBranchType" />
1380
1381 <element name="ClassificationBranch">
1382 <complexType>
1383 <sequence>
1384 <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
1385 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1386 <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1387 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1388 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1389 </sequence>
1390 </complexType>
1391 </element>
1392
1393 <element name="SlotBranch">
1394 <complexType>
1395 <sequence>
1396 <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />

```

```

1397     <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
1398   </sequence>
1399 </complexType>
1400 </element>
1401
1402 <element name="UserBranch">
1403   <complexType>
1404     <sequence>
1405       <element ref="tns:UserFilter" minOccurs="0" maxOccurs="1" />
1406       <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
1407       <element ref="tns:TelephoneFilter" minOccurs="0" maxOccurs="unbounded" />
1408       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1409     </sequence>
1410   </complexType>
1411 </element>
1412
1413 <element name="ServiceBindingBranch">
1414   <complexType>
1415     <sequence>
1416       <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
1417       <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
1418     </sequence>
1419   </complexType>
1420 </element>
1421
1422 <element name="SpecificationLinkBranch">
1423   <complexType>
1424     <sequence>
1425       <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
1426       <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" minOccurs="0"
1427         maxOccurs="1" />
1428       <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" minOccurs="0"
1429         maxOccurs="1" />
1430     </sequence>
1431   </complexType>
1432 </element>
1433

```

1434 **Semantic Rules**

- 1435 1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The
 1436 following steps will eliminate instances in RO that do not satisfy the conditions of the
 1437 specified filters.
- 1438 a) If RO is empty then continue below.
- 1439 b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a
 1440 registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from
 1441 RO. If RO is empty then continue below. If a NameBranch is not specified then go to the
 1442 next step; otherwise, let x be a remaining registry object in RO. If x does not have a name
 1443 then remove x from RO. If RO is empty then continue below; otherwise treat
 1444 NameBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by
 1445 some of the LocalizedStrings that constitute the name of the registry object then remove x
 1446 from RO. If RO is empty then continue below.

- 1447 c) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a
1448 remaining registry object in RO. If x does not have a name then remove x from RO. If
1449 RO is empty then continue below; otherwise treat DescriptionBranch as follows: If any
1450 LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings
1451 that constitute the description of the registry object then remove x from RO. If RO is
1452 empty then continue below.
- 1453 d) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,
1454 let x be a remaining registry object in RO. If x is not the source object of some
1455 Association instance, then remove x from RO. If RO is empty then continue below;
1456 otherwise, treat each SourceAssociationBranch element separately as follows:
1457 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be
1458 the set of all Association instances that have x as a source object; otherwise, let AF be the
1459 set of Association instances that satisfy the AssociationFilter and have x as the source
1460 object. If AF is empty, then remove x from RO.
1461
1462 If RO is empty then continue below.
1463
1464 If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT
1465 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1466 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1467 empty then continue below.
1468
1469 If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT
1470 be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the
1471 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1472 empty then continue below.
1473
1474 If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let
1475 ROT be the set of ClassificationScheme instances that satisfy the
1476 ClassificationSchemeQuery and are the target object of some element of AF. If ROT is
1477 empty, then remove x from RO. If RO is empty then continue below.
1478
1479 If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let
1480 ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1481 and are the target object of some element of AF. If ROT is empty, then remove x from
1482 RO. If RO is empty then continue below.
1483
1484 If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT
1485 be the set of Organization instances that satisfy the OrganizationQuery and are the target
1486 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1487 then continue below.
1488
1489 If an AuditableEventQuery is specified within the SourceAssociationBranch, then let
1490 ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1491 the target object of some element of AF. If ROT is empty, then remove x from RO. RO is
1492 empty then continue below.

1493
1494 If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let
1495 ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1496 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1497 RO is empty then continue below.
1498
1499 If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let
1500 ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1501 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1502 is empty then continue below.
1503
1504 If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the
1505 set of Service instances that satisfy the ServiceQuery and are the target object of some
1506 element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue
1507 below.
1508
1509 If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT
1510 be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target
1511 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1512 then continue below.
1513
1514 If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let
1515 ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1516 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1517 RO is empty then continue below.
1518
1519 If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set
1520 of User instances that are the target object of some element of AF. If ROT is empty, then
1521 remove x from RO. If RO is empty then continue below. Let u be the member of ROT. If
1522 a UserFilter element is specified within the UserBranch, and if u does not satisfy that
1523 filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is
1524 empty then continue below. If a PostalAddressFilter element is specified within the
1525 UserBranch, and if the postal address of u does not satisfy that filter, then remove u from
1526 ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. If
1527 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
1528 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
1529 remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then
1530 continue below. If an OrganizationQuery element is specified within the UserBranch,
1531 then let o be the Organization instance that is identified by the organization that u is
1532 affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then
1533 remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then
1534 continue below.
1535

1536 If a ClassificationBranch is specified within the SourceAssociationBranch then let ROT
1537 be the set of Classification instances that are the target object of some element of AF. If
1538 ROT is empty, then remove x from RO. If RO is empty then continue below. Let cb be
1539 the member of ROT. If ClassificationFilter element is specified within the
1540 ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROT. If
1541 ROT is empty, then remove x from RO. If RO is empty then continue below. if a
1542 ClassificationSchemeQuery element is specified within the ClassificationBranch then
1543 replace ROT by the set of remaining Classification instances in ROT whose defining
1544 classification scheme satisfies the ClassificationSchemeQuery. If ROT is empty, then
1545 remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery
1546 element is specified within the ClassificationBranch, then replace ROT by the set of
1547 remaining Classification instances in ROT for which a classification node exists and for
1548 which that classification node satisfies the ClassificationNodeQuery. If ROT is empty,
1549 then remove x from RO. If RO is empty then continue below..

1550

1551 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1552 ROT be the set of ServiceBinding instances that are the target object of some element of
1553 AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let
1554 sb be the member of ROT. If a ServiceBindingFilter element is specified within the
1555 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROT. If
1556 ROT is empty then remove x from RO. If RO is empty then continue below. If a
1557 SpecificationLinkBranch is specified within the ServiceBindingBranch then consider
1558 each SpecificationLinkBranch element separately as follows:

1559 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link
1560 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1561 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1562 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then
1563 remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1564 element is specified within the SpecificationLinkBranch then let sl be a remaining
1565 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
1566 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
1567 specification link for some registry object in RO, then remove sl from SL. If SL is empty
1568 then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then
1569 continue below. If a RegistryEntryQuery element is specified within the
1570 SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
1571 RegistryEntryQuery element as follows: Let RE be the result set of the
1572 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
1573 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROT.
1574 If ROT is empty then remove x from RO. If RO is empty then continue below.

1575

1576 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
1577 ROT be the set of SpecificationLink instances that are the target object of some element
1578 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.
1579 Let sl be the member of ROT. If a SpecificationLinkFilter element is specified within the
1580 SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROT.
1581 If ROT is empty then remove x from RO. If RO is empty then continue below. If a
1582 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1583 be a remaining specification link in ROT. Treat RegistryObjectQuery element as follows:
1584 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
1585 not a specification link for some registry object in RO, then remove sl from ROT. If ROT
1586 is empty then remove x from RO. If RO is empty then continue below. If a
1587 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
1588 be a remaining specification link in ROT. Treat RegistryEntryQuery element as follows:
1589 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
1590 a specification link for some registry entry in RE, then remove sl from ROT. If ROT is
1591 empty then remove x from RO. If RO is empty then continue below.
1592
1593 If a SourceAssociationBranch is specified within the SourceAssociationBranch, then let
1594 ROT be the set of RegistryObject instances that satisfy the SourceAssociationBranch and
1595 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1596 RO is empty then continue below.
1597
1598 If a TargetAssociationBranch is specified within the SourceAssociationBranch, then let
1599 ROT be the set of RegistryObject instances that satisfy the TargetAssociationBranch and
1600 are the source object of some element of AF. If ROT is empty, then remove x from RO.
1601 If RO is empty then continue below.
1602 e) If a TargetAssociationBranch element is not specified then go to the next step; otherwise,
1603 let x be a remaining registry object in RO. If x is not the target object of some
1604 Association instance, then remove x from RO. If RO is empty then continue below;
1605 otherwise, treat each TargetAssociationBranch element separately as follows:
1606
1607 If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be
1608 the set of all Association instances that have x as a target object; otherwise, let AF be the
1609 set of Association instances that satisfy the AssociationFilter and have x as the target
1610 object. If AF is empty, then remove x from RO. If RO is empty then continue below.
1611
1612 If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS
1613 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1614 source object of some element of AF. If ROS is empty, then remove x from RO. If RO is
1615 empty then continue below
1616
1617 If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS
1618 be the set of
1619 RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of
1620 some element of AF. If ROS is empty, then remove x from RO. If RO is empty then
1621 continue below.
1622

1623 If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let
1624 ROS be the set of ClassificationScheme instances that satisfy the
1625 ClassificationSchemeQuery and are the source object of some element of AF. If ROS is
1626 empty, then remove x from RO. If RO is empty then continue below.

1627

1628 If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let
1629 ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1630 and are the source object of some element of AF. If ROS is empty, then remove x from
1631 RO. If RO is empty then continue below.

1632

1633 If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS
1634 be the set of Organization instances that satisfy the OrganizationQuery and are the source
1635 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1636 then continue below.

1637

1638 If an AuditableEventQuery is specified within the TargetAssociationBranch, then let
1639 ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1640 the source object of some element of AF. If ROS is empty, then remove x from RO. RO
1641 is empty then continue below.

1642

1643 If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let
1644 ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1645 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1646 RO is empty then continue below.

1647

1648 If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let
1649 ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1650 the source object of some element of AF. If ROS is empty, then remove x from RO. If
1651 RO is empty then continue below.

1652

1653 If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the
1654 set of Service instances that satisfy the ServiceQuery and are the source object of some
1655 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue
1656 below.

1657

1658 If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be
1659 the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source
1660 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1661 then continue below.

1662

1663 If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let
1664 ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1665 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1666 RO is empty then continue below.

1667

1668 If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set
1669 of User instances that are the source object of some element of AF. If ROS is empty, then
1670 remove x from RO. If RO is empty then continue below. Let u be the member of ROS. If
1671 a UserFilter element is specified within the UserBranch, and if u does not satisfy that
1672 filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is
1673 empty then continue below. If a PostalAddressFilter element is specified within the
1674 UserBranch, and if the postal address of u does not satisfy that filter, then remove u from
1675 ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. If
1676 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
1677 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
1678 remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then
1679 continue below. If an OrganizationQuery element is specified within the UserBranch,
1680 then let o be the Organization instance that is identified by the organization that u is
1681 affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then
1682 remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then
1683 continue below.

1684
1685 If a ClassificationBranch is specified within the TargetAssociationBranch then let ROS
1686 be the set of Classification instances that are the source object of some element of AF. If
1687 ROS is empty, then remove x from RO. If RO is empty then continue below. Let cb be
1688 the member of ROS. If ClassificationFilter element is specified within the
1689 ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROS. If
1690 ROS is empty, then remove x from RO. If RO is empty then continue below. if a
1691 ClassificationSchemeQuery element is specified within the ClassificationBranch then
1692 replace ROS by the set of remaining Classification instances in ROS whose defining
1693 classification scheme satisfies the ClassificationSchemeQuery. If ROS is empty, then
1694 remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery
1695 element is specified within the ClassificationBranch, then replace ROS by the set of
1696 remaining Classification instances in ROS for which a classification node exists and for
1697 which that classification node satisfies the ClassificationNodeQuery. If ROS is empty,
1698 then remove x from RO. If RO is empty then continue below.

1699
1700 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1701 ROS be the set of ServiceBinding instances that are the source object of some element of
1702 AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let
1703 sb be the member of ROS. If a ServiceBindingFilter element is specified within the
1704 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROS. If
1705 ROS is empty then remove x from RO. If RO is empty then continue below. If a
1706 SpecificationLinkBranch is specified within the ServiceBindingBranch then consider
1707 each SpecificationLinkBranch element separately as follows:

1708 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link
1709 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1710 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1711 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then
1712 remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1713 element is specified within the SpecificationLinkBranch then let sl be a remaining
1714 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
1715 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
1716 specification link for some registry object in RO, then remove sl from SL. If SL is empty
1717 then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then
1718 continue below. If a RegistryEntryQuery element is specified within the
1719 SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
1720 RegistryEntryQuery element as follows: Let RE be the result set of the
1721 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
1722 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.
1723 If ROS is empty then remove x from RO. If RO is empty then continue below.
1724
1725 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
1726 ROS be the set of SpecificationLink instances that are the source object of some element
1727 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.
1728 Let sl be the member of ROS. If a SpecificationLinkFilter element is specified within the
1729 SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROS.
1730 If ROS is empty then remove x from RO. If RO is empty then continue below. If a
1731 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1732 be a remaining specification link in ROS. Treat RegistryObjectQuery element as follows:
1733 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
1734 not a specification link for some registry object in RO, then remove sl from ROS. If ROS
1735 is empty then remove x from RO. If RO is empty then continue below. If a
1736 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
1737 be a remaining specification link in ROS. Treat RegistryEntryQuery element as follows:
1738 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
1739 a specification link for some registry entry in RE, then remove sl from ROS. If ROS is
1740 empty then remove x from RO. If RO is empty then continue below
1741
1742 If a SourceAssociationBranch is specified within the TargetAssociationBranch, then let
1743 ROS be the set of RegistryObject instances that satisfy the SourceAssociationBranch and
1744 are the target object of some element of AF. If ROS is empty, then remove x from RO. If
1745 RO is empty then continue below.
1746
1747 If a TargetAssociationBranch is specified within the TargetAssociationBranch, then let
1748 ROS be the set of RegistryObject instances that satisfy the TargetAssociationBranch and
1749 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1750 RO is empty then continue below.
1751 f) If a ClassificationBranch element is not specified, or if RO is empty, then continue
1752 below; otherwise, let x be a remaining registry object in RO. If x is not the
1753 classifiedObject of some Classification instance, then remove x from RO; otherwise, treat
1754 each ClassificationBranch element separately as follows:

- 1755 If no ClassificationFilter is specified within the ClassificationBranch, then let CL be the
 1756 set of all Classification instances that have x as the classifiedObject; otherwise, let CL be
 1757 the set of Classification instances that satisfy the ClassificationFilter and have x as the
 1758 classifiedObject. If CL is empty, then remove x from RO and continue below.
 1759 Otherwise, if CL is not empty, and if a ClassificationSchemeQuery is specified, then
 1760 replace CL by the set of remaining Classification instances in CL whose defining
 1761 classification scheme satisfies the ClassificationSchemeQuery. If the new CL is empty,
 1762 then remove x from RO and continue below. Otherwise, if CL remains not empty, and if
 1763 a ClassificationNodeQuery is specified, then replace CL by the set of remaining
 1764 Classification instances in CL for which a classification node exists and for which that
 1765 classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then
 1766 remove x from RO. If RO is empty then continue below.
- 1767 g) If an ExternalIdentifierFilter element is not specified, or if RO is empty, then continue
 1768 below; otherwise, let x be a remaining registry object in RO. If x is not linked to some
 1769 ExternalIdentifier instance, then remove x from RO; otherwise, treat each
 1770 ExternalIdentifierFilter element separately as follows: Let EI be the set of
 1771 ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If
 1772 EI is empty, then remove x from RO. If RO is empty then continue below.
- 1773 h) If a SlotBranch element is not specified, or if RO is empty, then continue below;
 1774 otherwise, let x be a remaining registry object in RO. If x is not linked to some Slot
 1775 instance, then remove x from RO. If RO is empty then continue below; otherwise, treat
 1776 each SlotBranch element separately as follows: If a SlotFilter is not specified within the
 1777 SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set
 1778 of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,
 1779 then remove x from RO and continue below. Otherwise, if SL remains not empty, and if a
 1780 SlotValueFilter is specified, replace SL by the set of remaining Slot instances in SL for
 1781 which every specified SlotValueFilter is valid. If SL is empty, then remove x from RO.
- 1782 2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise,
 1783 return RO as the result of the RegistryObjectQuery.
- 1784 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 1785 RegistryObjectQuery.

1786 Examples

1787 A client application needs all items that are classified by two different classification schemes,
 1788 one based on "Industry" and another based on "Geography". Both schemes have been defined by
 1789 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",
 1790 respectively. The following query identifies registry entries for all registered items that are
 1791 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of
 1792 "Asia/Japan".

```

1793
1794 <AdhocQueryRequest>
1795   <ResponseOption returnType = "RegistryEntry"/>
1796   <FilterQuery>
1797     <RegistryObjectQuery>
1798       <ClassifiedByBranch>
1799         <ClassificationFilter>
1800           <Clause>
1801             <SimpleClause leftArgument = "path">
1802               <StringClause stringPredicate = "equal">//Automotive</StringClause>

```

```

1803     </SimpleClause>
1804   </Clause>
1805 </ClassificationFilter>
1806 <ClassificationSchemeQuery>
1807   <NameBranch>
1808     <LocalizedStringFilter>
1809       <Clause>
1810         <SimpleClause leftArgument = "value">
1811           <StringClause stringPredicate = "equal">urn:ebxml:cs:industry</StringClause>
1812         </SimpleClause>
1813       </Clause>
1814     </LocalizedStringFilter>
1815   </NameBranch>
1816 </ClassificationSchemeQuery>
1817 </ClassifiedByBranch>
1818 <ClassifiedByBranch>
1819   <ClassificationFilter>
1820     <Clause>
1821       <SimpleClause leftArgument = "path">
1822         <StringClause stringPredicate = "startswith">/Geography-id/Asia/Japan</StringClause>
1823       </SimpleClause>
1824     </Clause>
1825   </ClassificationFilter>
1826 </ClassificationSchemeQuery>
1827   <NameBranch>
1828     <LocalizedStringFilter>
1829       <Clause>
1830         <SimpleClause leftArgument = "value">
1831           <StringClause stringPredicate = "equal">urn:ebxml:cs:geography</StringClause>
1832         </SimpleClause>
1833       </Clause>
1834     </LocalizedStringFilter>
1835   </NameBranch>
1836 </ClassificationSchemeQuery>
1837 </ClassifiedByBranch>
1838 </RegistryObjectQuery>
1839 </FilterQuery>
1840 </AdhocQueryRequest>
1841

```

1842 A client application wishes to identify all RegistryObject instances that are classified by some
1843 internal classification scheme and have some given keyword as part of the description of one of
1844 the classification nodes of that classification scheme. The following query identifies all such
1845 RegistryObject instances. The query takes advantage of the knowledge that the classification
1846 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode
1847 instances.

```

1848
1849 <AdhocQueryRequest>
1850   <ResponseOption returnType = "RegistryObject"/>
1851   <FilterQuery>
1852     <RegistryObjectQuery>
1853       <ClassifiedByBranch>
1854         <ClassificationNodeQuery>
1855           <DescriptionBranch>
1856             <LocalizedStringFilter>
1857               <Clause>
1858                 <SimpleClause leftArgument = "value">

```

```

1859     <StringClause stringPredicate = "equal">transistor</StringClause>
1860     </SimpleClause>
1861     </Clause>
1862     </LocalizedStringFilter>
1863     </DescriptionBranch>
1864     </ClassificationNodeQuery>
1865     </ClassifiedByBranch>
1866     </RegistryObjectQuery>
1867 </FilterQuery>
1868 </AdhocQueryRequest>
1869

```

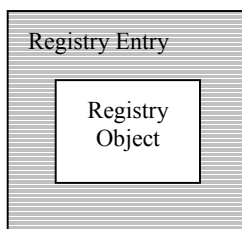
1870 8.2.3 RegistryEntryQuery

1871 Purpose

1872 To identify a set of registry entry instances as the result of a query over selected registry
 1873 metadata.

1874 ebRIM Binding

1875



1876 **Figure 18: ebRIM Binding for RegistryEntryQuery**

1877 Definition

```

1878
1879 <complexType name="RegistryEntryQueryType">
1880   <complexContent>
1881     <extension base="tns:RegistryObjectQueryType">
1882       <sequence>
1883         <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
1884       </sequence>
1885     </extension>
1886   </complexContent>
1887 </complexType>
1888 <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
1889
1890 <element name="RegistryEntryQueryResult">
1891   <complexType>
1892     <choice minOccurs="0" maxOccurs="unbounded">
1893       <element ref="rim:ObjectRef" />
1894       <element ref="rim:ClassificationScheme" />
1895       <element ref="rim:ExtrinsicObject" />
1896       <element ref="rim:RegistryEntry" />
1897       <element ref="rim:RegistryObject" />
1898       <element ref="rim:RegistryPackage" />
1899     </choice>
1900   </complexType>
1901 </element>
1902

```


1903 **Semantic Rules**

- 1904 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
 1905 steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
- 1906 a) If RE is empty then continue below.
- 1907 b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
 1908 registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
 1909 If RE is empty then continue below.
- 1910 c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
 1911 RegistryObjectQuery over RE as explained in section 8.2.2.
- 1912 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return
 1913 RE as the result of the RegistryEntryQuery.
- 1914 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 1915 RegistryEntryQuery.

1916 **Examples**

1917 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if
 1918 they have registered any of their business documents in the Registry. The following query
 1919 returns a set of registry entry identifiers for currently registered items submitted by any
 1920 organization whose name includes the string "XYZ". It does not return any registry entry
 1921 identifiers for superseded, replaced, deprecated, or withdrawn items.

```

1922
1923 <AdhocQueryRequest>
1924   <ResponseOption returnType = "ObjectRef"/>
1925   <FilterQuery>
1926     <RegistryEntryQuery>
1927       <SourceAssociationBranch>
1928         <AssociationFilter>
1929           <Clause>
1930             <SimpleClause leftArgument = "associationType">
1931               <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
1932             </SimpleClause>
1933           </Clause>
1934         </AssociationFilter>
1935       <OrganizationQuery>
1936         <NameBranch>
1937           <LocalizedStringFilter>
1938             <Clause>
1939               <SimpleClause leftArgument = "value">
1940                 <StringClause stringPredicate = "contains">XYZ</StringClause>
1941               </SimpleClause>
1942             </Clause>
1943           </LocalizedStringFilter>
1944         </NameBranch>
1945       </OrganizationQuery>
1946     </SourceAssociationBranch>
1947   <RegistryEntryFilter>
1948     <Clause>
1949       <SimpleClause leftArgument = "status">
1950         <StringClause stringPredicate = "equal">Approved</StringClause>
1951       </SimpleClause>
1952     </Clause>
1953   </RegistryEntryFilter>
  
```

```

1954 </RegistryEntryQuery>
1955 </FilterQuery>
1956 </AdhocQueryRequest>
1957

```

1958 A client is using the United Nations Standard Product and Services Classification (UNSPSC)
 1959 scheme and wants to identify all companies that deal with products classified as "Integrated
 1960 circuit components", i.e. UNSPSC code "321118". The client knows that companies have
 1961 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
 1962 such profile has been classified by UNSPSC according to the products the company deals with.
 1963 However, the client does not know if the UNSPSC classification scheme is internal or external to
 1964 this registry. The following query returns a set of approved registry entry instances for CPP's of
 1965 companies that deal with integrated circuit components.

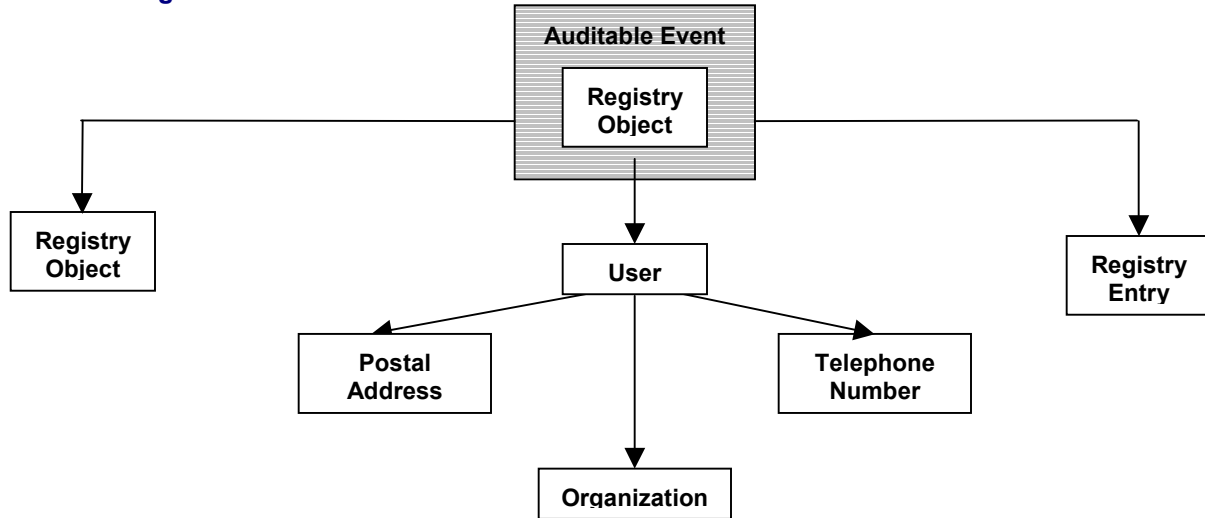
```

1966 <AdhocQueryRequest>
1967 <ResponseOption returnType = "RegistryEntry"/>
1968 <FilterQuery>
1969 <RegistryEntryQuery>
1970 <ClassifiedByBranch>
1971 <ClassificationFilter>
1972 <Clause>
1973 <SimpleClause leftArgument = "code">
1974 <StringClause stringPredicate = "equal">321118</StringClause>
1975 </SimpleClause>
1976 </Clause>
1977 </ClassificationFilter>
1978 <ClassificationSchemeQuery>
1979 <NameBranch>
1980 <LocalizedStringFilter>
1981 <Clause>
1982 <SimpleClause leftArgument = "value">
1983 <StringClause stringPredicate = "equal">urn:org:un:spsc:cs2001</StringClause>
1984 </SimpleClause>
1985 </Clause>
1986 </LocalizedStringFilter>
1987 </NameBranch>
1988 </ClassificationSchemeQuery>
1989 </ClassifiedByBranch>
1990 <RegistryEntryFilter>
1991 <Clause>
1992 <CompoundClause connectivePredicate = "And">
1993 <Clause>
1994 <SimpleClause leftArgument = "objectType">
1995 <StringClause stringPredicate = "equal">CPP</StringClause>
1996 </SimpleClause>
1997 </Clause>
1998 <Clause>
1999 <SimpleClause leftArgument = "status">
2000 <StringClause stringPredicate = "equal">Approved</StringClause>
2001 </SimpleClause>
2002 </Clause>
2003 </CompoundClause>
2004 </Clause>
2005 </RegistryEntryFilter>
2006 </RegistryEntryQuery>
2007 </FilterQuery>
2008 </AdhocQueryRequest>
2009
2010

```

2011 **8.2.4 AuditableEventQuery**2012 **Purpose**

2013 To identify a set of auditable event instances as the result of a query over selected registry
 2014 metadata.

2015 **ebRIM Binding**

2016 **Figure 19: ebRim binding for AuditableEventQuery**

2017 **Definition**

```

2018 <complexType name="AuditableEventQueryType">
2019   <complexContent>
2020     <extension base="tns:RegistryObjectQueryType">
2021       <sequence>
2022         <element ref="tns:AuditableEventFilter" minOccurs="0" />
2023         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2024         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2025         <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2026       </sequence>
2027     </extension>
2028   </complexContent>
2029 </complexType>
2030 <element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />
2031
2032 <element name="AuditableEventQueryResult">
2033   <complexType>
2034     <choice minOccurs="0" maxOccurs="unbounded">
2035       <element ref="rim:ObjectRef" />
2036       <element ref="rim:RegistryObject" />
2037       <element ref="rim:AuditableEvent" />
2038     </choice>
2039   </complexType>
2040 </element>
2041
2042
  
```

2043 **Semantic Rules**

- 2044 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
 2045 following steps will eliminate instances in AE that do not satisfy the conditions of the
 2046 specified filters.

- 2047 a) If AE is empty then continue below.
- 2048 b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an
2049 auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from
2050 AE. If AE is empty then continue below.
- 2051 c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
2052 x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:
2053 Let RO be the result set of the RegistryObjectQuery as defined in [Section 8.2.2](#). If x is
2054 not an auditable event for some registry object in RO, then remove x from AE. If AE is
2055 empty then continue below.
- 2056 d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
2057 be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let
2058 RE be the result set of the RegistryEntryQuery as defined in [Section 8.2.3](#). If x is not an
2059 auditable event for some registry entry in RE, then remove x from AE. If AE is empty
2060 then continue below.
- 2061 e) If an UserBranch element is not specified then go to the next step; otherwise, let x be a
2062 remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter
2063 element is specified within the UserBranch, and if u does not satisfy that filter, then
2064 remove x from AE. If a PostalAddressFilter element is specified within the UserBranch,
2065 and if the postal address of u does not satisfy that filter, then remove x from AE. If
2066 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
2067 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
2068 remove x from AE. If an OrganizationQuery element is specified within the UserBranch,
2069 then let o be the Organization instance that is identified by the organization that u is
2070 affiliated with. If o doesn't satisfy OrganizationQuery as defined in [Section 8.2.9](#) then
2071 remove x from AE. If AE is empty then continue below.
- 2072 f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited
2073 RegistryObjectQuery over AE as explained in [section 8.2.2](#).
- 2074 2. If AE is empty, then raise the warning: *auditable event query result is empty*; otherwise
2075 return AE as the result of the AuditableEventQuery.
- 2076 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
2077 AuditableEventQuery.

2078 Examples

2079 A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The
2080 client is now interested in all events since the beginning of the year that have impacted that item.
2081 The following query will return a set of AuditableEvent instances for all such events.

```
2082
2083 <AdhocQueryRequest>
2084   <ResponseOption returnType = "LeafClass"/>
2085   <FilterQuery>
2086     <AuditableEventQuery>
2087       <TargetAssociationBranch>
2088         <AssociationFilter>
2089           <Clause>
2090             <SimpleClause leftArgument = "associationType">
2091               <StringClause stringPredicate = "equal">AuditedBy</StringClause>
2092             </SimpleClause>
2093           </Clause>
```

```

2094     </AssociationFilter>
2095     <RegistryEntryQuery>
2096       <NameBranch>
2097         <LocalizedStringFilter>
2098           <Clause>
2099             <SimpleClause leftArgument = "value">
2100               <StringClause stringPredicate = "equal">urn:path:myitem</StringClause>
2101             </SimpleClause>
2102           </Clause>
2103         </LocalizedStringFilter>
2104       </NameBranch>
2105     </RegistryEntryQuery>
2106   </TargetAssociationBranch>
2107 <AuditableEventFilter>
2108   <Clause>
2109     <SimpleClause leftArgument = "timestamp">
2110       <RationalClause logicalPredicate = "GE">
2111         <IntClause>20000101</IntClause>
2112       </RationalClause>
2113     </SimpleClause>
2114   </Clause>
2115 </AuditableEventFilter>
2116 </AuditableEventQuery>
2117 </FilterQuery>
2118 </AdhocQueryRequest
2119

```

2120 A client company has many registered objects in the Registry. The Registry allows events
 2121 submitted by other organizations to have an impact on your registered items, e.g. new
 2122 classifications and new associations. The following query will return a set of identifiers for all
 2123 auditable events, invoked by some other party, that had an impact on an item submitted by
 2124 “myorg”.

```

2125
2126 <AdhocQueryRequest>
2127   <ResponseOption returnType = "LeafClass"/>
2128   <FilterQuery>
2129     <AuditableEventQuery>
2130       <TargetAssociationBranch>
2131         <AssociationFilter>
2132           <Clause>
2133             <SimpleClause leftArgument = "associationType">
2134               <StringClause stringPredicate = "equal">AuditedBy</StringClause>
2135             </SimpleClause>
2136           </Clause>
2137         </AssociationFilter>
2138       <RegistryEntryQuery>
2139         <SourceAssociationBranch>
2140           <AssociationFilter>
2141             <Clause>
2142               <SimpleClause leftArgument = "associationType">
2143                 <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
2144               </SimpleClause>
2145             </Clause>
2146           </AssociationFilter>
2147         <OrganizationQuery>
2148           <NameBranch>
2149             <LocalizedStringFilter>
2150               <Clause>
2151                 <SimpleClause leftArgument = "value">

```

```

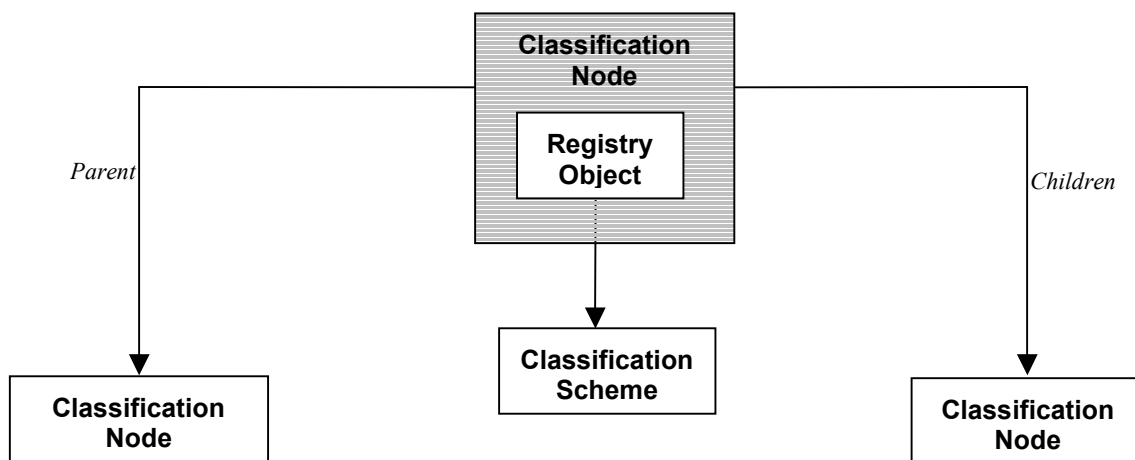
2152     <StringClause stringPredicate = "equal">myorg</StringClause>
2153     </SimpleClause>
2154     </Clause>
2155     </LocalizedStringFilter>
2156     </NameBranch>
2157     </OrganizationQuery>
2158     </SourceAssociationBranch>
2159     </RegistryEntryQuery>
2160     </TargetAssociationBranch>
2161     <UserBranch>
2162     <OrganizationQuery>
2163     <NameBranch>
2164     <LocalizedStringFilter>
2165     <Clause>
2166     <SimpleClause leftArgument = "value">
2167     <StringClause stringPredicate = "-equal">myorg</StringClause>
2168     </SimpleClause>
2169     </Clause>
2170     </LocalizedStringFilter>
2171     </NameBranch>
2172     </OrganizationQuery>
2173     </UserBranch>
2174     </AuditableEventQuery>
2175     </FilterQuery>
2176 </AdhocQueryRequest>
2177

```

2178 8.2.5 ClassificationNodeQuery

2179 Purpose

2180 To identify a set of classification node instances as the result of a query over selected registry
 2181 metadata.



2182 ebRIM Binding

2183

Figure 20: ebRim binding for ClassificationNodeQuery

2184 Definition

```

2185 <complexType name="ClassificationNodeQueryType">
2186   <complexContent>
2187     <extension base="tns:RegistryObjectQueryType">
2188

```

```

2189 <sequence>
2190 <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
2191 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
2192 <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
2193 maxOccurs="1" />
2194 <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
2195 minOccurs="0" maxOccurs="unbounded" />
2196 </sequence>
2197 </extension>
2198 </complexContent>
2199 </complexType>
2200 <element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />
2201
2202 <element name="ClassificationNodeQueryResult">
2203 <complexType>
2204 <choice minOccurs="0" maxOccurs="unbounded">
2205 <element ref="rim:ObjectRef" />
2206 <element ref="rim:RegistryObject" />
2207 <element ref="rim:ClassificationNode" />
2208 </choice>
2209 </complexType>
2210 </element>
2211

```

2212 Semantic Rules

- 2213 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The
2214 following steps will eliminate instances in CN that do not satisfy the conditions of the
2215 specified filters.
 - 2216 a) If CN is empty then continue below.
 - 2217 b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be
2218 a classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove
2219 x from CN. If CN is empty then continue below.
 - 2220 c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
2221 be a remaining classification node in CN. If the defining classification scheme of x does
2222 not satisfy the ClassificationSchemeQuery as defined in [section 8.2.6](#), then remove x
2223 from CN. If CN is empty then continue below.
 - 2224 d) If a ClassificationNodeParentBranch element is not specified, then go to the next step;
2225 otherwise, let x be a remaining classification node in CN and execute the following
2226 paragraph with n=x.
2227 Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base
2228 level node), then remove x from CN and go to the next step; otherwise, let p be the parent
2229 node of n. If a ClassificationNodeFilter element is directly contained in the
2230 ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,
2231 then remove x from CN. If CN is empty then continue below. If a
2232 ClassificationSchemeQuery element is directly contained in the
2233 ClassificationNodeParentBranch and if defining classification scheme of p does not
2234 satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then
2235 continue below.
2236 If another ClassificationNodeParentBranch element is directly contained within this
2237 ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.

- 2238 e) If a ClassificationNodeChildrenBranch element is not specified, then continue below;
 2239 otherwise, let x be a remaining classification node in CN. If x is not the parent node of
 2240 some ClassificationNode instance, then remove x from CN and if CN is empty continue
 2241 below; otherwise, treat each ClassificationNodeChildrenBranch element separately and
 2242 execute the following paragraph with n = x.
- 2243 Let n be a classification node instance. If a ClassificationNodeFilter element is not
 2244 specified within the ClassificationNodeChildrenBranch element then let CNC be the set
 2245 of all classification nodes that have n as their parent node; otherwise, let CNC be the set
 2246 of all classification nodes that satisfy the ClassificationNodeFilter and have n as their
 2247 parent node. If CNC is empty, then remove x from CN and if CN is empty continue
 2248 below; otherwise, let c be any member of CNC. If a ClassificationSchemeQuery element
 2249 is directly contained in the ClassificationNodeChildrenBranch and if the defining
 2250 classification scheme of c does not satisfy the ClassificationSchemeQuery then remove c
 2251 from CNC. If CNC is empty then remove x from CN. If CN is empty then continue
 2252 below; otherwise, let y be an element of CNC and continue with the next paragraph.
- 2253 If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly
 2254 contain another ClassificationNodeChildrenBranch element, then continue below;
 2255 otherwise, repeat the previous paragraph with the new ClassificationNodeChildrenBranch
 2256 element and with n = y.
- 2257 f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited
 2258 RegistryObjectQuery over CN as explained in [section 8.2.2](#).
- 2259 2. If CN is empty, then raise the warning: **classification node query result is empty**; otherwise
 2260 return CN as the result of the ClassificationNodeQuery.
- 2261 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2262 ClassificationNodeQuery.

2263 Path Filter Expression usage in ClassificationNodeFilter

2264 The path filter expression is used to match classification nodes in ClassificationNodeFilter
 2265 elements involving the path attribute of the ClassificationNode class as defined by the getPath
 2266 method in [ebRIM].

2267 The path filter expressions are based on a very small and proper sub-set of location path syntax
 2268 of XPath.

2269 The path filter expression syntax includes support for matching multiple nodes by using wild
 2270 card syntax as follows:

- 2271 • Use of '*' as a wildcard in place of any path element in the pathFilter
- 2272 • Use of '/' syntax to denote any descendent of a node in the pathFilter

2273 It is defined by the following BNF grammar:

```

2274 pathFilter ::= '/' schemeId nodePath
2275 nodePath  ::= slashes nodeCode
2276           | slashes '*'
2277           | slashes nodeCode ( nodePath )?
2278 Slashes ::= '/' | '/'
2279
2280
```

2281 In the above grammar, schemeId is the id attribute of the ClassificationScheme instance. In the
 2282 above grammar nodeCode is defined by NCName production as defined by
 2283 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

2284 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a
 2285 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause

2286 is a PATH Filter expression.

2287 This is illustrated in the following example that matches all second level nodes in
2288 ClassificationScheme with id 'Geography-id' and with code 'Japan':

```

2289
2290 <ClassificationNodeQuery>
2291   <ClassificationNodeFilter>
2292     <Clause>
2293       <SimpleClause leftArgument = "path">
2294         <StringClause stringPredicate = "equal">//Geography-id/*/Japan</StringClause>
2295       </SimpleClause>
2296     </Clause>
2297   </ClassificationNodeFilter>
2298 </ClassificationNodeQuery>
2299

```

2300 Use Cases and Examples of Path Filter Expressions

2301 The following table lists various use cases and examples using the sample Geography scheme
2302 below:

```

2303
2304 <ClassificationScheme id='Geography-id' name="Geography" />
2305
2306 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
2307 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
2308
2309 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
2310 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
2311 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
2312

```

2313

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'

2314 Examples

2315 A client application wishes to identify all of the classification nodes in the first three levels of a
2316 classification scheme hierarchy. The client knows that the name of the underlying classification
2317 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three

2318 levels.

```

2319 <AdhocQueryRequest>
2320   <ResponseOption returnType = "LeafClass"/>
2321   <FilterQuery>
2322     <ClassificationNodeQuery>
2323       <ClassificationNodeFilter>
2324         <Clause>
2325           <SimpleClause leftArgument = "levelNumber">
2326             <RationalClause logicalPredicate = "LE">
2327               <IntClause>3</IntClause>
2328             </RationalClause>
2329           </SimpleClause>
2330         </Clause>
2331       </ClassificationNodeFilter>
2332     </ClassificationNodeQuery>
2333     <ClassificationSchemeQuery>
2334       <NameBranch>
2335         <LocalizedStringFilter>
2336           <Clause>
2337             <SimpleClause leftArgument = "value">
2338               <StringClause stringPredicate = "equal">urn:ebxml:cs:myscheme</StringClause>
2339             </SimpleClause>
2340           </Clause>
2341         </LocalizedStringFilter>
2342       </NameBranch>
2343     </ClassificationSchemeQuery>
2344   </FilterQuery>
2345 </AdhocQueryRequest>
2346
2347

```

2348 If, instead, the client wishes all levels returned, they could simply delete the
 2349 ClassificationNodeFilter element from the query.

2350 The following query finds all children nodes of a first level node whose code is NorthAmerica.

```

2351 <AdhocQueryRequest>
2352   <ResponseOption returnType = "LeafClass"/>
2353   <FilterQuery>
2354     <ClassificationNodeQuery>
2355       <ClassificationNodeFilter>
2356         <Clause>
2357           <SimpleClause leftArgument = "path">
2358             <StringClause stringPredicate = "equal">/Geography-id/NorthAmerica/*</StringClause>
2359           </SimpleClause>
2360         </Clause>
2361       </ClassificationNodeFilter>
2362     </ClassificationNodeQuery>
2363   </FilterQuery>
2364 </AdhocQueryRequest>
2365
2366

```

2367 The following query finds all third level nodes with code of Tokyo.

```

2368 <AdhocQueryRequest>
2369   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2370   <FilterQuery>
2371     <ClassificationNodeQuery>
2372       <ClassificationNodeFilter xmlns = "">
2373         <Clause>
2374           <SimpleClause leftArgument = "path">
2375

```

```

2376     <StringClause stringPredicate = "equal">/Geography-id/**/Tokyo</StringClause>
2377     </SimpleClause>
2378     </Clause>
2379     </ClassificationNodeFilter>
2380     </ClassificationNodeQuery>
2381     </FilterQuery>
2382 </AdhocQueryRequest>
2383

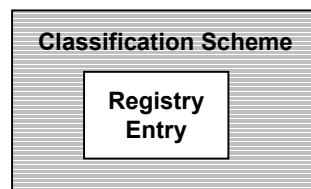
```

2384 8.2.6 ClassificationSchemeQuery

2385 Purpose

2386 To identify a set of classification scheme instances as the result of a query over selected registry
 2387 metadata.

2388 ebRIM Binding



2389 **Figure 21: ebRIM Binding for ClassificationSchemeQuery**

2390 Definition

```

2391 <complexType name="ClassificationSchemeQueryType">
2392   <complexContent>
2393     <extension base="tns:RegistryEntryQueryType">
2394       <sequence>
2395         <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
2396       </sequence>
2397     </extension>
2398   </complexContent>
2399 </complexType>
2400 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
2401
2402

```

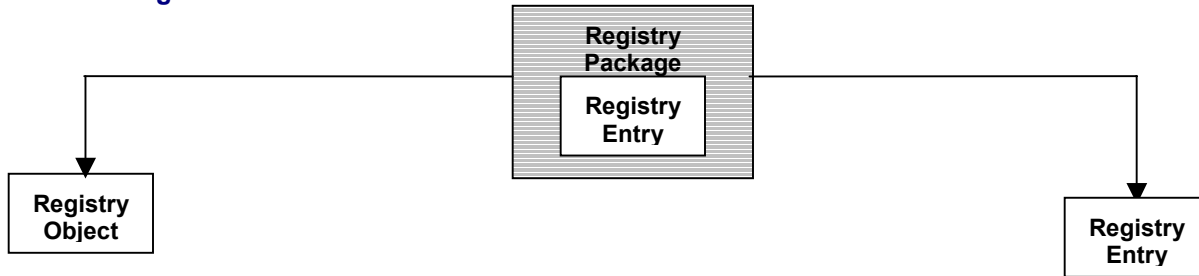
2403 Semantic Rules

- 2404 1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The
 2405 following steps will eliminate instances in CS that do not satisfy the conditions of the
 2406 specified filters.
 - 2407 a) If CS is empty then continue below.
 - 2408 b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x
 2409 be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,
 2410 then remove x from CS. If CS is empty then continue below.
 - 2411 c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited
 2412 RegistryEntryQuery over CS as explained in section 8.2.3.
- 2413 2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,
 2414 return CS as the result of the ClassificationSchemeQuery.
- 2415 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2416 ClassificationSchemeQuery.

2417

2418 **8.2.7 RegistryPackageQuery**2419 **Purpose**

2420 To identify a set of registry package instances as the result of a query over selected registry
2421 metadata.

2422 **ebRIM Binding**

2423

Figure 22: ebRim binding for RegistryPackageQuery

2424 **Definition**

```

2425 <complexType name="RegistryPackageQueryType">
2426   <complexContent>
2427     <extension base="tns:RegistryEntryQueryType">
2428       <sequence>
2429         <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
2430         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
2431         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
2432       </sequence>
2433     </extension>
2434   </complexContent>
2435 </complexType>
2436 <element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />
2437
2438 <element name="RegistryPackageQueryResult">
2439   <complexType>
2440     <choice minOccurs="0" maxOccurs="unbounded">
2441       <element ref="rim:ObjectRef" />
2442       <element ref="rim:RegistryEntry" />
2443       <element ref="rim:RegistryObject" />
2444       <element ref="rim:RegistryPackage" />
2445     </choice>
2446   </complexType>
2447 </element>
2448
2449
  
```

2450 **Semantic Rules**

- 2451 1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The
2452 following steps will eliminate instances in RP that do not satisfy the conditions of the
2453 specified filters.
- 2454 a) If RP is empty then continue below.

- 2455 b) If a RegistryPackageFilter is not specified, then continue below; otherwise, let x be a
 2456 registry package instance in RP. If x does not satisfy the RegistryPackageFilter then
 2457 remove x from RP. If RP is empty then continue below.
- 2458 c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery
 2459 element then let RO be the set of RegistryObject instances returned by the
 2460 RegistryObjectQuery as defined in Section 8.2.2 and let PO be the subset of RO that are
 2461 members of the package x. If PO is empty, then remove x from RP. If RP is empty then
 2462 continue below. If a RegistryEntryQuery element is directly contained in the
 2463 RegistryPackageQuery element then let RE be the set of RegistryEntry instances returned
 2464 by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be the subset of RE that
 2465 are members of the package x. If PE is empty, then remove x from RP.
- 2466 d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited
 2467 RegistryEntryQuery over RP as explained in section 8.2.3.
- 2468 2. If RP is empty, then raise the warning: *registry package query result is empty*; otherwise
 2469 return RP as the result of the RegistryPackageQuery.
- 2470 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2471 RegistryPackageQuery.

2472 Examples

2473 A client application wishes to identify all package instances in the Registry that contain an
 2474 Invoice extrinsic object as a member of the package.

```

2475 <AdhocQueryRequest>
2476   <ResponseOption returnType = "LeafClass"/>
2477   <FilterQuery>
2478     <RegistryPackageQuery>
2479       <RegistryEntryQuery>
2480         <RegistryEntryFilter>
2481           <Clause>
2482             <SimpleClause leftArgument = "objectType">
2483               <StringClause stringPredicate = "equal">Invoice</StringClause>
2484             </SimpleClause>
2485           </Clause>
2486         </RegistryEntryFilter>
2487       </RegistryEntryQuery>
2488     </RegistryPackageQuery>
2489   </FilterQuery>
2490 </AdhocQueryRequest>
  
```

2493 A client application wishes to identify all package instances in the Registry that are not empty.

```

2494 <AdhocQueryRequest>
2495   <ResponseOption returnType = "LeafClass"/>
2496   <FilterQuery>
2497     <RegistryPackageQuery/>
2498   </FilterQuery>
2499 </AdhocQueryRequest>
  
```

2502 A client application wishes to identify all package instances in the Registry that are empty. Since
 2503 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate
 2504 RegistryPackageQuery requests, one to find all packages and another to find all non-empty
 2505 packages, and then do the set difference themselves. Alternatively, they could do a more

2506 complex RegistryEntryQuery and check that the packaging association between the package and
2507 its members is non-existent.

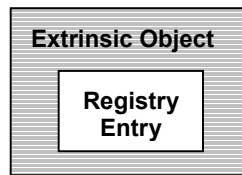
2508 **Note:** A registry package is an intrinsic RegistryEntry instance that is completely determined by
2509 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an
2510 equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However,
2511 the equivalent RegistryEntryQuery is often more complicated to write.

2512 8.2.8 ExtrinsicObjectQuery

2513 Purpose

2514 To identify a set of extrinsic object instances as the result of a query over selected registry
2515 metadata.

2516 ebRIM Binding



2517 Figure 23:ebRIM Binding for ExtrinsicObjectQuery

2518 Definition

```

2519 <complexType name="ExtrinsicObjectQueryType">
2520   <complexContent>
2521     <extension base="tns:RegistryEntryQueryType">
2522       <sequence>
2523         <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
2524       </sequence>
2525     </extension>
2526   </complexContent>
2527 </complexType>
2528 </complexType>
2529 <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
2530
2531 <element name="ExtrinsicObjectQueryResult">
2532   <complexType>
2533     <choice minOccurs="0" maxOccurs="unbounded">
2534       <element ref="rim:ObjectRef" />
2535       <element ref="rim:RegistryEntry" />
2536       <element ref="rim:RegistryObject" />
2537       <element ref="rim:ExtrinsicObject" />
2538     </choice>
2539   </complexType>
2540 </element>
2541
  
```

2542 Semantic Rules

- 2543 1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The
2544 following steps will eliminate instances in EO that do not satisfy the conditions of the
2545 specified filters.
- 2546 a) If EO is empty then continue below.

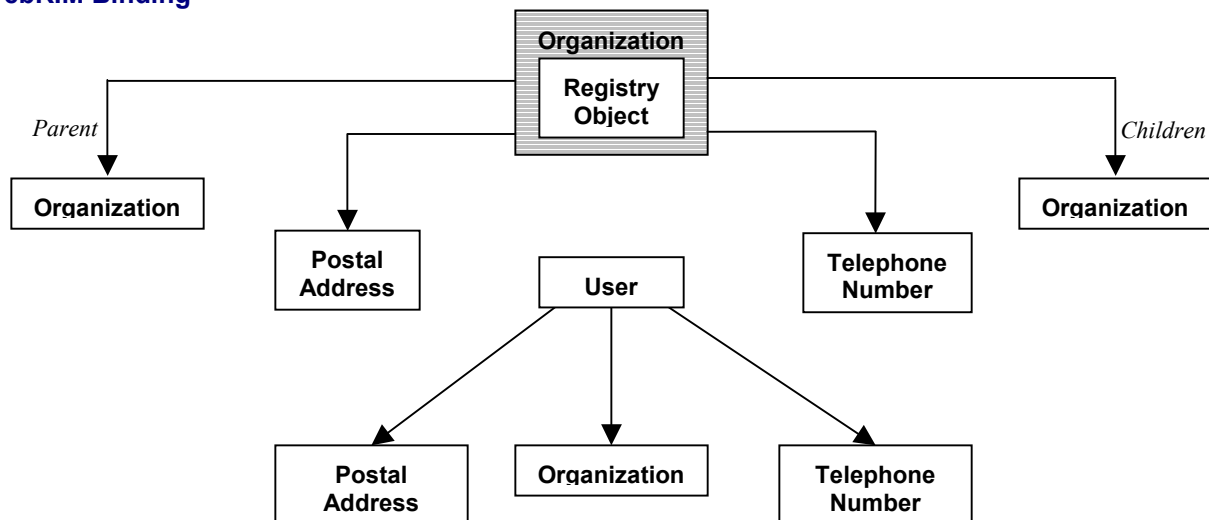
- 2547 b) If a `ExtrinsicObjectFilter` is not specified then go to the next step; otherwise, let `x` be an
 2548 extrinsic object in `EO`. If `x` does not satisfy the `ExtrinsicObjectFilter` then remove `x` from
 2549 `EO`. If `EO` is empty then continue below.
- 2550 c) Let `EO` be the set of remaining `ExtrinsicObject` instances. Evaluate inherited
 2551 `RegistryEntryQuery` over `EO` as explained in section 8.2.3.
- 2552 2. If `EO` is empty, then raise the warning: *extrinsic object query result is empty*; otherwise,
 2553 return `EO` as the result of the `ExtrinsicObjectQuery`.
- 2554 3. Return any accumulated warnings or exceptions as the `StatusResult` associated with the
 2555 `ExtrinsicObjectQuery`.

2556 8.2.9 OrganizationQuery

2557 Purpose

2558 To identify a set of organization instances as the result of a query over selected registry
 2559 metadata.

2560 ebRIM Binding



2561 Figure 24: ebRim Binding for OrganizationQuery

2562 Definition

```

2563 <complexType name="OrganizationQueryType">
2564   <complexContent>
2565     <extension base="tns:RegistryObjectQueryType">
2566       <sequence>
2567         <element ref="tns:OrganizationFilter" minOccurs="0" maxOccurs="1" />
2568         <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
2569         <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
2570         <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2571         <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0"
2572           maxOccurs="1" />
2573         <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
2574           maxOccurs="unbounded" />
2575       </sequence>
2576     </extension>
2577   </complexContent>
2578 </complexType>
2579 
```

```

2580 <element name="OrganizationQuery" type="tns:OrganizationQueryType" />
2581
2582 <element name="OrganizationQueryResult">
2583   <complexType>
2584     <choice minOccurs="0" maxOccurs="unbounded">
2585       <element ref="rim:ObjectRef" />
2586       <element ref="rim:RegistryObject" />
2587       <element ref="rim:Organization" />
2588     </choice>
2589   </complexType>
2590 </element>
2591

```

2592 **Semantic Rules**

- 2593 1. Let ORG denote the set of all persistent Organization instances in the Registry. The
 2594 following steps will eliminate instances in ORG that do not satisfy the conditions of the
 2595 specified filters.
- 2596 a) If ORG is empty then continue below.
- 2597 b) If an OrganizationFilter element is not directly contained in the OrganizationQuery
 2598 element, then go to the next step; otherwise let x be an organization instance in ORG. If x
 2599 does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then
 2600 continue below.
- 2601 c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery
 2602 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal
 2603 address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is
 2604 empty then continue below.
- 2605 d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery
 2606 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of
 2607 the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then
 2608 remove x from ORG. If ORG is empty then continue below.
- 2609 e) If a UserBranch element is not directly contained in the OrganizationQuery element then
 2610 go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user
 2611 instance that is affiliated with x. If a UserFilter element is specified within the
 2612 UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a
 2613 PostalAddressFilter element is specified within the UserBranch, and if the postal address
 2614 of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s)
 2615 are specified within the UserBranch and if any of the TelephoneNumberFilters isn't
 2616 satisfied by some of the telephone numbers of x then remove x from ORG. If an
 2617 OrganizationQuery element is specified within the UserBranch, then let o be the
 2618 Organization instance that is identified by the organization that u is affiliated with. If o
 2619 doesn't satisfy OrganizationQuery as defined in section 8.2.9 then remove x from ORG.
 2620 If ORG is empty then continue below.
- 2621 f) If a OrganizationParentBranch element is not specified within the OrganizationQuery,
 2622 then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the
 2623 following paragraph with o = x:

- 2624 Let *o* be an organization instance. If an `OrganizationFilter` is not specified within the
 2625 `OrganizationParentBranch` and if *o* has no parent (i.e. if *o* is a root organization in the
 2626 Organization hierarchy), then remove *x* from `ORG`; otherwise, let *p* be the parent
 2627 organization of *o*. If *p* does not satisfy the `OrganizationFilter`, then remove *x* from `ORG`.
 2628 If `ORG` is empty then continue below.
- 2629 If another `OrganizationParentBranch` element is directly contained within this
 2630 `OrganizationParentBranch` element, then repeat the previous paragraph with *o* = *p*.
- 2631 g) If a `OrganizationChildrenBranch` element is not specified, then continue below;
 2632 otherwise, let *x* be a remaining organization in `ORG`. If *x* is not the parent node of some
 2633 organization instance, then remove *x* from `ORG` and if `ORG` is empty continue below;
 2634 otherwise, treat each `OrganizationChildrenBranch` element separately and execute the
 2635 following paragraph with *n* = *x*.
- 2636 Let *n* be an organization instance. If an `OrganizationFilter` element is not specified within
 2637 the `OrganizationChildrenBranch` element then let `ORGC` be the set of all organizations
 2638 that have *n* as their parent node; otherwise, let `ORGC` be the set of all organizations that
 2639 satisfy the `OrganizationFilter` and have *n* as their parent node. If `ORGC` is empty, then
 2640 remove *x* from `ORG` and if `ORG` is empty continue below; otherwise, let *c* be any
 2641 member of `ORGC`. If a `PostalAddressFilter` element is directly contained in the
 2642 `OrganizationChildrenBranch` and if the postal address of *c* does not satisfy the
 2643 `PostalAddressFilter` then remove *c* from `ORGC`. If `ORGC` is empty then remove *x* from
 2644 `ORG`. If `ORG` is empty then continue below. If no `TelephoneNumberFilter` element is
 2645 directly contained in the `OrganizationChildrenBranch` and if If any of the
 2646 `TelephoneNumberFilters` isn't satisfied by some of the telephone numbers of *c* then
 2647 remove *c* from `ORGC`. If `ORGC` is empty then remove *x* from `ORG`. If `ORG` is empty
 2648 then continue below; otherwise, let *y* be an element of `ORGC` and continue with the next
 2649 paragraph.
- 2650 If the `OrganizationChildrenBranch` element is terminal, i.e. if it does not directly contain
 2651 another `OrganizationChildrenBranch` element, then continue below; otherwise, repeat the
 2652 previous paragraph with the new `OrganizationChildrenBranch` element and with *n* = *y*.
- 2653 h) Let `ORG` be the set of remaining Organization instances. Evaluate inherited
 2654 `RegistryObjectQuery` over `ORG` as explained in section 8.2.2.
- 2655 2. If `ORG` is empty, then raise the warning: *organization query result is empty*; otherwise return
 2656 `ORG` as the result of the `OrganizationQuery`.
- 2657 3. Return any accumulated warnings or exceptions as the `StatusResult` associated with the
 2658 `OrganizationQuery`.

2659 Examples

2660 A client application wishes to identify a set of organizations, based in France, that have
 2661 submitted a `PartyProfile` extrinsic object this year.

```

2662 <AdhocQueryRequest>
2663   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2664   <FilterQuery>
2665     <OrganizationQuery>
2666       <TargetAssociationBranch>
2667         <AssociationFilter>
2668           <Clause>
2669             <SimpleClause leftArgument = "associationType">
2670               <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
2671             </SimpleClause>
2672           </Clause>
2673         </AssociationFilter>
2674       </TargetAssociationBranch>
2675     </OrganizationQuery>
2676   </FilterQuery>
2677 </AdhocQueryRequest>
  
```

```

2674     </AssociationFilter>
2675     <RegistryObjectQuery>
2676       <RegistryObjectFilter>
2677         <Clause>
2678           <SimpleClause leftArgument = "objectType">
2679             <StringClause stringPredicate = "equal">CPP</StringClause>
2680           </SimpleClause>
2681         </Clause>
2682       </RegistryObjectFilter>
2683       <SourceAssociationBranch>
2684         <AssociationFilter>
2685           <Clause>
2686             <SimpleClause leftArgument = "associationType">
2687               <StringClause stringPredicate =
2688 "equal">AuditedBy</StringClause>
2689             </SimpleClause>
2690           </Clause>
2691         </AssociationFilter>
2692         <AuditableEventQuery>
2693           <AuditableEventFilter>
2694             <Clause>
2695               <SimpleClause leftArgument = "timestamp">
2696                 <RationalClause logicalPredicate = "GE">
2697                   <IntClause>20010101</IntClause>
2698                 </RationalClause>
2699               </SimpleClause>
2700             </Clause>
2701           </AuditableEventFilter>
2702         </AuditableEventQuery>
2703       </SourceAssociationBranch>
2704     </RegistryObjectQuery>
2705   </TargetAssociationBranch>
2706   <PostalAddressFilter>
2707     <Clause>
2708       <SimpleClause leftArgument = "country">
2709         <StringClause stringPredicate = "equal">France</StringClause>
2710       </SimpleClause>
2711     </Clause>
2712   </PostalAddressFilter>
2713 </OrganizationQuery>
2714 </FilterQuery>
2715 </AdhocQueryRequest>
2716

```

2717 A client application wishes to identify all organizations that have Corporation named XYZ as a
 2718 parent.

```

2719
2720 <AdhocQueryRequest>
2721   <ResponseOption returnType = "LeafClass"/>
2722   <FilterQuery>
2723     <OrganizationQuery>
2724       <OrganizationParentBranch>
2725         <NameBranch>
2726           <LocalizedStringFilter>
2727             <Clause>
2728               <SimpleClause leftArgument = "value">
2729                 <StringClause stringPredicate = "equal">XYZ</StringClause>
2730               </SimpleClause>
2731             </Clause>
2732           </LocalizedStringFilter>
2733         </NameBranch>
2734       </OrganizationParentBranch>
2735     </OrganizationQuery>
2736   </FilterQuery>
2737 </AdhocQueryRequest>
2738

```

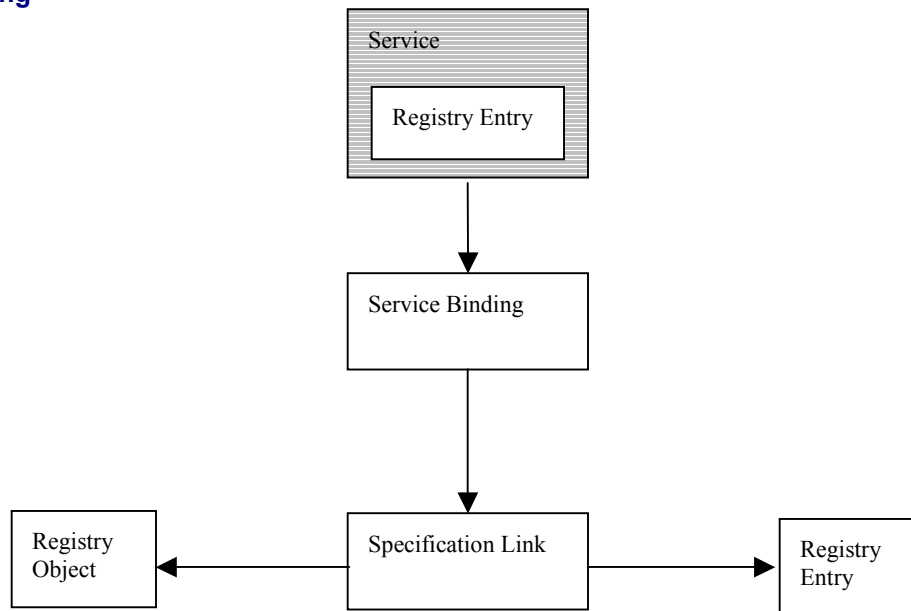
2739 8.2.10 ServiceQuery

2740 Purpose

2741

2742 To identify a set of service instances as the result of a query over selected registry metadata.

2743 **ebRIM Binding**



2744 **Figure 25:ebRIM Binding for ServiceQuery**

2745 **Definition**

```

2746 <complexType name="ServiceQueryType">
2747   <complexContent>
2748     <extension base="tns:RegistryEntryQueryType">
2749       <sequence>
2750         <element ref="tns:ServiceFilter" minOccurs="0"
2751           maxOccurs="1" />
2752         <element ref="tns:ServiceBindingBranch" minOccurs="0"
2753           maxOccurs="unbounded" />
2754       </sequence>
2755     </extension>
2756   </complexContent>
2757 </complexType>
2758 </complexType>
2759 <element name="ServiceQuery" type="tns:ServiceQueryType" />
2760
2761 <element name="ServiceQueryResult">
2762   <complexType>
2763     <choice minOccurs="0" maxOccurs="unbounded">
2764       <element ref="rim:ObjectRef" />
2765       <element ref="rim:RegistryObject" />
2766       <element ref="rim:Service" />
2767     </choice>
2768   </complexType>
2769 </element>
2770
  
```

2771 **Semantic Rules**

- 2772 1. Let S denote the set of all persistent Service instances in the Registry. The following steps
- 2773 will eliminate instances in S that do not satisfy the conditions of the specified filters.
- 2774 a) If S is empty then continue below.

- 2775 b) If a ServiceFilter is not specified then go to the next step; otherwise, let x be a service in
 2776 S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then
 2777 continue below.
- 2778 c) If a ServiceBindingBranch is not specified then continue below; otherwise, consider each
 2779 ServiceBindingBranch element separately as follows:
 2780 Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the
 2781 member of SB. If a ServiceBindingFilter element is specified within the
 2782 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If
 2783 SB is empty then remove x from S. If S is empty then continue below. If a
 2784 SpecificationLinkBranch is not specified within the ServiceBindingBranch then continue
 2785 below; otherwise, consider each SpecificationLinkBranch element separately as follows:
 2786 Let sb be a remaining service binding in SB. Let SL be the set of all specification link
 2787 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
 2788 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
 2789 remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove
 2790 x from S. If S is empty then continue below. If a RegistryObjectQuery element is
 2791 specified within the SpecificationLinkBranch then let sl be a remaining specification link
 2792 in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the
 2793 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
 2794 registry object in RO, then remove sl from SL. If SL is empty then remove sb from SB. If
 2795 SB is empty then remove x from S. If S is empty then continue below. If a
 2796 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
 2797 be a remaining specification link in SL. Treat RegistryEntryQuery element as follows:
 2798 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
 2799 a specification link for some registry entry in RE, then remove sl from SL. If SL is empty
 2800 then remove sb from SB. If SB is empty then remove x from S. If S is empty then
 2801 continue below.
- 2802 d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery
 2803 over AE as explained in section 8.2.3.
- 2804 2. If S is empty, then raise the warning: *service query result is empty*; otherwise return S as the
 2805 result of the ServiceQuery.
- 2806 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2807 ServiceQuery.

2808 Examples

2809

2810 8.2.11 Registry Filters

2811 Purpose

2812 To identify a subset of the set of all persistent instances of a given registry class.

2813 Definition

```
2814 <complexType name="FilterType">
2815   <sequence>
2816     <element ref="tns:Clause" />
2817   </sequence>
2818 </complexType>
```

```

2820 <element name="RegistryObjectFilter" type="tns:FilterType" />
2821 <element name="RegistryEntryFilter" type="tns:FilterType" />
2822 <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
2823 <element name="RegistryPackageFilter" type="tns:FilterType" />
2824 <element name="OrganizationFilter" type="tns:FilterType" />
2825 <element name="ClassificationNodeFilter" type="tns:FilterType" />
2826 <element name="AssociationFilter" type="tns:FilterType" />
2827 <element name="ClassificationFilter" type="tns:FilterType" />
2828 <element name="ClassificationSchemeFilter" type="tns:FilterType" />
2829 <element name="ExternalLinkFilter" type="tns:FilterType" />
2830 <element name="ExternalIdentifierFilter" type="tns:FilterType" />
2831 <element name="SlotFilter" type="tns:FilterType" />
2832 <element name="AuditableEventFilter" type="tns:FilterType" />
2833 <element name="UserFilter" type="tns:FilterType" />
2834 <element name="SlotValueFilter" type="tns:FilterType" />
2835 <element name="PostalAddressFilter" type="tns:FilterType" />
2836 <element name="TelephoneNumberFilter" type="tns:FilterType" />
2837 <element name="ServiceFilter" type="tns:FilterType" />
2838 <element name="ServiceBindingFilter" type="tns:FilterType" />
2839 <element name="SpecificationLinkFilter" type="tns:FilterType" />
2840 <element name="LocalizedStringFilter" type="tns:FilterType" />
2841

```

2842 Semantic Rules

- 2843 1. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
- 2844 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
2845 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in
2846 [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set
2847 of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the
2848 Clause predicate.
- 2849 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing
2850 SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in
2851 [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
2852 returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
2853 for the Clause predicate.
- 2854 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
2855 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in
2856 [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter
2857 returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to
2858 *True* for the Clause predicate.
- 2859 5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing
2860 SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in
2861 [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns
2862 a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for
2863 the Clause predicate.
- 2864 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing
2865 SimpleClause shall identify a public attribute of the Organization or PostalAddress UML
2866 classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The
2867 OrganizationFilter returns a set of identifiers for Organization instances whose attribute
2868 values evaluate to *True* for the Clause predicate.

- 2869 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
2870 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
2871 class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. The
2872 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose
2873 attribute values evaluate to *True* for the Clause predicate.
- 2874 8. For every AssociationFilter XML element, the leftArgument attribute of any containing
2875 SimpleClause shall identify a public attribute of the Association UML class defined in
2876 [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a
2877 set of identifiers for Association instances whose attribute values evaluate to *True* for the
2878 Clause predicate.
- 2879 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing
2880 SimpleClause shall identify a public attribute of the Classification UML class defined in
2881 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter
2882 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*
2883 for the Clause predicate.
- 2884 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any
2885 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
2886 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The
2887 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances
2888 whose attribute values evaluate to *True* for the Clause predicate.
- 2889 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing
2890 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in
2891 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns
2892 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the
2893 Clause predicate.
- 2894 12. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing
2895 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in
2896 [ebRIM]. If not, raise exception: *external identifier attribute error*. The
2897 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose
2898 attribute values evaluate to *True* for the Clause predicate.
- 2899 13. For every SlotFilter XML element, the leftArgument attribute of any containing
2900 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If
2901 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot
2902 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2903 14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
2904 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in
2905 [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter
2906 returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to
2907 *True* for the Clause predicate.
- 2908 15. For every UserFilter XML element, the leftArgument attribute of any containing
2909 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If
2910 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User
2911 instances whose attribute values evaluate to *True* for the Clause predicate.

- 2912 16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one
2913 SlotValue instance for each “value” in the “values” list of a Slot instance. The visible
2914 attribute of SlotValue is “value”. It is a character string. The dynamic instances of SlotValue
2915 are derived from the “values” attribute defined in ebRIM for a Slot instance. For every
2916 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause
2917 shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception:
2918 *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value”
2919 attribute evaluates to *True* for the Clause predicate.
- 2920 17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing
2921 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in
2922 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter
2923 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*
2924 for the Clause predicate.
- 2925 18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any
2926 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML
2927 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.
2928 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances
2929 whose attribute values evaluate to *True* for the Clause predicate.
- 2930 19. For every ServiceFilter XML element, the leftArgument attribute of any containing
2931 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].
2932 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for
2933 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 2934 20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
2935 SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in
2936 [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter
2937 returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to
2938 *True* for the Clause predicate.
- 2939 21. For every SpecificationLinkFilter XML element, the leftArgument attribute of any
2940 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class
2941 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The
2942 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose
2943 attribute values evaluate to *True* for the Clause predicate.
- 2944 22. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing
2945 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in
2946 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter
2947 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to
2948 *True* for the Clause predicate.

2949 **8.2.12 XML Clause Constraint Representation**

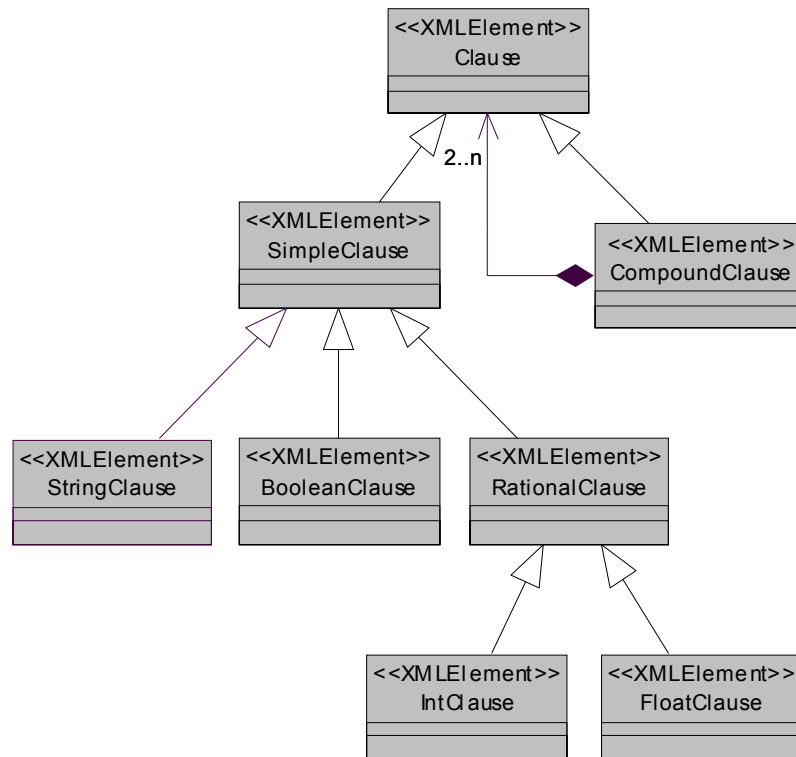
2950 **NOTE:** The Filter Query proposal makes no changes to this Section, however, it still needs to be
2951 updated to reflect XML Schema changes.

2952 **Purpose**

2953 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate Clauses*.
2954 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to
2955 simply as ***Clauses*** in this specification.

2956 **Conceptual UML Diagram**

2957 The following is a conceptual diagram outlining the Clause base structure. It is expressed in
 2958 UML for visual depiction.



2959
 2960

Figure 26: The Clause base structure

2961 **Semantic Rules**

2962 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate - RightArgument"
 2963 format to form a *Clause*. There are two types of Clauses: *SimpleClauses* and *CompoundClauses*.

2964 **SimpleClauses**

2965 A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the
 2966 *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.
 2967 SimpleClause is extended to support BooleanClause, StringClause, and RationalClause
 2968 (abstract).

2969 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
 2970 boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-
 2971 compare operations and a right argument as the element's text data. Rational number support is
 2972 provided through a common RationalClause providing an enumeration of appropriate rational
 2973 number compare operations, which is further extended to IntClause and FloatClause, each with
 2974 appropriate signatures for the right argument.

2975 **CompoundClauses**

2976 A CompoundClause contains two or more Clauses (Simple or Compound) and a connective
 2977 predicate. This provides for arbitrarily complex Clauses to be formed.

2978 **Definition**

2979
 2980
 2981

```
<!ELEMENT Clause ( SimpleClause | CompoundClause )>
```



```

2982 <!ELEMENT SimpleClause
2983   ( BooleanClause | RationalClause | StringClause )>
2984 <!ATTLIST SimpleClause
2985   leftArgument   CDATA #REQUIRED >
2986
2987 <!ELEMENT CompoundClause ( Clause, Clause+ )>
2988 <!ATTLIST CompoundClause
2989   connectivePredicate ( And | Or ) #REQUIRED>
2990
2991 <!ELEMENT BooleanClause EMPTY >
2992 <!ATTLIST BooleanClause
2993   booleanPredicate ( True | False ) #REQUIRED>
2994
2995 <!ELEMENT RationalClause ( IntClause | FloatClause )>
2996 <!ATTLIST RationalClause
2997   logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
2998
2999 <!ELEMENT IntClause ( #PCDATA )
3000 <!ATTLIST IntClause
3001   e-dtype NMTOKEN #FIXED 'int' >
3002
3003 <!ELEMENT FloatClause ( #PCDATA )>
3004 <!ATTLIST FloatClause
3005   e-dtype NMTOKEN #FIXED 'float' >
3006
3007 <!ELEMENT StringClause ( #PCDATA )>
3008 <!ATTLIST StringClause
3009   stringPredicate
3010     ( contains | -contains |
3011       startswith | -startswith |
3012       equal | -equal
3013       endswith | -endswith ) #REQUIRED >
3014

```

3015 Examples

3016 Simple BooleanClause: "Smoker" = True

```

3017
3018 <?xml version="1.0" encoding="UTF-8"?>
3019 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3020 <Clause>
3021   <SimpleClause leftArgument="Smoker">
3022     <BooleanClause booleanPredicate="True"/>
3023   </SimpleClause>
3024 </Clause>
3025

```

3026 Simple StringClause: "Smoker" contains "mo"

```

3027
3028 <?xml version="1.0" encoding="UTF-8"?>
3029 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3030 <Clause>
3031   <SimpleClause leftArgument="Smoker">
3032     <StringClause stringcomparepredicate="contains">
3033       mo
3034     </StringClause>
3035   </SimpleClause>
3036 </Clause>
3037

```

3038 Simple IntClause: "Age" >= 7

```

3039
3040 <?xml version="1.0" encoding="UTF-8"?>
3041 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3042 <Clause>
3043   <SimpleClause leftArgument="Age">
3044     <RationalClause logicalPredicate="GE">
3045       <IntClause e-dtype="int">7</IntClause>
3046     </RationalClause>
3047   </SimpleClause>
3048 </Clause>
3049

```

3050 Simple FloatClause: "Size" = 4.3

```

3051
3052 <?xml version="1.0" encoding="UTF-8"?>
3053 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3054 <Clause>
3055   <SimpleClause leftArgument="Size">
3056     <RationalClause logicalPredicate="E">
3057       <FloatClause e-dtype="float">4.3</FloatClause>
3058     </RationalClause>
3059   </SimpleClause>
3060 </Clause>
3061

```

3062 Compound with two Simples (("Smoker" = False)AND("Age" =< 45))

```

3063
3064 <?xml version="1.0" encoding="UTF-8"?>
3065 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3066 <Clause>
3067   <CompoundClause connectivePredicate="And">
3068     <Clause>
3069       <SimpleClause leftArgument="Smoker">
3070         <BooleanClause booleanPredicate="False"/>
3071       </SimpleClause>
3072     </Clause>
3073     <Clause>
3074       <SimpleClause leftArgument="Age">
3075         <RationalClause logicalPredicate="EL">
3076           <IntClause e-dtype="int">45</IntClause>
3077         </RationalClause>
3078       </SimpleClause>
3079     </Clause>
3080   </CompoundClause>
3081 </Clause>
3082

```

3083 Compound with one Simple and one Compound

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

```

3084
3085 <?xml version="1.0" encoding="UTF-8"?>
3086 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
3087 <Clause>
3088   <CompoundClause connectivePredicate="And">
3089     <Clause>
3090       <SimpleClause leftArgument="Smoker">
3091         <BooleanClause booleanPredicate="False"/>
3092       </SimpleClause>
3093

```

```

3094     </Clause>
3095     <Clause>
3096         <CompoundClause connectivePredicate="Or">
3097             <Clause>
3098                 <SimpleClause leftArgument="Age">
3099                     <RationalClause logicalPredicate="EL">
3100                         <IntClause e-dtype="int">45</IntClause>
3101                     </RationalClause>
3102                 </SimpleClause>
3103             </Clause>
3104             <Clause>
3105                 <SimpleClause leftArgument="American">
3106                     <BooleanClause booleanPredicate="True"/>
3107                 </SimpleClause>
3108             </Clause>
3109         </CompoundClause>
3110     </Clause>
3111 </CompoundClause>
3112 </Clause>
3113

```

3114 **8.3 SQL Query Support**

3115 The Registry may optionally support an SQL based query capability that is designed for Registry
 3116 clients that demand more advanced query capability. The optional SQLQuery element in the
 3117 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query
 3118 language.

3119 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of
 3120 the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database
 3121 Language SQL [SQL], extended to include <sql invoked routines> (also known as
 3122 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined
 3123 in template form in Appendix D.3. The syntax of the Registry query language is defined by the
 3124 BNF grammar in D.1.

3125 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use
 3126 relational databases in a Registry implementation.

3127 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

3128 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational
 3129 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as
 3130 described in the following sections.

3131 **8.3.1.1 Class Binding**

3132 A subset of the class names defined in [ebRIM] map to table names that may be queried by an
 3133 SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an
 3134 SQL query.

3135 The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3
 3136 is as follows:

- 3137 • Classes that have concrete instances are mapped to relational tables. In addition entity classes
 3138 (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- 3139 • The intermediate classes in the inheritance hierarchy, namely RegistryObject and
 3140 RegistryEntry, map to relational views.

- 3141 • The names of relational tables and views are the same as the corresponding [ebRIM] class
3142 name. However, the name binding is case insensitive.
- 3143 • Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in
3144 Appendix D.3 where the column definitions are based on a subset of attributes defined for
3145 that class in [ebRIM]. The attributes that map to columns include the inherited attributes for
3146 the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed
3147 which column definitions.

3148 An SQLQuery against a table not defined in Appendix D.3 may raise an error condition:
3149 InvalidQueryException.

3150 The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn
3151 definitions.

3152 **8.3.1.2 Primitive Attributes Binding**

3153 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same
3154 way as column names in SQL. Again the exact attribute names are defined in the class
3155 definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is
3156 therefore valid for a query to contain attribute names that do not exactly match the case defined
3157 in [ebRIM].

3158 **8.3.1.3 Reference Attribute Binding**

3159 A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a
3160 class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject
3161 class returns a reference to an instance of an AccessControlPolicy object.

3162 In such cases the reference maps to the id attribute for the referenced object. The name of the
3163 resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data
3164 type for the column is VARCHAR(64) as defined in Appendix D.3.

3165 When a reference attribute value holds a null reference, it maps to a null value in the SQL
3166 binding and may be tested with the <null specification> (“IS [NOT] NULL” syntax) as defined
3167 by [SQL].

3168 Reference attribute binding is a special case of a primitive attribute mapping.

3169 **8.3.1.4 Complex Attribute Binding**

3170 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of
3171 a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type
3172 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

3173 The SQL query schema does not map complex attributes as columns in the table for the class for
3174 which the attribute is defined. Instead the complex attributes are mapped to columns in the table
3175 for the domain class that represents the data type for the complex attribute (e.g.

3176 TelephoneNumber). A column links the row in the domain table to the row in the parent table
3177 (e.g. User). An additional column named ‘attribute_name’ identifies the attribute name in the
3178 parent class, in case there are multiple attributes with the same complex attribute type.

3179 This mapping also easily allows for attributes that are a collection of a complex type. For
3180 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the
3181 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier
3182 and an attribute_name.

3183 **8.3.1.5 Binding of Methods Returning Collections**

3184 Several of the [ebRIM] classes define methods in addition to attributes, where these methods

3185 return collections of references to instances of classes defined by [ebRIM]. For example, the
3186 getPackages method of the ManagedObject class returns a Collection of references to instances
3187 of Packages that the object is a member of.
3188 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in
3189 Appendix D.3 such that these stored procedures return a collection of id attribute values. The
3190 returned value of these stored procedures can be treated as the result of a table sub-query in SQL.
3191 These stored procedures may be used as the right-hand-side of an SQL IN clause to test for
3192 membership of an object in such collections of references.

3193 **8.3.2 Semantic Constraints On Query Syntax**

3194 This section defines simplifying constraints on the query syntax that cannot be expressed in the
3195 BNF for the query syntax. These constraints must be applied in the semantic analysis of the
3196 query.

- 3197 1. Class names and attribute names must be processed in a case insensitive manner.
- 3198 2. The syntax used for stored procedure invocation must be consistent with the syntax of an
3199 SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 3200 3. For this version of the specification, the SQL select column list consists of exactly one
3201 column, and must always be t.id, where t is a table reference in the FROM clause.
- 3202 4. Join operations must be restricted to simple joins involving only those columns that have an
3203 index defined within the normative SQL schema. This constraint is to prevent queries that
3204 may be computationally too expensive.

3205 **8.3.3 SQL Query Results**

3206 The result of an SQL query resolves to a collection of objects within the registry. It never
3207 resolves to partial attributes. The objects related to the result set may be returned as an
3208 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the
3209 responseOption parameter specified by the client on the AdHocQueryRequest. The entire result
3210 set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 1.1.

3211 **8.3.4 Simple Metadata Based Queries**

3212 The simplest form of an SQL query is based upon metadata attributes specified for a single class
3213 within [ebRIM]. This section gives some examples of simple metadata based queries.

3214 For example, to get the collection of ExtrinsicObjects whose name contains the word 'Acme'
3215 and that have a version greater than 1.3, the following query must be submitted:

```
3216 SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND  
3217 eo.id = nm.parent AND  
3218 eo.majorVersion >= 1 AND  
3219 (eo.majorVersion >= 2 OR eo.minorVersion > 3);  
3220  
3221
```

3222 Note that the query syntax allows for conjugation of simpler predicates into more complex
3223 queries as shown in the simple example above.

3224 **8.3.5 RegistryObject Queries**

3225 The schema for the SQL query defines a special view called RegistryObject that allows doing a
3226 polymorphic query against all RegistryObject instances regardless of their actual concrete type or
3227 table name.

3228 The following example is the similar to that in Section 8.3.4 except that it is applied against all
 3229 RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id
 3230 for all qualifying RegistryObject instances whose name contains the word ‘Acme’ and whose
 3231 description contains the word “bicycle”.

```
3232
3233 SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND
3234 d.value LIKE '%bicycle%' AND
3235 ro.id = nm.parent AND ro.id = d.parent;
3236
```

3237 **8.3.6 RegistryEntry Queries**

3238 The schema for the SQL query defines a special view called RegistryEntry that allows doing a
 3239 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or
 3240 table name.

3241 The following example is the same as Section 8.3.4 except that it is applied against all
 3242 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id
 3243 for all qualifying RegistryEntry instances whose name contains the word ‘Acme’ and that have a
 3244 version greater than 1.3.

```
3245
3246 SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
3247 re.id = nm.parent AND
3248 re.majorVersion >= 1 AND
3249 (re.majorVersion >= 2 OR re.minorVersion > 3);
3250
```

3251 **8.3.7 Classification Queries**

3252 This section describes the various classification related queries that must be supported.

3253 **8.3.7.1 Identifying ClassificationNodes**

3254 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may
 3255 also be identified as a path attribute that specifies an XPATH expression [XPT] from a root
 3256 classification node to the specified classification node in the XML document that would
 3257 represent the ClassificationNode tree including the said ClassificationNode.

3258 **8.3.7.2 Getting ClassificationSchemes**

3259 To get the collection of ClassificationSchemes the following query predicate must be supported:

```
3260
3261 SELECT scheme.id FROM ClassificationScheme scheme;
3262
```

3263 The above query returns all ClassificationSchemes. Note that the above query may also specify
 3264 additional predicates (e.g. name, description etc.) if desired.

3265 **8.3.7.3 Getting Children of Specified ClassificationNode**

3266 To get the children of a ClassificationNode given the ID of that node the following style of query
 3267 must be supported:

```
3268
3269 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
3270
```

3271 The above query returns all ClassificationNodes that have the node specified by <id> as their
 3272 parent attribute.

3273 **8.3.7.4 Getting Objects Classified By a ClassificationNode**

3274 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the
 3275 following style of query must be supported:

```

3276
3277 SELECT id FROM ExtrinsicObject
3278 WHERE
3279     id IN (SELECT classifiedObject FROM Classification
3280           WHERE
3281             classificationNode IN (SELECT id FROM ClassificationNode
3282                                   WHERE path = '/Geography/Asia/Japan'))
3283 AND
3284     id IN (SELECT classifiedObject FROM Classification
3285           WHERE
3286             classificationNode IN (SELECT id FROM ClassificationNode
3287                                   WHERE path = '/Industry/Automotive'))
3288

```

3289 The above query gets the collection of ExtrinsicObjects that are classified by the Automotive
3290 Industry and the Japan Geography. Note that according to the semantics defined for
3291 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by
3292 descendents of the specified ClassificationNodes.

3293 **8.3.7.5 Getting Classifications That Classify an Object**

3294 To get the collection of Classifications that classify a specified Object the following style of
3295 query must be supported:

```

3296
3297 SELECT id FROM Classification c
3298     WHERE c.classifiedObject = <id>;
3299

```

3300 **8.3.8 Association Queries**

3301 This section describes the various Association related queries that must be supported.

3302 **8.3.8.1 Getting All Association With Specified Object As Its Source**

3303 To get the collection of Associations that have the specified Object as its source, the following
3304 query must be supported:

```

3305
3306 SELECT id FROM Association WHERE sourceObject = <id>
3307

```

3308 **8.3.8.2 Getting All Association With Specified Object As Its Target**

3309 To get the collection of Associations that have the specified Object as its target, the following
3310 query must be supported:

```

3311
3312 SELECT id FROM Association WHERE targetObject = <id>
3313

```

3314 **8.3.8.3 Getting Associated Objects Based On Association Attributes**

3315 To get the collection of Associations that have specified Association attributes, the following
3316 queries must be supported:

3317 Select Associations that have the specified name.

```

3318
3319 SELECT id FROM Association WHERE name = <name>
3320

```

3321 Select Associations that have the specified association type, where association type is a string
3322 containing the corresponding field name described in [ebRIM].

```

3323
3324 SELECT id FROM Association WHERE
3325     associationType = <associationType>
3326

```

3327 **8.3.8.4 Complex Association Queries**

3328 The various forms of Association queries may be combined into complex predicates. The

3329 following query selects Associations that have a specific sourceObject, targetObject and
 3330 associationType:

```
3331
3332 SELECT id FROM Association WHERE
3333     sourceObject = <id1> AND
3334     targetObject = <id2> AND
3335     associationType = <associationType>;
3336
```

3337 8.3.9 Package Queries

3338 To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
3339
3340 SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));
3341
```

3342 8.3.9.1 Complex Package Queries

3343 The following query gets all Packages that a specified object belongs to, that are not deprecated
 3344 and where name contains "RosettaNet."

```
3345
3346 SELECT id FROM Package p, Name n WHERE
3347     p.id IN (RegistryObject_packages(<id>)) AND
3348     nm.value LIKE '%RosettaNet%' AND nm.parent = p.id AND
3349     p.status <> 'Deprecated'
3350
```

3351 8.3.10 ExternalLink Queries

3352 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is
 3353 specified:

```
3354
3355 SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
3356
```

3357 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is
 3358 specified:

```
3359
3360 SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
3361
```

3362 8.3.10.1 Complex ExternalLink Queries

3363 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that
 3364 contain the word 'legal' in their description and have a URL for their externalURI.

```
3365
3366 SELECT id FROM ExternalLink WHERE
3367     id IN (RegistryObject_externalLinks(<id>)) AND
3368     description LIKE '%legal%' AND
3369     externalURI LIKE '%http://%'
3370
```

3371 8.3.11 Audit Trail Queries

3372 To get the complete collection of AuditableEvent objects for a specified ManagedObject, the
 3373 following query is specified:

```
3374
3375 SELECT id FROM AuditableEvent WHERE registryObject = <id>
3376
```

3377 8.4 Content Retrieval

3378 A client retrieves content via the Registry by sending the GetContentRequest to the
 3379 ObjectQueryManager. The GetContentRequest specifies a list of Object references for Objects

3380 that need to be retrieved. The ObjectQueryManager returns the specified content by sending a
 3381 GetContentResponse message to the ObjectQueryManagerClient class of the client. If there are
 3382 no errors encountered, the GetContentResponse message includes the specified content as
 3383 additional payloads within the message. In addition to the GetContentResponse payload, there is
 3384 one additional payload for each content that was requested. If there are errors encountered, the
 3385 RegistryResponse payload includes an error and there are no additional content specific
 3386 payloads.

3387 **8.4.1 Identification Of Content Payloads**

3388 Since the GetContentResponse message may include several repository items as additional
 3389 payloads, it is necessary to have a way to identify each payload in the message. To facilitate this
 3390 identification, the Registry must do the following:

- 3391 • Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mime-
 3392 part that contains the corresponding repository item for the ExtrinsicObject
- 3393 • In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the
 3394 repository item in the Reference element for that object in the Manifest element of the
 3395 ebXMLHeader.

3396 **8.4.2 GetContentResponse Message Structure**

3397 The following message fragment illustrates the structure of the GetContentResponse Message
 3398 that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs
 3399 for the requested objects.

```

3400 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
3401
3402 --Boundary
3403 Content-ID: <GetContentRequest@example.com>
3404 Content-Type: text/xml
3405
3406 <?xml version="1.0" encoding="UTF-8"?>
3407 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
3408   xmlns:eb='http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
3409 <SOAP-ENV:Header>
3410
3411   ...ebMS header goes here if using ebMS
3412
3413 </SOAP-ENV:Header>
3414 <SOAP-ENV:Body>
3415
3416   ...ebMS manifest goes here if using ebMS
3417
3418 <?xml version="1.0" encoding="UTF-8"?>
3419
3420 <GetContentRequest>
3421   <ObjectRefList>
3422     <ObjectRef id="d8163dfb-f45a-4798-81d9-88aca29c24ff" .../>
3423     <ObjectRef id="212c3a78-1368-45d7-acc9-a935197e1e4f" .../>
3424   </ObjectRefList>
3425 </GetContentRequest>
3426
3427 </SOAP-ENV:Body>
3428 </SOAP-ENV:Envelope>
3429
3430 --Boundary
3431 Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
3432 Content-Type: text/xml
3433
3434 <?xml version="1.0" encoding="UTF-8"?>
3435 <CPP>
3436   ....
3437 </CPP>
  
```

3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450

```
--Boundary--  
Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f  
Content-Type: text/xml  
  
<CPP>  
.....  
</CPP>  
  
--Boundary--
```

3451 **9 Registry Security**

3452 This chapter describes the security features of the ebXML Registry. It is assumed that the reader
3453 is familiar with the security related classes in the Registry information model as described in
3454 [ebRIM]. Security glossary terms can be referenced from RFC 2828.

3455 **9.1 Security Concerns**

3456 The security risks broadly stem from the following concerns. After a description of these
3457 concerns and potential solutions, we identify the concerns that we address in the current
3458 specification

- 3459 1. Is the content of the registry (data) trustworthy?
 - 3460 a) How to make sure “what is in the registry” is “what is put there” by a submitting
3461 organization? This concern can be addressed by ensuring that the publisher is
3462 authenticated using digital signature (Source Integrity), message is not corrupted during
3463 transfer using digital signature (Data Integrity), and the data is not altered by
3464 unauthorized subjects based on access control policy (Authorization)
 - 3465 b) How to protect data while in transmission?
3466 Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data
3467 Confidentiality that can be addressed by encrypting the data in transmission. How to
3468 protect against a replay attack.
 - 3469 c) Is the content up to date? The versioning as well as any time stamp processing, when
3470 done securely will ensure the “latest content” is guaranteed to be the latest content.
3471 Authorization with access control policy could solve this problem.
 - 3472 d) How to ensure only bona fide responsible organizations add contents to registry?
3473 Ensuring Source Integrity (as in 1a).
 - 3474 e) How to ensure that bona fide publishers add contents to registry only at authorized
3475 locations? (System Integrity)
 - 3476 f) What if the publishers deny modifying certain content after-the-fact? To prevent this
3477 (Nonrepudiation) audit trails may be kept which contain signed message digests.
 - 3478 g) What if the reader denies getting information from the registry?
- 3479 2. How to provide selective access to registry content? The broad answer is, by using an access
3480 control policy – applies to (a), (b), and (c) directly.
 - 3481 a) How does a submitting organization restrict access to the content to only specific registry
3482 readers?
 - 3483 b) How can a submitting organization allow some “partners” (fellow publishers) to modify
3484 content?
 - 3485 c) How to provide selective access to partners the registry usage data?
 - 3486 d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw
3487 failure of the registry security components? The solution to this problem is by having
3488 System Integrity.
 - 3489 e) Data confidentiality of RegistryObject

3490 3. How do we make “who can see what” policy itself visible to limited parties, even excluding
3491 the administrator (self & confidential maintenance of access control policy). By making sure
3492 there is an access control policy for accessing the policies themselves.

3493 4. How to transfer credentials? The broad solution is to use credentials assertion (such as being
3494 worked on in SAML). Currently, Registry does not support the notion of a session.

3495 Therefore, some of these concerns are not relevant to the current specification.

3496 a) How to transfer credentials (authorization/authentication) to federated registries?

3497 b) How do aggregators get credentials (authorization/authentication) transferred to them?

3498 c) How to store credentials through a session?

3499 In the current version of this specification, we address concern 1(d) above. We have used a
3500 minimalist approach with respect to the access control concern in item 2 above. Essentially, “any
3501 *known* entity (Submitting Organization) can publish content and *anyone* can view published
3502 content.” The Registry information model has been designed to allow more sophisticated
3503 security policies in future versions of this specification.

3504 9.2 Integrity of Registry Content

3505 It is assumed that most business registries do not have the resources to validate the veracity of
3506 the content submitted to them. The minimal integrity that the Registry must provide is to ensure
3507 that content submitted by a Submitting Organization (SO) is maintained in the Registry without
3508 any tampering *within* the Registry. Furthermore, the Registry must make it possible to identify
3509 the SO for any Registry content unambiguously.

3510 9.2.1 Message Payload Signature

3511 Integrity of Registry content requires that all submitted content be signed by the Registry client.
3512 The signature on the submitted content ensures that:

- 3513 • The content has not been tampered within the Registry.
- 3514 • The content’s veracity can be ascertained by its association with a specific submitting
3515 organization

3516 This section specifies the requirements for generation, packaging and validation of payload
3517 signatures. A payload signature is packaged with the payload. Therefore the requirements apply
3518 regardless of whether the Registry Client and the Registry Operator communicate over vanilla
3519 SOAP with Attachments.

3520 ebXML Messaging Service ([ebMS]) does not specify the generation, validation and packaging
3521 of payload signatures. The specification of payload signatures is left upto the application (such as
3522 [ebRS]). So the requirements on the payload signatures augment the [ebMS] specification.

3523 Use Case

3524 This Use Case illustrates the use of header and payload signatures.

- 3525 • RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the
3526 content along with the payload signature to the Registry.
- 3527 • RC2 (Registry Client 2) retrieves RC1’s content from the Registry.
- 3528 • RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
3529 the content, the response from the Registry Operator to RC2 contains the following:
 - 3530 – Payload containing the content that has been published by RC1.
 - 3531 – RC1’s payload signature (represented by a ds:Signature element) over RC1’s published
3532 content.

- 3533 – Either the key for validating RC1’s payload signature in ds:Signature element (using the
3534 KeyInfo element as specified in [XMLDSIG]) or RC1’s identity so RC2 can obtain the
3535 validation key for signature (e.g. retrieve a certificate containing the public key for RC1).
3536 – A ds:Signature element containing the header signature. Note that the Registry Operator
3537 not RC1 generates this signature.

3538 9.2.2 Payload Signature Requirements

3539 9.2.2.1 Payload Signature Packaging Requirements

3540 A payload signature is represented by a ds:Signature element. The payload signature must be
3541 packaged with the payload as specified here. This packaging assumes that the payload is always
3542 signed.

- 3543 • The payload and its signature must be enclosed in a MIME multipart message with a
3544 Content-Type of multipart/Related.
- 3545 • The first body part must contain the XML signature as specified in the section “Payload
3546 Signature Generation Requirements”.
- 3547 • The second body part must be the content

3548 The packaging of the payload signature is as follows:

```
3549
3550 MIME-Version: 1.0
3551 Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
3552 Content-Description: ebXML Message
3553
3554 -- MIME_boundary
3555 Content-Type: text/xml; charset=UTF-8
3556 Content-Transfer-Encoding: 8bit
3557 Content-ID: http://claiming-it.com/claim061400a.xml
3558
3559 <?xml version='1.0' encoding="utf-8"?>
3560 <SOAP-ENV: Envelope>
3561   ...
3562   SOAP-ENV: Envelope>
3563
3564 --MIME_boundary
3565 Content-Type: multipart/Related; boundary=PAYLOAD_boundary
3566
3567 --PAYLOAD_boundary
3568 Content-Type: text/xml; charset=UTF-8
3569 Content-Transfer-Encoding: 8bit
3570 Content-ID: payload1
3571 <ds:Signature>
3572   ... Payload signature
3573 </ds: Signature>
3574
3575 --PAYLOAD_boundary
3576 Content-Type: text/xml; charset=UTF-8
3577 Content-Transfer-Encoding: 8bit
3578 Content-ID: payload2
3579 <SubmitObjectsRequest>...</SubmitObjectsRequest>
3580 --MIME_boundary
3581
```

3582 9.2.2.2 Payload Signature Generation Requirements

3583 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in
3584 this section. Note: the “ds” name space reference is to <http://www.w3.org/2000/09/xmlsig#>

3585 • ds:SignatureMethod must be present. For same reasons as noted in Section “Message Header
3586 Requirements”, the client must sign using the following Algorithm attribute:
3587 <http://www.w3.org/2000/09/xmlsig/#dsa-sha1>

3588 • The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. . The
3589 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
3590 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>

3591 • One ds:Reference element to reference the payload that needs to be signed must be created..
3592 The ds:Reference element:

3593 – Must identify the payload to be signed using the URI attribute of the ds:Reference
3594 element.

3595 – Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be
3596 support the following digest algorithm:

3597 <http://www.w3.org/2000/09/xmlsig/#sha1>

3598 – Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

3599 The ds:SignedValue must be generated as specified in [XMLDSIG].

3600 The ds:KeyInfo element may be present But when present, the ds:KeyInfo field is subject to the
3601 requirements stated in the “KeyDistribution and KeyInfo element” section of this document.

3602 9.2.2.3 Message Payload Signature Validation

3603 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

3604 9.2.2.4 Payload Signature Example

3605 The following example shows the format of the payload signature:

```
3606 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
3607 <ds:SignedInfo>
3608     <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmlsig/#dsa-sha1/>
3609     <ds:CanonicalizationMethod>
3610         Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
3611     </ds:CanonicalizationMethod>
3612     <ds:Reference URI=#Payload1>
3613         <ds:DigestMethod DigestAlgorithm="http://www.w3.org/2000/09/xmlsig/#sha1">
3614         <ds:DigestValue> ... </ds:DigestValue>
3615     </ds:Reference>
3616 </ds:SignedInfo>
3617 <ds:SignatureValue> ... </ds:SignatureValue>
3618 </ds:Signature>
```

3621 9.3 Authentication

3622 The Registry must be able to authenticate the identity of the Principal associated with client
3623 requests. *Authentication* is required to identify the ownership of content as well as to identify
3624 what “privileges” a Principal can be assigned with respect to the specific objects in the Registry.

3625 The Registry must perform Authentication on a per request basis. From a security point of view,
3626 all messages are independent and there is no concept of a session encompassing multiple
3627 messages or conversations. Session support may be added as an optimization feature in future

3628 versions of this specification.

3629 It is important to note that the registry can only guarantee data integrity and it may be used for
3630 Authentication knowing that it is vulnerable to replay types of attacks. The mechanism we
3631 describe in this section would fail to correctly authenticate a Submitting Organization in case of
3632 replay attacks. True support for authentication requires timestamps or nonce that are signed

3633 **9.3.1 Message Header Signature**

3634 Message headers are signed to provide data integrity while the message is in transit. Note that the
3635 signature within the message header also signs the digests of the payloads.

3636 **Header Signature Requirements**

3637 Message headers can be signed and are referred to as a header signature. This section specifies
3638 the requirements for generation, packaging and validation of a header signature. These
3639 requirements apply when the Registry Client and Registry Operator communicate using vanilla
3640 SOAP with Attachments. When ebXML MS is used for communication, then the [ebMS]
3641 specifies the generation, packaging and validation of XML signatures in the SOAP header.
3642 Therefore the header signature requirements do not apply when the ebXML MS is used for
3643 communication. However, payload signature generation requirements (specified elsewhere in
3644 this document) do apply whether vanilla SOAP with Attachments or ebXML MS is used for
3645 communication.

3646 **9.3.1.1 Packaging Requirements**

3647 A header signature is represented by a ds:Signature element. The ds:Signature element generated
3648 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature
3649 element in the SOAP header field is shown below.

```
3650
3651 MIME-Version: 1.0
3652 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3653 Content-Description: ebXML Message
3654
3655 -- MIME_boundary
3656 Content-Type: text/xml; charset=UTF-8
3657 Content-Transfer-Encoding: 8bit
3658 Content-ID: http://claiming-it.com/claim061400a.xml
3659
3660 <?xml version='1.0' encoding="utf-8"?>
3661 <SOAP-ENV:Envelope
3662   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3663   <SOAP-ENV:Header>
3664     <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3665       ...signature over soap envelope
3666     </ds:Signature>
3667   </SOAP-ENV:Header>
3668   <SOAP-ENV:Body>
3669     ...
3670   </SOAP-ENV:Body>
3671 </SOAP-ENV:Envelope>
3672
```

3673 **9.3.1.2 Header Signature Generation Requirements**

3674 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in
3675 this section. A ds:Signature element contains:

- 3676 • ds:SignedInfo
- 3677 • ds:SignatureValue
- 3678 • ds:KeyInfo

3679 The ds:SignedInfo element must be generated as follows:

- 3680 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
3681 using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,
3682 a client must be capable of signing using only the following Algorithm attribute:
3683 <http://www.w3.org/2000/09/xmlsig/#dsa-sha1> This algorithm is being chosen because all
3684 XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
- 3685 2. The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The
3686 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
3687 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
- 3688 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.
3689 This signs the entire ds:Reference element and:
 - 3690 – Must include the following ds:Transform:
3691 <http://www.w3.org/2000/09/xmlsig#enveloped-signature>
3692 This ensures that the signature (which is embedded in the <SOAP-ENV:Header>
3693 element) is not included in the signature calculation.
 - 3694 – Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the
3695 ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) .
3696 The URI attribute must be “”.
 - 3697 – Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support
3698 the following digest algorithm: <http://www.w3.org/2000/09/xmlsig/#sha1>
 - 3699 – Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

3700 The ds:SignatureValue must be generated as specified in [XMLDSIG].

3701 The ds:KeyInfo element may be present But when present, it is subject to the requirements stated
3702 in the “KeyDistribution and KeyInfo element” section of this document.

3703 9.3.1.3 Header Signature Validation Requirements

3704 The ds:Signature element for the ebXML message header must be validated by the recipient as
3705 specified by [XMLDSIG].

3706 9.3.1.4 Header Signature Example

3707 The following example shows the format of a header signature:

```
3708
3709 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
3710   <ds:SignedInfo>
3711     <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmlsig/#dsa-sha1/>
3712     <ds:CanonicalizationMethod>
3713       Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
3714     </ds:CanonicalizationMethod>
3715     <ds:Reference URI= "">
3716       <ds:Transform>
3717         http://www.w3.org/2000/09/xmlsig#enveloped-signature
3718       </ds:Transform>
3719       <ds:DigestMethod DigestAlgorithm="./xmlsig#sha1">
3720     <ds:DigestValue> ... </ds:DigestValue>
```



```

3721     </ds:Reference>
3722   </ds:SignedInfo>
3723   <ds:SignatureValue> ... </ds:SignatureValue>
3724 </ds:Signature>
3725

```

3726 9.4 Key Distribution and KeyInfo Element

3727 To validate a signature, the recipient of the signature needs the validation key corresponding to
 3728 the signer's key. The following use cases need to be handled:

- 3729 • Registry Operator needs the validation key of the Registry Client to validate the signature
- 3730 • Registry Client needs the validation key of the Registry Operator to validate the Registry's
 3731 signature.
- 3732 • Registry Client RC1 needs the validation key of Registry Client (RC2) to validate the content
 3733 signed by RC1.

3734 [XMLDSIG] provides a *ds:KeyInfo* element that can be used to pass the recipient information
 3735 for retrieving the validation key. *ds:KeyInfo* is an optional element as specified in [XMLDSIG].
 3736 This field together with the procedures outlined in this section is used to securely pass the
 3737 validation key to a recipient. *ds:KeyInfo* can be used to pass information such as keys,
 3738 certificates, names etc. The intended usage of KeyInfo field for ebXML is as follows:

- 3739 • Pass a X509 Certificate. This recipient extracts the X509 Certificate and the public key from
 3740 the certificate.

3741 The following assumptions are also made:

- 3742 1. A Certificate is associated both with the Registry Operator and a Registry Client.
- 3743 2. A Registry Client registers its certificate with the Registry Operator. The mechanism used for
 3744 this is not specified here.
- 3745 3. A Registry Client obtains the Registry Operator's certificate and stores it in its own local key
 3746 store. The mechanism is not specified here.

3747 9.4.1 Using ds:KeyInfo Field

3748 Two typical usage scenarios are described below, and the contents of the *ds:KeyInfo* field based
 3749 on these use cases are deduced from these scenarios.

3750 Scenario 1

- 3751 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
- 3752 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
- 3753 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
- 3754 4. Registry Operator retrieves the certificate from the KeyInfo field in the header signature It
 3755 establishes trust in the certificate by comparing it with the certificate that RC must already
 3756 have registered.
- 3757 5. Registry Operator validates the header signature using the public key from the certificate.
- 3758 6. Registry Operator validates the payload signature by repeating steps 4 and 5 using the
 3759 certificate from the KeyInfo field of the payload signature. This is only required if the
 3760 Registry Operator wants to ensure that contents have not been modified on the network.

3761 Scenario 2

- 3762 1. RC1 signs the payload and SOAP envelope using its private key and publishes to the
3763 Registry.
 - 3764 2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
 - 3765 3. The certificate of RC2 is passed to the Registry in the KeyInfo field of the payload signature.
 - 3766 4. RC2 retrieves content from the Registry.
 - 3767 5. Registry Operator signs the SOAP envelope using its private key. Registry Operator sends
3768 RC1's content and the RC1's signature (signed by RC1).
 - 3769 6. Registry Operator need not send its certificate in the KeyInfo field since RC2 is assumed to
3770 have obtained the Registry Operator's certificate out of band and installed it in its local key
3771 store.
 - 3772 7. RC2 obtains Registry Operator's certificate out of its local key store and verifies the Registry
3773 Operator's signature.
 - 3774 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates
3775 the signature on the payload.
- 3776 **Use Case Summary**
- 3777 Based on the above use cases, a Registry Client and Registry must support the following:
- 3778 • X509Certificate element.
 - 3779 – This is a child element of X509Data, which in turn is a child element of KeyInfo. This
3780 can be used to pass the certificate to the recipient. X509Certificate element contains a
3781 base64-encoded certificate.

3782 **9.5 Confidentiality**

3783 **9.5.1 On-the-wire Message Confidentiality**

3784 It is suggested but not required that message payloads exchanged between clients and the
3785 Registry be encrypted during transmission. Payload encryption must abide by any restrictions set
3786 forth in [SEC].

3787 **9.5.2 Confidentiality of Registry Content**

3788 In the current version of this specification, there are no provisions for confidentiality of Registry
3789 content. All content submitted to the Registry may be discovered and read by *any* client.
3790 Therefore, the Registry must be able to decrypt any submitted content after it has been received
3791 and prior to storing it in its repository. This implies that the Registry and the client have an a
3792 priori agreement regarding encryption algorithm, key exchange agreements, etc. This service is
3793 not addressed in this specification.

3794 **9.6 Authorization**

3795 The Registry must provide an authorization mechanism based on the information model defined
3796 in [ebRIM]. In this version of the specification the authorization mechanism is based on a default
3797 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of
3798 this specification will allow for custom Access Control Policies to be defined by the Submitting
3799 Organization. The authorization is going to be applied on a specific set of privileges. A
3800 privilege is the ability to carry a specific action.

3801 **9.6.1 Actions**

3802 Life Cycle Actions

3803 submitObjects
 3804 updateObjects
 3805 addSlots
 3806 removeSlots
 3807 approveObjects
 3808 deprecateObjects
 3809 removeObjects

3810 Read Actions

3811 The various getXXX() methods in QueryManagement Service.

3812 **9.7 Access Control**

3813 The Registry must create a default AccessControlPolicy object that grants the default
 3814 permissions to Registry users based upon their assigned role. The following table defines the
 3815 Permissions granted by the Registry to the various pre-defined roles for Registry users based
 3816 upon the default AccessControlPolicy.

Table 11: Default Access Control Policies

Role	Permissions
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).

3818 The following list summarizes the default role-based AccessControlPolicy:

- 3819 • The Registry must implement the default AccessControlPolicy and associate it with all
- 3820 Objects in the Registry
- 3821 • Anyone can publish content, but needs to be authenticated
- 3822 • Anyone can access the content without requiring authentication
- 3823 • The ContentOwner has access to all methods for Registry Objects owned by them
- 3824 • The RegistryAdministrator has access to all methods on all Registry Objects
- 3825 • Unauthenticated clients can access all read-only (getXXX) methods
- 3826 • At the time of content submission, the Registry must assign the default ContentOwner role to
- 3827 the Submitting Organization (SO) as authenticated by the credentials in the submission
- 3828 message. In the current version of this specification, the Submitting Organization will be the
- 3829 DN as identified by the certificate
- 3830 • Clients that browse the Registry need not use certificates. The Registry must assign the
- 3831 default RegistryGuest role to such clients.

3832 **Appendix A Web Service Architecture**3833 **A.1 Registry Service Abstract Specification**

```

3834 <?xml version = "1.0" encoding = "UTF-8"?>
3835 <definitions name = "RegistryService"
3836   targetNamespace = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
3837   xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
3838   xmlns:xsd1 = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
3839   xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
3840   xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
3841   xmlns = "http://schemas.xmlsoap.org/wsdl/"
3842
3843   <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
3844   "http://schemas.xmlsoap.org/wsdl/ file:///c:/jsews/ebxmlrr-spec/misc/schema/wsdl.xsd"-->
3845   <documentation>$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/services/Registry.wsdl,v 1.9
3846   2001/11/13 00:51:07 farrukh_najmi Exp $
3847   This is the the normative abstract WSDL service definition for the OASIS ebXML Registry
3848   services.</documentation>
3849
3850   <import namespace = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0" location =
3851   "http://www.oasis-open.org/ebxml/registry/2.0/schemas/Registry.xsd"/>
3852   <!-- Commonly re-used Messages -->
3853
3854   <message name = "RegistryResponse">
3855     <part name = "RegistryResponse" element = "xsd1:RegistryResponse"/>
3856     <!--This part is optional and contains the mime/multipart containing content for a
3857     GetContentRequest-->
3858
3859     <part name = "content" type = "xsd:any"/>
3860   </message>
3861   <!-- Messages used by QueryManager -->
3862
3863   <message name = "GetContentsRequest">
3864     <part name = "GetContentsRequest" element = "xsd1:GetContentsRequest"/>
3865   </message>
3866
3867   <message name = "SubmitAdhocQueryRequest">
3868     <part name = "SubmitAdhocQueryRequest" element = "xsd1:SubmitAdhocQueryRequest"/>
3869   </message>
3870
3871   <!-- Messages used by LifeCycleManager -->
3872   <message name = "AddSlotsRequest">
3873     <part name = "AddSlotsRequest" element = "xsd1:AddSlotsRequest"/>
3874   </message>
3875   <message name = "ApproveObjectsRequest">
3876     <part name = "ApproveObjectsRequest" element = "xsd1:ApproveObjectsRequest"/>
3877   </message>
3878   <message name = "DeprecateObjectsRequest">
3879     <part name = "DeprecateObjectsRequest" element = "xsd1:DeprecateObjectsRequest"/>
3880   </message>
3881   <message name = "RemoveObjectsRequest">
3882     <part name = "RemoveObjectsRequest" element = "xsd1:RemoveObjectsRequest"/>
3883   </message>
3884   <message name = "RemoveSlotsRequest">
3885     <part name = "RemoveSlotsRequest" element = "xsd1:RemoveSlotsRequest"/>
3886   </message>
3887   <message name = "SubmitObjectsRequest">
3888     <part name = "SubmitObjectsRequest" element = "xsd1:SubmitObjectsRequest"/>
3889     <!--This part is the mime/multipart containing content-->
3890     <part name = "content" type = "xsd:any"/>
3891   </message>
3892
3893   <portType name = "QueryManagerPortType">
3894     <documentation>Maps to the QueryManager interface of Registry Services
3895     spec.</documentation>
3896
3897     <operation name = "getContents">
3898       <input message = "tns:GetContentsRequest"/>
3899       <output message = "tns:RegistryResponse"/>
3900

```

```

3901         </operation>
3902
3903         <operation name = "submitAdhocQuery">
3904             <input message = "tns:SubmitAdhocQueryRequest"/>
3905             <output message = "tns:RegistryResponse"/>
3906         </operation>
3907     </portType>
3908
3909     <portType name = "LifeCycleManagerPortType">
3910         <documentation>Maps to the LifeCycleManager interface of Registry Services
3911 spec.</documentation>
3912         <operation name = "addSlots">
3913             <input message = "tns:AddSlotsRequest"/>
3914             <output message = "tns:RegistryResponse"/>
3915         </operation>
3916         <operation name = "approveObjectsRequest">
3917             <input message = "tns:ApproveObjectsRequest"/>
3918             <output message = "tns:RegistryResponse"/>
3919         </operation>
3920         <operation name = "deprecateObjectsRequest">
3921             <input message = "tns:DeprecateObjectsRequest"/>
3922             <output message = "tns:RegistryResponse"/>
3923         </operation>
3924         <operation name = "removeObjectsRequest">
3925             <input message = "tns:RemoveObjectsRequest"/>
3926             <output message = "tns:RegistryResponse"/>
3927         </operation>
3928         <operation name = "removeSlotsRequest">
3929             <input message = "tns:RemoveSlotsRequest"/>
3930             <output message = "tns:RegistryResponse"/>
3931         </operation>
3932         <operation name = "submitObjectsRequest">
3933             <input message = "tns:SubmitObjectsRequest"/>
3934             <output message = "tns:RegistryResponse"/>
3935         </operation>
3936     </portType>
3937 </definitions>
3938

```

3939 A.2 Registry Service SOAP Binding

```

3940 <?xml version = "1.0" encoding = "UTF-8"?>
3941 <definitions name = "RegistryServiceSOAPBinding"
3942     targetNamespace = "urn:oasis:names:tc:ebxml-regrep:soapbinding:wSDL:2.0"
3943     xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:soapbinding:wSDL:2.0"
3944     xmlns:registry = "urn:oasis:names:tc:ebxml-regrep:services:wSDL:2.0"
3945     xmlns:soap = "http://schemas.xmlsoap.org/wSDL/soap/"
3946     xmlns:wSDL = "http://schemas.xmlsoap.org/wSDL/"
3947     xmlns:mime = "http://schemas.xmlsoap.org/wSDL/mime/"
3948     xmlns = "http://schemas.xmlsoap.org/wSDL/"
3949
3950
3951
3952
3953
3954     <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
3955 "http://schemas.xmlsoap.org/wSDL/ file:///C:/jsews/ebxmlrr-spec/misc/schema/wSDL.xsd"-->
3956     <documentation>$Header: /cvsroot/ebxmlrr/ebxmlrr-
3957 spec/misc/services/RegistrySOAPBinding.wSDL,v 1.11 2001/11/20 12:29:17 farrukh_najmi Exp $
3958 This is the the normative concrete SOAP/HTTP binding for the OASIS ebXML Registry
3959 services.</documentation>
3960
3961     <!--Import the definition of the abstract OASIS ebXML Registry services-->
3962     <import namespace = "urn:oasis:names:tc:ebxml-regrep:services:wSDL:2.0"
3963         location = "http://www.oasis-open.org/ebxml/registry/1.1/services/Registry.wSDL"/>
3964
3965     <!--The SOAP bindings to the abstract services follow-->
3966
3967     <binding name = "QueryManagerSoapBinding" type = "tns:QueryManagerPortType">
3968         <!--
3969         transport attribute below specifies use of http transport for SOAP binding.
3970         Currently this is the only normative definition of transport for SOAP binding.
3971         -->
3972

```

```

3973     <soap:binding style = "document" transport =
3974 "http://schemas.xmlsoap.org/soap/http"/>
3975
3976     <operation name = "getContent">
3977         <soap:operation soapAction =
3978 "uri:oasis:ebxml:registry:services:ObjectQueryManager:getContent"/>
3979         <input>
3980             <soap:body use = "literal"/>
3981         </input>
3982
3983         <output>
3984             <mime:multipartRelated>
3985                 <mime:part>
3986                     <soap:body parts = "RegistryResponse" use =
3987 "literal"/>
3988                 </mime:part>
3989                 <mime:part>
3990                     <mime:content part = "content" type = "*/"/>
3991                 </mime:part>
3992             </mime:multipartRelated>
3993         </output>
3994     </operation>
3995
3996     <operation name = "submitAdhocQuery">
3997         <soap:operation soapAction =
3998 "uri:oasis:ebxml:registry:services:QueryManager:submitAdhocQueries"/>
3999         <input>
4000             <soap:body use = "literal"/>
4001         </input>
4002         <output>
4003             <soap:body use = "literal"/>
4004         </output>
4005     </operation>
4006 </binding>
4007
4008     <binding name = "LifeCycleManagerSoapBinding" type = "tns:LifeCycleManagerPortType">
4009         <soap:binding style = "document" transport =
4010 "http://schemas.xmlsoap.org/soap/http"/>
4011
4012         <operation name = "addSlots">
4013             <soap:operation soapAction =
4014 "uri:oasis:ebxml:registry:services:LifeCycleManager:addSlots"/>
4015             <input>
4016                 <soap:body use = "literal"/>
4017             </input>
4018             <output>
4019                 <soap:body use = "literal"/>
4020             </output>
4021         </operation>
4022
4023         <operation name = "approveObjects">
4024             <soap:operation soapAction =
4025 "uri:oasis:ebxml:registry:services:LifeCycleManager:approveObjects"/>
4026             <input>
4027                 <soap:body use = "literal"/>
4028             </input>
4029             <output>
4030                 <soap:body use = "literal"/>
4031             </output>
4032         </operation>
4033
4034         <operation name = "deprecateObjects">
4035             <!--Need undeprecateObjects?!-->
4036
4037             <soap:operation soapAction =
4038 "uri:oasis:ebxml:registry:services:LifeCycleManager:deprecateObjects"/>
4039             <input>
4040                 <soap:body use = "literal"/>
4041             </input>
4042             <output>
4043                 <soap:body use = "literal"/>
4044             </output>
4045         </operation>
4046
4047         <operation name = "removeObjects">

```

```

4048     <soap:operation soapAction =
4049 "uri:osis:ebxml:registry:services:QueryManager:removeObjects"/>
4050     <input>
4051         <soap:body use = "literal"/>
4052     </input>
4053     <output>
4054         <soap:body use = "literal"/>
4055     </output>
4056 </operation>
4057
4058     <operation name = "removeSlots">
4059         <soap:operation soapAction =
4060 "uri:osis:ebxml:registry:services:QueryManager:removeSlots"/>
4061         <input>
4062             <soap:body use = "literal"/>
4063         </input>
4064         <output>
4065             <soap:body use = "literal"/>
4066         </output>
4067     </operation>
4068
4069     <operation name = "submitObjects">
4070         <soap:operation soapAction =
4071 "uri:osis:ebxml:registry:services:QueryManager:submitObjects"/>
4072         <input>
4073             <mime:multipartRelated>
4074                 <mime:part>
4075                     <soap:body parts = "SubmitObjectsRequest" use =
4076 "literal"/>
4077                 </mime:part>
4078                 <mime:part>
4079                     <mime:content part = "content" type = "*/*/">
4080                 </mime:part>
4081             </mime:multipartRelated>
4082         </input>
4083         <output>
4084             <soap:body use = "literal"/>
4085         </output>
4086     </operation>
4087
4088 </binding>
4089 <!--The concrete services bound to the SOAP binding follows-->
4090
4091     <service name = "RegistryService">
4092         <documentation>The QueryManager service of OASIS ebXML registry version
4093 1.1</documentation>
4094         <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
4095             <soap:address location = "http://your_URL_to_your_QueryManager"/>
4096         </port>
4097         <port name = "LifeCycleManagerSOAPBinding" binding =
4098 "tns:LifeCycleManagerSOAPBinding">
4099             <soap:address location = "http://your_URL_to_your_QueryManager"/>
4100         </port>
4101     </service>
4102 </definitions>
4103

```

4105 Appendix B ebXML Registry Schema Definitions

4106 B.1 RIM Schema

```

4107 <?xml version = "1.0" encoding = "UTF-8"?>
4108 <?xml version = "1.0" encoding = "UTF-8"?>
4109 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
4110 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
4111     targetNamespace = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
4112     xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
4113     >
4114     <annotation>
4115         <documentation xml:lang = "en">The schema for OASIS ebXML Registry Information
4116 Model</documentation>
4117     </annotation>
4118     <!--$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/schema/rim.xsd,v 1.18 2001/11/24 01:45:44
4119 farrukh_najmi Exp $-->
4120
4121
4122     <!--Begin information model mapping from ebRIM.-->
4123
4124     <complexType name = "AssociationType1">
4125         <annotation>
4126             <documentation xml:lang = "en">
4127 Association is the mapping of the same named interface in ebRIM.
4128 It extends RegistryObject.
4129 An Association specifies references to two previously submitted
4130 registry entrys.
4131 The sourceObject is id of the sourceObject in association
4132 The targetObject is id of the targetObject in association
4133     </documentation>
4134     </annotation>
4135     <complexContent>
4136         <extension base = "tns:RegistryObjectType">
4137             <attribute name = "associationType" use = "required" type = "string"/>
4138             <attribute name = "sourceObject" use = "required" type = "IDREF"/>
4139             <attribute name = "targetObject" use = "required" type = "IDREF"/>
4140         </extension>
4141     </complexContent>
4142     </complexType>
4143     <element name = "Association" type = "tns:AssociationType1"/>
4144     <complexType name = "AuditableEventType">
4145         <annotation>
4146             <documentation xml:lang = "en">
4147 Mapping of the same named interface in ebRIM.
4148     </documentation>
4149         </annotation>
4150         <complexContent>
4151             <extension base = "tns:RegistryObjectType">
4152                 <attribute name = "eventType" use = "required">
4153                     <simpleType>
4154                         <restriction base = "NMTOKEN">

```



```

4158     <enumeration value = "Created"/>
4159     <enumeration value = "Deleted"/>
4160     <enumeration value = "Deprecated"/>
4161     <enumeration value = "Updated"/>
4162     <enumeration value = "Versioned"/>
4163   </restriction>
4164 </simpleType>
4165 </attribute>
4166 <attribute name = "registryObject" use = "required" type = "IDREF"/>
4167 <attribute name = "timestamp" use = "required" type = "string"/>
4168 <attribute name = "user" use = "required" type = "IDREF"/>
4169 </extension>
4170 </complexContent>
4171 </complexType>
4172 <element name = "AuditableEvent" type = "tns:AuditableEventType"/>
4173
4174 <complexType name = "ClassificationType">
4175   <annotation>
4176     <documentation xml:lang = "en">
4177 Classification is the mapping of the same named interface in ebRIM.
4178 It extends RegistryObject.
4179
4180 A Classification specifies references to two registry entrys.
4181
4182 The classifiedObject is id of the Object being classified.
4183 The classificationNode is id of the ClassificationNode classyng the object
4184   </documentation>
4185   </annotation>
4186   <complexContent>
4187     <extension base = "tns:RegistryObjectType">
4188       <attribute name = "classificationScheme" use = "optional" type = "IDREF"/>
4189       <attribute name = "classifiedObject" use = "required" type = "IDREF"/>
4190       <attribute name = "classificationNode" use = "required" type = "IDREF"/>
4191       <attribute name = "nodeRepresentation" use = "optional" type = "string"/>
4192     </extension>
4193   </complexContent>
4194 </complexType>
4195 <element name = "Classification" type = "tns:ClassificationType"/>
4196
4197 <complexType name = "ClassificationNodeType">
4198   <annotation>
4199     <documentation xml:lang = "en">
4200 ClassificationNode is the mapping of the same named interface in ebRIM.
4201 It extends RegistryObject.
4202
4203 ClassificationNode is used to submit a Classification tree to the Registry.
4204
4205 The parent attribute is the id to the parent node. code is an optional code value for a
4206 ClassificationNode
4207 often defined by an external taxonomy (e.g. NAICS)
4208   </documentation>
4209   </annotation>
4210   <complexContent>
4211     <extension base = "tns:RegistryObjectType">
4212
4213     <sequence>
4214       <element ref="tns:ClassificationNode" minOccurs="0" maxOccurs="unbounded" />

```

```

4215     </sequence>
4216     <attribute name = "parent" type = "IDREF"/>
4217     <attribute name = "code" type = "string"/>
4218   </extension>
4219 </complexContent>
4220 </complexType>
4221 <element name = "ClassificationNode" type = "tns:ClassificationNodeType"/>
4222
4223 <complexType name = "ClassificationSchemeType">
4224   <annotation>
4225     <documentation xml:lang = "en">
4226 ClassificationScheme is the mapping of the same named interface in ebRIM.
4227 It extends RegistryEntry.
4228   </documentation>
4229 </annotation>
4230 <complexContent>
4231   <extension base = "tns:RegistryEntryType">
4232     <sequence>
4233       <element ref = "tns:ClassificationNode" minOccurs = "0" maxOccurs = "unbounded"/>
4234     </sequence>
4235     <attribute name = "isInternal" use = "required" type = "boolean"/>
4236     <attribute name = "nodeType" use = "required">
4237       <simpleType>
4238         <restriction base = "NMTOKEN">
4239           <enumeration value = "UniqueCode"/>
4240           <enumeration value = "EmbeddedPath"/>
4241           <enumeration value = "NonUniqueCode"/>
4242         </restriction>
4243       </simpleType>
4244     </attribute>
4245   </extension>
4246 </complexContent>
4247 </complexType>
4248 <element name = "ClassificationScheme" type = "tns:ClassificationSchemeType"/>
4249 <complexType name = "ExternalIdentifierType">
4250   <annotation>
4251     <documentation xml:lang = "en">
4252 ExternalIdentifier is the mapping of the same named interface in ebRIM.
4253 It extends RegistryObject.
4254   </documentation>
4255 </annotation>
4256 <complexContent>
4257   <extension base = "tns:RegistryObjectType">
4258     <attribute name = "identificationScheme" use = "required" type = "IDREF"/>
4259     <attribute name = "value" use = "required" type = "string"/>
4260   </extension>
4261 </complexContent>
4262 </complexType>
4263 <element name = "ExternalIdentifier" type = "tns:ExternalIdentifierType"/>
4264 <complexType name = "ExternalLinkType">
4265   <annotation>
4266     <documentation xml:lang = "en">
4267 ExternalLink is the mapping of the same named interface in ebRIM.
4268 It extends RegistryObject.
4269   </documentation>
4270 </annotation>
4271 <complexContent>

```

```

4272     <extension base = "tns:RegistryObjectType">
4273         <attribute name = "externalURI" type = "anyURI"/>
4274     </extension>
4275 </complexContent>
4276 </complexType>
4277 <element name = "ExternalLink" type = "tns:ExternalLinkType"/>
4278 <complexType name = "ExtrinsicObjectType">
4279     <annotation>
4280         <documentation xml:lang = "en">
4281 ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.
4282 It inherits RegistryEntryAttributes
4283     </documentation>
4284 </annotation>
4285 <!-- Need clarity on use of contentURI in both directions -->
4286 <complexContent>
4287     <extension base = "tns:RegistryEntryType">
4288         <attribute name = "contentURI" use = "required" type = "anyURI"/>
4289         <attribute name = "mimeType" type = "string"/>
4290         <attribute name = "opaque" use = "optional" type = "boolean"/>
4291     </extension>
4292 </complexContent>
4293 </complexType>
4294
4295 <!--Following element decl neds to be lower case but using upper camel case for backward
4296 compatibility-->
4297
4298 <element name = "ExtrinsicObject" type = "tns:ExtrinsicObjectType"/>
4299 <element name = "Address" type = "tns:PostalAddressType"/>
4300 <complexType name = "OrganizationType">
4301     <annotation>
4302         <documentation xml:lang = "en">
4303 Mapping of the same named interface in ebRIM.
4304     </documentation>
4305 </annotation>
4306 <complexContent>
4307     <extension base = "tns:RegistryObjectType">
4308         <sequence maxOccurs = "unbounded">
4309             <element ref = "tns:Address"/>
4310             <element ref = "tns:TelephoneNumber"/>
4311         </sequence>
4312         <attribute name = "parent" type = "IDREF"/>
4313         <attribute name = "primaryContact" use = "required" type = "IDREF"/>
4314     </extension>
4315 </complexContent>
4316 </complexType>
4317 <element name = "Organization" type = "tns:OrganizationType"/>
4318 <complexType name = "SlotType1">
4319     <sequence minOccurs = "0">
4320         <element ref = "tns:ValueList"/>
4321     </sequence>
4322     <attribute name = "name" use = "required" type = "string"/>
4323     <attribute name = "slotType" use = "optional" type = "string"/>
4324 </complexType>
4325 <element name = "Slot" type = "tns:SlotType1"/>
4326 <complexType name = "ValueListType">
4327     <sequence maxOccurs = "unbounded">
4328         <element name = "Value" type = "string"/>

```

```
4329     </sequence>
4330 </complexType>
4331 <element name = "ValueList" type = "tns:ValueListType"/>
4332 <complexType name = "PersonNameType">
4333   <annotation>
4334     <documentation xml:lang = "en">
4335 Mapping of the same named interface in ebRIM.
4336   </documentation>
4337 </annotation>
4338 <sequence minOccurs = "0" maxOccurs = "unbounded">
4339   <element ref = "tns:Slot"/>
4340 </sequence>
4341 <attribute name = "firstName" use = "required" type = "string"/>
4342 <attribute name = "middleName" type = "string"/>
4343 <attribute name = "lastName" use = "required" type = "string"/>
4344 </complexType>
4345 <element name = "PersonName" type = "tns:PersonNameType"/>
4346
4347 <complexType name = "PostalAddressType">
4348   <annotation>
4349     <documentation xml:lang = "en">
4350 Mapping of the same named interface in ebRIM.
4351   </documentation>
4352 </annotation>
4353 <sequence minOccurs = "0" maxOccurs = "unbounded">
4354   <element ref = "tns:Slot"/>
4355 </sequence>
4356 <attribute name = "city" use = "optional" type = "string"/>
4357 <attribute name = "country" use = "optional" type = "string"/>
4358 <attribute name = "postalCode" use = "optional" type = "string"/>
4359 <attribute name = "state" use = "optional" type = "string"/>
4360 <attribute name = "street" use = "optional" type = "string"/>
4361 <attribute name = "streetNumber" use = "optional" type = "string"/>
4362 </complexType>
4363 <element name = "PostalAddress" type = "tns:PostalAddressType"/>
4364
4365 <complexType name = "RegistryEntryType">
4366   <complexContent>
4367     <extension base = "tns:RegistryObjectType">
4368       <attribute name = "expiration" use = "optional" type = "date"/>
4369       <attribute name = "majorVersion" default = "1" type = "integer"/>
4370       <attribute name = "minorVersion" default = "0" type = "integer"/>
4371       <attribute name = "stability" use = "optional">
4372         <simpleType>
4373           <restriction base = "NMTOKEN">
4374             <enumeration value = "Dynamic"/>
4375             <enumeration value = "DynamicCompatible"/>
4376             <enumeration value = "Static"/>
4377           </restriction>
4378         </simpleType>
4379       </attribute>
4380
4381       <attribute name = "status">
4382         <simpleType>
4383           <restriction base = "NMTOKEN">
4384             <enumeration value = "Submitted"/>
4385             <enumeration value = "Approved"/>
```

```

4386         <enumeration value = "Deprecated"/>
4387         <enumeration value = "Withdrawn"/>
4388     </restriction>
4389 </simpleType>
4390 </attribute>
4391     <attribute name = "userVersion" use = "optional" type = "string"/>
4392 </extension>
4393 </complexContent>
4394 </complexType>
4395 <element name = "RegistryEntry" type = "tns:RegistryEntryType"/>
4396
4397 <complexType name = "InternationalStringType">
4398     <sequence minOccurs = "0" maxOccurs = "unbounded">
4399         <element name = "LocalizedString" type = "tns:LocalizedStringType"/>
4400     </sequence>
4401 </complexType>
4402 <element name = "InternationalString" type = "tns:InternationalStringType"/>
4403
4404 <complexType name = "LocalizedStringType">
4405     <attribute name = "lang" use = "optional" default = "en-us" form = "qualified" type = "language"/>
4406     <attribute name = "charset" use = "optional" default = "UTF-8" />
4407     <attribute name = "value" use = "required" type = "string"/>
4408 </complexType>
4409
4410 <complexType name = "RegistryObjectType">
4411     <annotation>
4412         <documentation xml:lang = "en">
4413 id may be empty. If specified it may be in urn:uuid format or be in some
4414 arbitrary format. If id is empty registry must generate globally unique id.
4415
4416 If id is provided and in proper UUID syntax (starts with urn:uuid:)
4417 registry will honour it.
4418
4419 If id is provided and is not in proper UUID syntax then it is used for
4420 linkage within document and is ignored by the registry. In this case the
4421 registry generates a UUID for id attribute.
4422
4423 id must not be null when object is being retrieved from the registry.
4424
4425     </documentation>
4426 </annotation>
4427     <sequence minOccurs = "0" maxOccurs = "1">
4428         <element name = "Name" type = "tns:InternationalStringType" minOccurs = "0"/>
4429         <element name = "Description" type = "tns:InternationalStringType" minOccurs = "0"/>
4430         <element ref = "tns:Slot" minOccurs = "0" maxOccurs = "unbounded"/>
4431         <element ref = "tns:Classification" minOccurs = "0" maxOccurs = "unbounded"/>
4432         <element ref = "tns:ExternalIdentifier" minOccurs = "0" maxOccurs = "unbounded"/>
4433     </sequence>
4434     <attribute name = "accessControlPolicy" use = "optional" type = "IDREF"/>
4435     <attribute name = "id" type = "ID"/>
4436     <attribute name = "objectType" use = "optional" type = "string"/>
4437 </complexType>
4438 <element name = "RegistryObject" type = "tns:RegistryObjectType"/>
4439 <complexType name = "RegistryPackageType">
4440     <annotation>
4441         <documentation xml:lang = "en">
4442 RegistryPackage is the mapping of the same named interface in ebRIM.

```

```

4443 It extends RegistryEntry.
4444
4445 A RegistryPackage is a named collection of objects.
4446   </documentation>
4447   </annotation>
4448   <complexContent>
4449     <extension base = "tns:RegistryEntryType">
4450       <sequence>
4451         <element ref = "tns:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
4452       </sequence>
4453     </extension>
4454   </complexContent>
4455 </complexType>
4456 <element name = "RegistryPackage" type = "tns:RegistryPackageType"/>
4457 <complexType name = "ServiceType">
4458   <complexContent>
4459     <extension base = "tns:RegistryEntryType">
4460       <sequence minOccurs = "0" maxOccurs = "unbounded">
4461         <element ref = "tns:ServiceBinding"/>
4462       </sequence>
4463     </extension>
4464   </complexContent>
4465 </complexType>
4466 <element name = "Service" type = "tns:ServiceType"/>
4467 <complexType name = "ServiceBindingType">
4468   <complexContent>
4469     <extension base = "tns:RegistryObjectType">
4470       <sequence minOccurs = "0" maxOccurs = "unbounded">
4471         <element ref = "tns:SpecificationLink"/>
4472       </sequence>
4473     </extension>
4474   </complexContent>
4475 </complexType>
4476 <element name = "ServiceBinding" type = "tns:ServiceBindingType"/>
4477 <complexType name = "SpecificationLinkType">
4478   <complexContent>
4479     <extension base = "tns:RegistryObjectType"/>
4480   </complexContent>
4481 </complexType>
4482 <element name = "SpecificationLink" type = "tns:SpecificationLinkType"/>
4483
4484 <!--??Need to fix TelephoneNumbers at the cost of backward compatibility-->
4485
4486 <complexType name = "TelephoneNumberType">
4487   <annotation>
4488     <documentation xml:lang = "en">
4489 TelephoneNumber is the mapping of the same named interface in ebRIM.
4490     </documentation>
4491   </annotation>
4492   <attribute name = "areaCode" use = "required" type = "string"/>
4493   <attribute name = "contryCode" use = "required" type = "string"/>
4494   <attribute name = "extension" type = "string"/>
4495   <attribute name = "number" use = "required" type = "string"/>
4496   <attribute name = "url" type = "anyURI"/>
4497 </complexType>
4498 <element name = "TelephoneNumber" type = "tns:TelephoneNumberType"/>
4499 <element name = "FaxNumber" type = "tns:TelephoneNumberType"/>

```

```

4500 <element name = "MobileTelephoneNumber" type = "tns:TelephoneNumberType"/>
4501 <element name = "PagerNumber" type = "tns:TelephoneNumberType"/>
4502 <complexType name = "TelephoneNumberListType">
4503   <sequence>
4504     <element ref = "tns:TelephoneNumber" minOccurs = "0" maxOccurs = "unbounded"/>
4505   </sequence>
4506 </complexType>
4507 <complexType name = "UserType">
4508   <annotation>
4509     <documentation xml:lang = "en">
4510 Mapping of the same named interface in ebRIM.
4511 </documentation>
4512 </annotation>
4513 <complexContent>
4514   <extension base = "tns:RegistryObjectType">
4515     <sequence maxOccurs = "unbounded">
4516       <element ref = "tns:Address"/>
4517       <element ref = "tns:PersonName"/>
4518       <element ref = "tns:TelephoneNumber"/>
4519     </sequence>
4520     <attribute name = "organization" type = "IDREF"/>
4521     <attribute name = "email" type = "string"/>
4522     <attribute name = "url" type = "anyURI"/>
4523   </extension>
4524 </complexContent>
4525 </complexType>
4526 <element name = "User" type = "tns:UserType"/>
4527 <complexType name = "ObjectRefType">
4528   <annotation>
4529     <documentation xml:lang = "en">
4530 Use to reference an Object by its id.
4531 Specifies the id attribute of the object as its id attribute.
4532 id attribute in ObjectAttributes is exactly the same syntax and semantics as
4533 id attribute in RegistryObject.
4534 </documentation>
4535 </annotation>
4536   <attribute name = "id" type = "ID"/>
4537 </complexType>
4538 <element name = "ObjectRef" type = "tns:ObjectRefType"/>
4539
4540 <element name = "ObjectRefList">
4541   <annotation>
4542     <documentation xml:lang = "en">A list of ObjectRefs</documentation>
4543   </annotation>
4544   <complexContent>
4545     <sequence minOccurs = "0" maxOccurs = "unbounded">
4546       <element ref = "tns:ObjectRef"/>
4547     </sequence>
4548   </complexContent>
4549 </element>
4550
4551
4552 <complexType name = "LeafRegistryObjectListType">
4553   <choice minOccurs = "0" maxOccurs = "unbounded">
4554     <element ref = "tns:ObjectRef"/>
4555     <element ref = "tns:Association"/>
4556     <element ref = "tns:AuditableEvent"/>

```

```

4557 <element ref = "tns:Classification"/>
4558 <element ref = "tns:ClassificationNode"/>
4559 <element ref = "tns:ClassificationScheme"/>
4560 <element ref = "tns:ExternalIdentifier"/>
4561 <element ref = "tns:ExternalLink"/>
4562 <element ref = "tns:ExtrinsicObject"/>
4563 <element ref = "tns:Organization"/>
4564 <element ref = "tns:RegistryPackage"/>
4565 <element ref = "tns:Service"/>
4566 <element ref = "tns:ServiceBinding"/>
4567 <element ref = "tns:SpecificationLink"/>
4568 <element ref = "tns:User"/>
4569 </choice>
4570 </complexType>
4571 <element name = "LeafRegistryObjectList" type = "tns:LeafRegistryObjectListType"/>
4572
4573 <complexType name = "RegistryObjectListType">
4574 <complexContent>
4575 <extension base = "tns:LeafRegistryObjectListType">
4576 <choice minOccurs = "0" maxOccurs = "unbounded">
4577 <element ref = "tns:RegistryEntry"/>
4578 <element ref = "tns:RegistryObject"/>
4579 </choice>
4580 </extension>
4581 </complexContent>
4582 </complexType>
4583 <element name = "RegistryObjectList" type = "tns:RegistryObjectListType"/>
4584
4585
4586 </schema>
4587

```

4588 B.2 Query Schema

```

4589 <?xml version = "1.0" encoding = "UTF-8"?>
4590
4591 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
4592 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
4593 targetNamespace = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
4594 xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
4595 xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0">
4596
4597 <!-- Import the rim.xsd file with XML schema mappaing from RIM -->
4598 <import namespace = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0" schemaLocation = "./rim.xsd"/>
4599
4600 <complexType name = "ResponseOptionType">
4601 <attribute name = "returnType" default="RegistryObject">
4602 <simpleType>
4603 <restriction base = "NMTOKEN">
4604 <enumeration value = "ObjectRef"/>
4605 <enumeration value = "RegistryObject"/>
4606 <enumeration value = "RegistryEntry"/>
4607 <enumeration value = "LeafClass"/>
4608 <enumeration value = "LeafClassWithRepositoryItem"/>
4609 </restriction>
4610 </simpleType>
4611

```



```

4612     </attribute>
4613
4614     <attribute name = "returnComposedObjects" type="boolean" default="false"/>
4615
4616 </complexType>
4617 <element name = "ResponseOption" type = "tns:ResponseOptionType"/>
4618
4619 <element name = "AdhocQueryRequest">
4620   <annotation>
4621     <documentation xml:lang = "en">
4622 An Ad hoc query request specifies a query string as defined by [RS] in the queryString attribute
4623
4624     </documentation>
4625   </annotation>
4626   <complexType>
4627     <sequence>
4628       <element ref = "tns:ResponseOption" minOccurs="1" maxOccurs="1" />
4629       <choice minOccurs="1" maxOccurs="1">
4630         <element ref = "tns:FilterQuery"/>
4631         <element ref = "tns:SQLQuery"/>
4632       </choice>
4633     </sequence>
4634   </complexType>
4635 </element>
4636 <element name = "SQLQuery" type = "string"/>
4637
4638 <element name = "AdhocQueryResponse">
4639   <annotation>
4640     <documentation xml:lang = "en">
4641 The response includes a RegistryObjectList which has zero or more
4642 RegistryObjects that match the query specified in AdhocQueryRequest.
4643
4644     </documentation>
4645   </annotation>
4646   <complexType>
4647     <choice>
4648       <element ref = "tns:SQLQueryResult"/>
4649       <element ref = "tns:FilterQueryResult"/>
4650     </choice>
4651   </complexType>
4652 </element>
4653
4654 <element name = "SQLQueryResult" type = "rim:RegistryObjectListType"/>
4655
4656 <element name = "FilterQuery">
4657   <complexType>
4658     <choice minOccurs = "1" maxOccurs = "1">
4659       <element ref = "tns:RegistryObjectQuery"/>
4660       <element ref = "tns:RegistryEntryQuery"/>
4661       <element ref = "tns:AuditableEventQuery"/>
4662       <element ref = "tns:ClassificationNodeQuery"/>
4663       <element ref = "tns:ClassificationSchemeQuery"/>
4664       <element ref = "tns:RegistryPackageQuery"/>
4665       <element ref = "tns:ExtrinsicObjectQuery"/>
4666       <element ref = "tns:OrganizationQuery"/>
4667       <element ref = "tns:ServiceQuery"/>
4668     </choice>

```

```
4669     </complexType>
4670 </element>
4671
4672 <complexType name = "RegistryObjectQueryType">
4673   <sequence>
4674     <element ref = "tns:RegistryObjectFilter" minOccurs = "0" maxOccurs="1"/>
4675     <element name = "NameBranch" type = "tns:InternationalStringBranchType" minOccurs = "0" maxOccurs
4676 = "1"/>
4677     <element name = "DescriptionBranch" type = "tns:InternationalStringBranchType" minOccurs = "0"
4678 maxOccurs = "1"/>
4679     <element ref = "tns:SourceAssociationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
4680     <element ref = "tns:TargetAssociationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
4681     <element ref = "tns:ClassificationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
4682     <element ref = "tns:ExternalIdentifierFilter" minOccurs = "0" maxOccurs = "unbounded"/>
4683     <element ref = "tns:SlotBranch" minOccurs = "0" maxOccurs = "unbounded"/>
4684   </sequence>
4685 </complexType>
4686 <element name = "RegistryObjectQuery" type = "tns:RegistryObjectQueryType"/>
4687
4688 <complexType name = "InternationalStringBranchType">
4689   <sequence>
4690     <element ref = "tns:LocalizedStringFilter" minOccurs = "0" maxOccurs="unbounded"/>
4691   </sequence>
4692 </complexType>
4693
4694 <complexType name = "RegistryEntryQueryType">
4695   <complexContent>
4696     <extension base = "tns:RegistryObjectQueryType">
4697       <sequence>
4698         <element ref = "tns:RegistryEntryFilter" minOccurs = "0" maxOccurs="1"/>
4699       </sequence>
4700     </extension>
4701   </complexContent>
4702 </complexType>
4703 <element name = "RegistryEntryQuery" type = "tns:RegistryEntryQueryType"/>
4704
4705 <complexType name = "ClassificationSchemeQueryType">
4706   <complexContent>
4707     <extension base = "tns:RegistryEntryQueryType">
4708       <sequence>
4709         <element ref = "tns:ClassificationSchemeFilter" minOccurs = "0" maxOccurs="1"/>
4710       </sequence>
4711     </extension>
4712   </complexContent>
4713 </complexType>
4714 <element name = "ClassificationSchemeQuery" type = "tns:ClassificationSchemeQueryType"/>
4715
4716 <complexType name = "AuditableEventQueryType">
4717   <complexContent>
4718     <extension base = "tns:RegistryObjectQueryType">
4719       <sequence>
4720         <element ref = "tns:AuditableEventFilter" minOccurs = "0"/>
4721         <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs = "1"/>
4722         <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs = "1"/>
4723         <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs = "1"/>
4724       </sequence>
4725     </extension>
```

```
4726     </complexContent>
4727 </complexType>
4728 <element name = "AuditableEventQuery" type = "tns:AuditableEventQueryType"/>
4729
4730 <complexType name = "ClassificationNodeQueryType">
4731   <complexContent>
4732     <extension base = "tns:RegistryObjectQueryType">
4733       <sequence>
4734         <element ref = "tns:ClassificationNodeFilter" minOccurs = "0" maxOccurs="1"/>
4735         <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
4736         <element name = "ClassificationNodeParentBranch" type="tns:ClassificationNodeQueryType"
4737 minOccurs = "0" maxOccurs="1"/>
4738         <element name = "ClassificationNodeChildrenBranch" type="tns:ClassificationNodeQueryType"
4739 minOccurs = "0" maxOccurs="unbounded"/>
4740       </sequence>
4741     </extension>
4742   </complexContent>
4743 </complexType>
4744 <element name = "ClassificationNodeQuery" type = "tns:ClassificationNodeQueryType"/>
4745
4746 <complexType name = "RegistryPackageQueryType">
4747   <complexContent>
4748     <extension base = "tns:RegistryEntryQueryType">
4749       <sequence>
4750         <element ref = "tns:RegistryPackageFilter" minOccurs = "0" maxOccurs="1"/>
4751         <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs = "unbounded"/>
4752         <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs = "unbounded"/>
4753       </sequence>
4754     </extension>
4755   </complexContent>
4756 </complexType>
4757 <element name = "RegistryPackageQuery" type = "tns:RegistryPackageQueryType"/>
4758
4759 <complexType name = "ExtrinsicObjectQueryType">
4760   <complexContent>
4761     <extension base = "tns:RegistryEntryQueryType">
4762       <sequence>
4763         <element ref = "tns:ExtrinsicObjectFilter" minOccurs = "0" maxOccurs="1"/>
4764       </sequence>
4765     </extension>
4766   </complexContent>
4767 </complexType>
4768 <element name = "ExtrinsicObjectQuery" type = "tns:ExtrinsicObjectQueryType"/>
4769
4770 <complexType name = "OrganizationQueryType">
4771   <complexContent>
4772     <extension base = "tns:RegistryObjectQueryType">
4773       <sequence>
4774         <element ref = "tns:OrganizationFilter" minOccurs = "0" maxOccurs="1"/>
4775         <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
4776         <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="unbounded"/>
4777         <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs = "1"/>
4778         <element name = "OrganizationParentQuery" type="tns:OrganizationQueryType" minOccurs = "0"
4779 maxOccurs="1" />
4780         <element name = "OrganizationChildrenQuery" type="tns:OrganizationQueryType" minOccurs = "0"
4781 maxOccurs="unbounded"/>
4782       </sequence>
4783     </extension>
```

```
4784     </complexContent>
4785 </complexType>
4786 <element name = "OrganizationQuery" type = "tns:OrganizationQueryType"/>
4787
4788 <complexType name = "ServiceQueryType">
4789   <complexContent>
4790     <extension base = "tns:RegistryEntryQueryType">
4791       <sequence>
4792         <element ref = "tns:ServiceFilter" minOccurs = "0" maxOccurs="1"/>
4793
4794         <element ref = "tns:ServiceBindingBranch" minOccurs = "0" maxOccurs="unbounded"/>
4795
4796       </sequence>
4797     </extension>
4798   </complexContent>
4799 </complexType>
4800 <element name = "ServiceQuery" type = "tns:ServiceQueryType"/>
4801
4802 <element name = "ServiceBindingBranch">
4803   <complexType>
4804     <sequence>
4805       <element ref = "tns:ServiceBindingFilter" minOccurs = "0" maxOccurs="1"/>
4806       <element ref = "tns:SpecificationLinkBranch" minOccurs = "0" maxOccurs="unbounded"/>
4807
4808     </sequence>
4809   </complexType>
4810 </element>
4811
4812 <element name = "SpecificationLinkBranch">
4813   <complexType>
4814     <sequence>
4815       <element ref = "tns:SpecificationLinkFilter" minOccurs = "0" maxOccurs="1"/>
4816       <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
4817       <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
4818     </sequence>
4819   </complexType>
4820 </element>
4821
4822 <element name = "FilterQueryResult" >
4823   <complexType>
4824     <choice minOccurs = "1" maxOccurs = "1">
4825       <element ref = "tns:RegistryObjectQueryResult"/>
4826       <element ref = "tns:RegistryEntryQueryResult"/>
4827       <element ref = "tns:AuditableEventQueryResult"/>
4828       <element ref = "tns:ClassificationNodeQueryResult"/>
4829       <element ref = "tns:ClassificationSchemeQueryResult"/>
4830       <element ref = "tns:RegistryPackageQueryResult"/>
4831       <element ref = "tns:ExtrinsicObjectQueryResult"/>
4832       <element ref = "tns:OrganizationQueryResult"/>
4833       <element ref = "tns:ServiceQueryResult"/>
4834     </choice>
4835   </complexType>
4836 </element>
4837
4838 <element name = "RegistryObjectQueryResult" type = "rim:RegistryObjectListType"/>
4839
4840
```

```
4841 <element name = "RegistryEntryQueryResult">
4842   <complexType>
4843     <choice minOccurs = "0" maxOccurs = "unbounded">
4844       <element ref = "rim:ObjectRef"/>
4845       <element ref = "rim:ClassificationScheme"/>
4846       <element ref = "rim:ExtrinsicObject"/>
4847       <element ref = "rim:RegistryEntry"/>
4848       <element ref = "rim:RegistryObject"/>
4849       <element ref = "rim:RegistryPackage"/>
4850     </choice>
4851   </complexType>
4852 </element>
4853
4854 <element name = "AuditableEventQueryResult">
4855   <complexType>
4856     <choice minOccurs = "0" maxOccurs = "unbounded">
4857       <element ref = "rim:ObjectRef"/>
4858       <element ref = "rim:RegistryObject"/>
4859       <element ref = "rim:AuditableEvent"/>
4860     </choice>
4861   </complexType>
4862 </element>
4863
4864 <element name = "ClassificationNodeQueryResult">
4865   <complexType>
4866     <choice minOccurs = "0" maxOccurs = "unbounded">
4867       <element ref = "rim:ObjectRef"/>
4868       <element ref = "rim:RegistryObject"/>
4869       <element ref = "rim:ClassificationNode"/>
4870     </choice>
4871   </complexType>
4872 </element>
4873
4874 <element name = "ClassificationSchemeQueryResult">
4875   <complexType>
4876     <choice minOccurs = "0" maxOccurs = "unbounded">
4877       <element ref = "rim:ObjectRef"/>
4878       <element ref = "rim:RegistryObject"/>
4879       <element ref = "rim:RegistryEntry"/>
4880       <element ref = "rim:ClassificationScheme"/>
4881     </choice>
4882   </complexType>
4883 </element>
4884
4885 <element name = "RegistryPackageQueryResult">
4886   <complexType>
4887     <choice minOccurs = "0" maxOccurs = "unbounded">
4888       <element ref = "rim:ObjectRef"/>
4889       <element ref = "rim:RegistryEntry"/>
4890       <element ref = "rim:RegistryObject"/>
4891       <element ref = "rim:RegistryPackage"/>
4892     </choice>
4893   </complexType>
4894 </element>
4895
4896 <element name = "ExtrinsicObjectQueryResult">
4897   <complexType>
```

```
4898     <choice minOccurs = "0" maxOccurs = "unbounded">
4899         <element ref = "rim:ObjectRef"/>
4900         <element ref = "rim:RegistryObject"/>
4901         <element ref = "rim:RegistryEntry"/>
4902         <element ref = "rim:ExtrinsicObject"/>
4903     </choice>
4904 </complexType>
4905 </element>
4906
4907 <element name = "OrganizationQueryResult">
4908     <complexType>
4909         <choice minOccurs = "0" maxOccurs = "unbounded">
4910             <element ref = "rim:ObjectRef"/>
4911             <element ref = "rim:RegistryObject"/>
4912             <element ref = "rim:Organization"/>
4913         </choice>
4914     </complexType>
4915 </element>
4916
4917 <element name = "ServiceQueryResult">
4918     <complexType>
4919         <choice minOccurs = "0" maxOccurs = "unbounded">
4920             <element ref = "rim:ObjectRef"/>
4921             <element ref = "rim:RegistryObject"/>
4922             <element ref = "rim:RegistryEntry"/>
4923             <element ref = "rim:Service"/>
4924         </choice>
4925     </complexType>
4926 </element>
4927
4928 <complexType name = "AssociationBranchType">
4929     <sequence>
4930         <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs="1"/>
4931         <choice minOccurs = "0" maxOccurs="1">
4932             <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
4933             <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
4934             <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
4935             <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
4936             <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
4937             <element ref = "tns:AuditableEventQuery" minOccurs = "0" maxOccurs="1"/>
4938             <element ref = "tns:RegistryPackageQuery" minOccurs = "0" maxOccurs="1"/>
4939             <element ref = "tns:ExtrinsicObjectQuery" minOccurs = "0" maxOccurs="1"/>
4940             <element ref = "tns:ServiceQuery" minOccurs = "0" maxOccurs="1"/>
4941             <element ref = "tns:ExternalLinkFilter" minOccurs = "0" maxOccurs="1"/>
4942             <element ref = "tns:ExternalIdentifierFilter" minOccurs = "0" maxOccurs="1"/>
4943             <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs="1"/>
4944             <element ref = "tns:SourceAssociationBranch" minOccurs = "0" maxOccurs="1"/>
4945             <element ref = "tns:TargetAssociationBranch" minOccurs = "0" maxOccurs="1"/>
4946             <element ref = "tns:ClassifiedByBranch" minOccurs = "0" maxOccurs="1"/>
4947             <element ref = "tns:ServiceBindingBranch" minOccurs = "0" maxOccurs="1"/>
4948             <element ref = "tns:SpecificationLinkBranch" minOccurs = "0" maxOccurs="1"/>
4949         </choice>
4950     </sequence>
4951 </complexType>
4952 <element name = "SourceAssociationBranch" type="tns:AssociationBranchType" />
4953 <element name = "TargetAssociationBranch" type="tns:AssociationBranchType" />
4954
```

```

4955 <element name = "ClassificationBranch">
4956   <complexType>
4957     <sequence>
4958       <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
4959       <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
4960       <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
4961     </sequence>
4962   </complexType>
4963 </element>
4964
4965 <element name = "ClassifiedByBranch">
4966   <complexType>
4967     <sequence>
4968       <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
4969       <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
4970       <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
4971       <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
4972       <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
4973     </sequence>
4974   </complexType>
4975 </element>
4976
4977 <element name = "SlotBranch">
4978   <complexType>
4979     <sequence>
4980       <element ref = "tns:SlotFilter" minOccurs = "0" maxOccurs="1"/>
4981       <element ref = "tns:SlotValueFilter" minOccurs = "0" maxOccurs = "unbounded"/>
4982     </sequence>
4983   </complexType>
4984 </element>
4985
4986 <element name = "OrganizationParentBranch">
4987   <complexType>
4988     <sequence>
4989       <element ref = "tns:OrganizationFilter" minOccurs = "0" maxOccurs="1"/>
4990       <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
4991       <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="1"/>
4992       <element ref = "tns:OrganizationParentBranch" minOccurs = "0"/>
4993     </sequence>
4994   </complexType>
4995 </element>
4996
4997 <element name = "UserBranch">
4998   <complexType>
4999     <sequence>
5000       <element ref = "tns:UserFilter" minOccurs = "0" maxOccurs="1"/>
5001       <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
5002       <element ref = "tns:TelephoneNumberFilter" minOccurs = "0"
5003 maxOccurs="unbounded"/>
5004       <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
5005     </sequence>
5006   </complexType>
5007 </element>
5008
5009 <complexType name = "FilterType">
5010   <sequence>
5011

```

```

5012     <element ref = "tns:Clause"/>
5013   </sequence>
5014 </complexType>
5015
5016
5017   <element name = "RegistryObjectFilter" type = "tns:FilterType"/>
5018   <element name = "RegistryEntryFilter" type = "tns:FilterType"/>
5019   <element name = "ExtrinsicObjectFilter" type = "tns:FilterType"/>
5020   <element name = "RegistryPackageFilter" type = "tns:FilterType"/>
5021   <element name = "OrganizationFilter" type = "tns:FilterType"/>
5022   <element name = "ClassificationNodeFilter" type = "tns:FilterType"/>
5023   <element name = "AssociationFilter" type = "tns:FilterType"/>
5024   <element name = "ClassificationFilter" type = "tns:FilterType"/>
5025   <element name = "ClassificationSchemeFilter" type = "tns:FilterType"/>
5026   <element name = "ExternalLinkFilter" type = "tns:FilterType"/>
5027   <element name = "ExternalIdentifierFilter" type = "tns:FilterType"/>
5028   <element name = "SlotFilter" type = "tns:FilterType"/>
5029   <element name = "AuditableEventFilter" type = "tns:FilterType"/>
5030   <element name = "UserFilter" type = "tns:FilterType"/>
5031   <element name = "SlotValueFilter" type = "tns:FilterType"/>
5032   <element name = "PostalAddressFilter" type = "tns:FilterType"/>
5033   <element name = "TelephoneNumberFilter" type = "tns:FilterType"/>
5034   <element name = "ServiceFilter" type = "tns:FilterType"/>
5035   <element name = "ServiceBindingFilter" type = "tns:FilterType"/>
5036   <element name = "SpecificationLinkFilter" type = "tns:FilterType"/>
5037   <element name = "LocalizedStringFilter" type = "tns:FilterType"/>
5038
5039   <element name = "Clause">
5040     <annotation>
5041       <documentation xml:lang = "en">
5042         The following lines define the XML syntax for Clause.
5043
5044         </documentation>
5045       </annotation>
5046     <complexType>
5047       <choice>
5048         <element ref = "tns:SimpleClause"/>
5049         <element ref = "tns:CompoundClause"/>
5050       </choice>
5051     </complexType>
5052   </element>
5053   <element name = "SimpleClause">
5054     <complexType>
5055       <choice>
5056         <element ref = "tns:BooleanClause"/>
5057         <element ref = "tns:RationalClause"/>
5058         <element ref = "tns:StringClause"/>
5059       </choice>
5060       <attribute name = "leftArgument" use = "required" type = "string"/>
5061     </complexType>
5062   </element>
5063   <element name = "CompoundClause">
5064     <complexType>
5065       <sequence>
5066         <element ref = "tns:Clause"/>
5067         <element ref = "tns:Clause" maxOccurs = "unbounded"/>
5068       </sequence>

```



```
5069     <attribute name = "connectivePredicate" use = "required">
5070     <simpleType>
5071         <restriction base = "NMTOKEN">
5072             <enumeration value = "And"/>
5073             <enumeration value = "Or"/>
5074         </restriction>
5075     </simpleType>
5076 </attribute>
5077 </complexType>
5078 </element>
5079 <element name = "BooleanClause">
5080     <complexType>
5081         <attribute name = "booleanPredicate" use = "required" type = "boolean"/>
5082     </complexType>
5083 </element>
5084 <element name = "RationalClause">
5085     <complexType>
5086         <choice>
5087             <element ref = "tns:IntClause"/>
5088             <element ref = "tns:FloatClause"/>
5089         </choice>
5090     <attribute name = "logicalPredicate" use = "required">
5091     <simpleType>
5092         <restriction base = "NMTOKEN">
5093             <enumeration value = "LE"/>
5094             <enumeration value = "LT"/>
5095             <enumeration value = "GE"/>
5096             <enumeration value = "GT"/>
5097             <enumeration value = "EQ"/>
5098             <enumeration value = "NE"/>
5099         </restriction>
5100     </simpleType>
5101 </attribute>
5102 </complexType>
5103 </element>
5104 <element name = "IntClause" type = "integer"/>
5105 <element name = "FloatClause" type = "float"/>
5106 <element name = "StringClause">
5107     <complexType>
5108         <simpleContent>
5109             <extension base = "string">
5110                 <attribute name = "stringPredicate" use = "required">
5111                 <simpleType>
5112                     <restriction base = "NMTOKEN">
5113                         <enumeration value = "contains"/>
5114                         <enumeration value = "-contains"/>
5115                         <enumeration value = "startswith"/>
5116                         <enumeration value = "-startswith"/>
5117                         <enumeration value = "equal"/>
5118                         <enumeration value = "-equal"/>
5119                         <enumeration value = "endswith"/>
5120                         <enumeration value = "-endswith"/>
5121                     </restriction>
5122                 </simpleType>
5123                 </attribute>
5124             </extension>
5125         </simpleContent>
```

```

5126     </complexType>
5127 </element>
5128
5129     <element name = "GetContentRequest">
5130     <annotation>
5131         <documentation xml:lang = "en">
5132 Gets the actual content (not metadata) specified by the ObjectRefList
5133         </documentation>
5134     </annotation>
5135
5136     <complexType>
5137     <sequence>
5138         <element ref = "rim:ObjectRefList"/>
5139     </sequence>
5140     </complexType>
5141 </element>
5142
5143     <element name = "GetContentResponse">
5144     <annotation>
5145         <documentation xml:lang = "en">
5146 The GetObjectsResponse will have no sub-elements if there were no errors.
5147 The actual contents will be in the other payloads of the message.
5148         </documentation>
5149     </annotation>
5150     <complexType/>
5151 </element>
5152
5153 </schema>
5154

```

5155 **B.3 Registry Services Interface Schema**

```

5156 <?xml version = "1.0" encoding = "UTF-8"?>
5157 <!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
5158 <!--$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/schema/rs.xsd,v 1.17 2001/11/23 18:52:12 farrukh_najmi
5159 Exp $-->
5160
5161 <schema xmlns = "http://www.w3.org/2001/XMLSchema"
5162     targetNamespace = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
5163     xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
5164     xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
5165     xmlns:query = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
5166     >
5167     <annotation>
5168         <documentation xml:lang = "en">The schema for OASIS ebXML Registry Services</documentation>
5169     </annotation>
5170
5171     <!-- Import the rim.xsd file with XML schema mappaing from RIM -->
5172     <import namespace="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0" schemaLocation="./rim.xsd"/>
5173
5174     <!-- Import the query.xsd file with XML schema for query related schema -->
5175     <import namespace="urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0" schemaLocation="./query.xsd"/>
5176
5177
5178     <element name = "RequestAcceptedResponse">
5179     <annotation>
5180         <documentation xml:lang = "en">

```

```

5181 Mapping of the same named interface in ebRS.
5182   </documentation>
5183   </annotation>
5184   <complexType/>
5185 </element>
5186
5187   <element name = "SubmitObjectsRequest">
5188     <annotation>
5189       <documentation xml:lang = "en">
5190 The SubmitObjectsRequest allows one to submit a list of RegistryObject elements. Each RegistryEntry
5191 element provides metadata for a single submitted object. Note that the repository item being submitted
5192 is in a separate document that is not in this DTD. The ebXML Messaging Services Specification defines
5193 packaging, for submission, of the metadata of a repository item with the repository item itself. The
5194 value of the contentURI attribute of the ExtrinsicObject element must be the same as the xlink:href
5195 attribute within the Reference element within the Manifest element of the MessageHeader.
5196       </documentation>
5197     </annotation>
5198     <complexType>
5199       <sequence>
5200         <element ref = "rim:LeafRegistryObjectList"/>
5201       </sequence>
5202     </complexType>
5203 </element>
5204
5205   <element name = "UpdateObjectsRequest">
5206     <annotation>
5207       <documentation xml:lang = "en">
5208 The UpdateObjectsRequest allows one to update a list of RegistryObject elements. Each RegistryEntry
5209 element provides metadata for a single submitted object. Note that the repository item being submitted
5210 is in a separate document that is not in this DTD. The ebXML Messaging Services Specification defines
5211 packaging, for submission, of the metadata of a repository item with the repository item itself. The
5212 value of the contentURI attribute of the ExtrinsicObject element must be the same as the xlink:href
5213 attribute within the Reference element within the Manifest element of the MessageHeader.
5214       </documentation>
5215     </annotation>
5216     <complexType>
5217       <sequence>
5218         <element ref = "rim:LeafRegistryObjectList"/>
5219       </sequence>
5220     </complexType>
5221 </element>
5222
5223   <element name = "AddSlotsRequest">
5224     <complexType>
5225       <sequence>
5226         <element ref = "rim:ObjectRef" minOccurs="1" maxOccurs="1"/>
5227         <element ref = "rim:Slot" minOccurs="1" maxOccurs="unbounded"/>
5228       </sequence>
5229     </complexType>
5230 </element>
5231   <element name = "RemoveSlotsRequest">
5232     <annotation>
5233       <documentation xml:lang = "en"> Only need name in Slot within SlotList </documentation>
5234     </annotation>
5235     <complexType>
5236       <sequence>
5237         <element ref = "rim:ObjectRef" minOccurs="1" maxOccurs="1"/>
5238         <element ref = "rim:Slot" minOccurs="1" maxOccurs="unbounded"/>
5239       </sequence>

```

```
5240     </complexType>
5241 </element>
5242 <element name = "ApproveObjectsRequest">
5243   <annotation>
5244     <documentation xml:lang = "en">
5245 The ObjectRefList is the list of
5246 refs to the registry entrys being approved.
5247     </documentation>
5248   </annotation>
5249   <complexType>
5250     <sequence>
5251       <element ref = "rim:ObjectRefList"/>
5252     </sequence>
5253   </complexType>
5254 </element>
5255 <element name = "DeprecateObjectsRequest">
5256   <annotation>
5257     <documentation xml:lang = "en">
5258 The ObjectRefList is the list of
5259 refs to the registry entrys being deprecated.
5260
5261     </documentation>
5262   </annotation>
5263   <complexType>
5264     <sequence>
5265       <element ref = "rim:ObjectRefList"/>
5266     </sequence>
5267   </complexType>
5268 </element>
5269 <element name = "RemoveObjectsRequest">
5270   <annotation>
5271     <documentation xml:lang = "en">
5272 The ObjectRefList is the list of
5273 refs to the registry entrys being removed
5274
5275     </documentation>
5276   </annotation>
5277
5278   <complexType>
5279     <sequence>
5280       <element ref = "rim:ObjectRefList"/>
5281     </sequence>
5282     <attribute name = "deletionScope" use = "optional">
5283       <simpleType>
5284         <restriction base = "NMTOKEN">
5285           <enumeration value = "DeleteAll"/>
5286           <enumeration value = "DeleteRepositoryItemOnly"/>
5287         </restriction>
5288       </simpleType>
5289     </attribute>
5290   </complexType>
5291 </element>
5292
5293
5294
5295 <element name = "RegistryProfile">
5296   <annotation>
```

```

5297     <documentation xml:lang = "en">
5298 Describes the capability profile for the registry and what optional features
5299 are supported
5300
5301     </documentation>
5302 </annotation>
5303 <complexType>
5304   <sequence>
5305     <element ref = "tns:OptionalFeaturesSupported"/>
5306   </sequence>
5307   <attribute name = "version" use = "required" type = "string"/>
5308 </complexType>
5309 </element>
5310
5311 <element name = "OptionalFeaturesSupported">
5312   <complexType>
5313     <attribute name = "sqlQuery" default = "false" type = "boolean"/>
5314     <attribute name = "xQuery" default = "false" type = "boolean"/>
5315   </complexType>
5316 </element>
5317
5318 <simpleType name = "ErrorType">
5319   <restriction base = "NMTOKEN">
5320     <enumeration value = "Warning"/>
5321     <enumeration value = "Error"/>
5322   </restriction>
5323 </simpleType>
5324 <element name = "RegistryErrorList">
5325   <annotation>
5326     <documentation xml:lang = "en"> End FilterQuery DTD </documentation>
5327     <documentation xml:lang = "en"> Begin RegistryError definition </documentation>
5328     <documentation xml:lang = "en"> The RegistryErrorList is derived from the ErrorList element from
5329 the ebXML Message Service Specification </documentation>
5330   </annotation>
5331   <complexType>
5332     <sequence>
5333       <element ref = "tns:RegistryError" maxOccurs = "unbounded"/>
5334     </sequence>
5335     <attribute name = "highestSeverity" use = "optional" type = "tns:ErrorType"/>
5336   </complexType>
5337 </element>
5338 <element name = "RegistryError">
5339   <complexType>
5340     <simpleContent>
5341       <extension base = "string">
5342         <attribute name = "codeContext" use = "required" type = "string"/>
5343         <attribute name = "errorCode" use = "required" type = "string"/>
5344         <attribute name = "severity" use = "optional" type = "tns:ErrorType"/>
5345         <attribute name = "location" type = "string"/>
5346       </extension>
5347     </simpleContent>
5348   </complexType>
5349 </element>
5350 <element name = "RegistryResponse">
5351   <complexType>
5352     <sequence>
5353     <choice minOccurs = "0">

```

```
5354     <element ref = "query:AdhocQueryResponse" />
5355     <element ref = "query:GetContentResponse" />
5356   </choice>
5357   <element ref = "tns:RegistryErrorList" minOccurs = "0" />
5358 </sequence>
5359 <attribute name = "status" use = "required">
5360   <simpleType>
5361     <restriction base = "NMTOKEN">
5362       <enumeration value = "success" />
5363       <enumeration value = "warning" />
5364       <enumeration value = "failure" />
5365       <enumeration value = "unavailable" />
5366     </restriction>
5367   </simpleType>
5368 </attribute>
5369 </complexType>
5370 </element>
5371
5372 <element name = "RootElement">
5373   <annotation>
5374     <documentation xml:lang = "en"> The contrived root node </documentation>
5375   </annotation>
5376   <complexType>
5377     <choice>
5378       <element ref = "tns:SubmitObjectsRequest" />
5379       <element ref = "tns:UpdateObjectsRequest" />
5380       <element ref = "tns:ApproveObjectsRequest" />
5381       <element ref = "tns:DeprecateObjectsRequest" />
5382       <element ref = "tns:RemoveObjectsRequest" />
5383       <element ref = "query:AdhocQueryRequest" />
5384       <element ref = "query:GetContentRequest" />
5385       <element ref = "tns:AddSlotsRequest" />
5386       <element ref = "tns:RemoveSlotsRequest" />
5387       <element ref = "tns:RegistryResponse" />
5388       <element ref = "tns:RegistryProfile" />
5389     </choice>
5390   </complexType>
5391 </element>
5392 </schema>
```

5393 **Appendix C Interpretation of UML Diagrams**

5394 This section describes in *abstract terms* the conventions used to define ebXML business process
5395 description in UML.

5396 **C.1 UML Class Diagram**

5397 A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP]) required
5398 to implement an ebXML Registry Services and clients. See on page **Error! Bookmark not**
5399 **defined. for an example.** The UML class diagram contains:

5400

5401 1. A collection of UML interfaces where each interface represents a Service Interface for a
5402 Registry service.

5403 2. Tabular description of methods on each interface where each method represents an
5404 Action (as defined by [ebCPP]) within the Service Interface representing the UML
5405 interface.

5406 3. Each method within a UML interface specifies one or more parameters, where the type of
5407 each method argument represents the ebXML message type that is exchanged as part of
5408 the Action corresponding to the method. Multiple arguments imply multiple payload
5409 documents within the body of the corresponding ebXML message.

5410 **C.2 UML Sequence Diagram**

5411 A UML sequence diagram is used to specify the business protocol representing the interactions
5412 between the UML interfaces for a Registry specific ebXML business process. A UML sequence
5413 diagram provides the necessary information to determine the sequencing of messages, request to
5414 response association as well as request to error response association as described by [ebCPP].

5415 Each sequence diagram shows the sequence for a specific conversation protocol as method calls
5416 from the requestor to the responder. Method invocation may be synchronous or asynchronous
5417 based on the UML notation used on the arrow-head for the link. A half arrow-head represents
5418 asynchronous communication. A full arrow-head represents synchronous communication.

5419 Each method invocation may be followed by a response method invocation from the responder to
5420 the requestor to indicate the ResponseName for the previous Request. Possible error response is
5421 indicated by a conditional response method invocation from the responder to the requestor. See
5422 Figure 8 on page 26 for an example.

5423 Appendix D SQL Query

5424 D.1 SQL Query Syntax Specification

5425 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The
5426 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax
5427 conforms to the <query specification>, modulo the restrictions identified below:

- 5428 1. A <select list> may contain at most one <select sublist>.
- 5429 2. In a <select list> must be is a single column whose data type is UUID, from the table in the
5430 <from clause>.
- 5431 3. A <derived column> may not have an <as clause>.
- 5432 4. <table expression> does not contain the optional <group by clause> and <having clause>
5433 clauses.
- 5434 5. A <table reference> can only consist of <table name> and <correlation name>.
- 5435 6. A <table reference> does not have the optional AS between <table name> and
5436 <correlation name>.
- 5437 7. There can only be one <table reference> in the <from clause>.
- 5438 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows
5439 for the right hand side of the <in predicate> to be limited to a restricted <query
5440 specification> as defined above.
- 5441 9. A <search condition> within the <where clause> may not include a <query expression>.
- 5442 10. Simple joins are allowed only if they are based on indexed columns within the relational
5443 schema.
- 5444 11. The SQL query syntax allows for the use of <sql invoked routines> invocation from
5445 [SQL/PSM] as the RHS of the <in predicate>.

5446 D.2 Non-Normative BNF for Query Syntax Grammar

5447 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as
5448 an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative
5449 syntax. For the normative syntax rules see Appendix D.1.

```
5450 /*****
5451  * The Registry Query (Subset of SQL-92) grammar starts here
5452  *****/
5453 RegistryQuery = SQLSelect [";"]
5454
5455 SQLSelect = "SELECT" ["DISTINCT"] SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
5456
5457 SQLSelectCols = ID
5458
5459 SQLTableList = SQLTableRef
5460
5461 SQLTableRef = ID
5462
5463 SQLWhere = "WHERE" SQLOrExpr
5464
5465 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
```



```

5469 SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*
5470
5471 SQLNotExpr = [ "NOT" ] SQLCompareExpr
5472
5473 SQLCompareExpr =
5474     (SQLColRef "IS") SQLIsClause
5475     | SQLSumExpr [ SQLCompareExprRight ]
5476
5477
5478 SQLCompareExprRight =
5479     SQLLikeClause
5480     | SQLInClause
5481     | SQLCompareOp SQLSumExpr
5482
5483 SQLCompareOp =
5484     "="
5485     | "<>"
5486     | ">"
5487     | ">="
5488     | "<"
5489     | "<="
5490
5491 SQLInClause = [ "NOT" ] "IN" "(" SQLValueList ")"
5492
5493 SQLValueList = SQLValueElement ( "," SQLValueElement )*
5494
5495 SQLValueElement = "NULL" | SQLSelect
5496
5497 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
5498
5499 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
5500
5501 SQLPattern = STRING_LITERAL
5502
5503 SQLLiteral =
5504     STRING_LITERAL
5505     | INTEGER_LITERAL
5506     | FLOATING_POINT_LITERAL
5507
5508 SQLColRef = SQLLvalue
5509
5510 SQLLvalue = SQLLvalueTerm
5511
5512 SQLLvalueTerm = ID ( "." ID )*
5513
5514 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
5515
5516 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr )*
5517
5518 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
5519
5520 SQLTerm = "(" SQLOrExpr ")"
5521     | SQLColRef
5522     | SQLLiteral
5523
5524 INTEGER_LITERAL = ([ "0"-"9" ])+
5525
5526 FLOATING_POINT_LITERAL =
5527     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ (EXPONENT)?
5528     | "." ([ "0"-"9" ])+ (EXPONENT)?
5529     | ([ "0"-"9" ])+ EXPONENT
5530     | ([ "0"-"9" ])+ (EXPONENT)?
5531
5532 EXPONENT = [ "e", "E" ] ( [ "+", "-" ]? ([ "0"-"9" ])+
5533
5534 STRING_LITERAL: "'" (~['"'])* ( "" (~['"'])* ) * "'"
5535
5536 ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
5537 LETTER = [ "A"-"Z", "a"-"z" ]
5538 DIGIT = [ "0"-"9" ]

```

5539 **D.3 Relational Schema For SQL Queries**

5540

```
5541 CREATE TABLE Association (  
5542  
5543 --Object Attributes  
5544   accessControlPolicy VARCHAR(64),  
5545   id VARCHAR(64) NOT NULL PRIMARY KEY,  
5546 --??Should be String32 in RIM not LongName  
5547   objectType VARCHAR(32) DEFAULT 'Association' CHECK (objectType = 'Association'),  
5548  
5549 --Association attributes  
5550   associationType VARCHAR(128) NOT NULL,  
5551   sourceObject VARCHAR(64) NOT NULL,  
5552   targetObject VARCHAR(64) NOT NULL  
5553 );  
5554  
5555 CREATE TABLE AuditableEvent (  
5556  
5557 --Object Attributes  
5558   accessControlPolicy VARCHAR(64),  
5559   id VARCHAR(64) NOT NULL PRIMARY KEY,  
5560   objectType VARCHAR(32) DEFAULT 'AuditableEvent' CHECK (objectType =  
5561 'AuditableEvent'),  
5562  
5563 --AuditableEvent attributes  
5564   eventType VARCHAR(128) NOT NULL,  
5565  
5566 --??What about keyword conflicts. Adding '_' suffix for now  
5567   timeStamp_ TIMESTAMP NOT NULL,  
5568   user_ VARCHAR(64) NOT NULL  
5569 );  
5570  
5571 CREATE TABLE Classification (  
5572 --Object Attributes  
5573   accessControlPolicy VARCHAR(64),  
5574   id VARCHAR(64) NOT NULL PRIMARY KEY,  
5575   objectType VARCHAR(32) DEFAULT 'Classification' CHECK (objectType =  
5576 'Classification'),  
5577  
5578 --Classification attributes.  
5579   classificationNode VARCHAR(64),  
5580   classificationScheme VARCHAR(64),  
5581   classifiedObject VARCHAR(64) NOT NULL,  
5582  
5583 --??What is the right String length for nodeRepresentation  
5584   nodeRepresentation VARCHAR(256)  
5585 );  
5586  
5587  
5588  
5589 CREATE TABLE ClassificationNode (  
5590 --Object Attributes  
5591   accessControlPolicy VARCHAR(64),  
5592   id VARCHAR(64) NOT NULL PRIMARY KEY,  
5593   objectType VARCHAR(32) DEFAULT 'ClassificationNode' CHECK (objectType =  
5594 'ClassificationNode'),  
5595  
5596 --ClassificationNode attributes  
5597   code VARCHAR(64),  
5598  
5599 --??Need to allow for null parent in RIM. Currently this is not so.  
5600   parent VARCHAR(64)  
5601 );  
5602  
5603 CREATE TABLE ClassificationScheme (  
5604 --Object Attributes  
5605   accessControlPolicy VARCHAR(64),  
5606   id VARCHAR(64) NOT NULL PRIMARY KEY,  
5607   objectType VARCHAR(32) DEFAULT 'ClassificationScheme' CHECK (objectType =  
5608 'ClassificationScheme'),  
5609  
5610 --RegistryEntry attributes  
5611   expirationDate TIMESTAMP,  
5612   majorVersion INT DEFAULT 0 NOT NULL,  
5613   minorVersion INT DEFAULT 1 NOT NULL,  
5614  
5615 --??stability and status in RIM1.1 are LongName. Should be ShortName
```

```
5616      stability          VARCHAR(128),
5617      status             VARCHAR(128) NOT NULL,
5618      userVersion        VARCHAR(64),
5619
5620 --ClassificationScheme attributes: currently none defined
5621      isInternal         BOOLEAN DEFAULT false NOT NULL,
5622      nodeType          VARCHAR(32) DEFAULT 'UniqueCode' NOT NULL
5623 );
5624
5625 CREATE TABLE ExternalIdentifier (
5626 --Object Attributes
5627     accessControlPolicy VARCHAR(64),
5628     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5629     objectType          VARCHAR(32) DEFAULT 'ExternalIdentifier' CHECK (objectType =
5630 'ExternalIdentifier'),
5631
5632 --ExternalIdentifier attributes
5633     registryObject      VARCHAR(64) NOT NULL,
5634     identificationScheme VARCHAR(64) NOT NULL,
5635     value               VARCHAR(64) NOT NULL
5636 );
5637
5638 CREATE TABLE ExternalLink (
5639 --Object Attributes
5640     accessControlPolicy VARCHAR(64),
5641     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5642     objectType          VARCHAR(32) DEFAULT 'ExternalLink' CHECK (objectType = 'ExternalLink'),
5643
5644 --ExternalLink attributes
5645     externalURI         VARCHAR(256) NOT NULL
5646 );
5647
5648 CREATE TABLE ExtrinsicObject (
5649 --Object Attributes
5650     accessControlPolicy VARCHAR(64),
5651     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5652     objectType          VARCHAR(32) DEFAULT 'text/xml',
5653
5654 --RegistryEntry attributes
5655     expirationDate      TIMESTAMP,
5656     majorVersion        INT DEFAULT 0 NOT NULL,
5657     minorVersion        INT DEFAULT 1 NOT NULL,
5658
5659     --??stability and status in RIM1.1 are LongName. Should be ShortName
5660     stability           VARCHAR(128),
5661     status              VARCHAR(128) NOT NULL,
5662     userVersion         VARCHAR(64),
5663
5664 --ExtrinsicObject attributes
5665     contentURI          VARCHAR(256),
5666     isOpaque            BOOLEAN DEFAULT false NOT NULL,
5667     mimeType            VARCHAR(128) NOT NULL
5668 );
5669
5670 CREATE TABLE Name (
5671 --LocalizedString attributes flattened for Name
5672     charset             VARCHAR(32),
5673     lang                VARCHAR(32) NOT NULL,
5674     value               VARCHAR(128) NOT NULL,
5675
5676 --The RegistryObject id for teh parent RegistryObject for which this is a Name
5677     parent              VARCHAR(64) NOT NULL,
5678
5679     PRIMARY KEY (parent, lang, value)
5680 );
5681
5682 CREATE TABLE Description (
5683 --LocalizedString attributes flattened for Description
5684     charset             VARCHAR(32),
5685     lang                VARCHAR(32) NOT NULL,
5686     value               VARCHAR(128) NOT NULL,
```

```
5691
5692 --The RegistryObject id for teh parent RegistryObject for which this is a Name
5693 parent          VARCHAR(64) NOT NULL,
5694
5695     PRIMARY KEY (parent, lang, value)
5696 );
5697
5698
5699 CREATE TABLE Organization (
5700
5701 --Object Attributes
5702     accessControlPolicy VARCHAR(64),
5703     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5704     objectType          VARCHAR(32) DEFAULT 'Organization' CHECK (objectType = 'Organization'),
5705
5706 --Organization attributes
5707
5708 --Organization.address attribute is in PostalAddress table
5709
5710     parent              VARCHAR(64),
5711
5712 --primary contact for Organization, points to a User.
5713     primaryContact      VARCHAR(64) NOT NULL
5714
5715 --Organization.telephoneNumbers attribute is in TelephoneNumber table
5716 );
5717
5718 CREATE TABLE RegistryPackage (
5719 --Object Attributes
5720     accessControlPolicy VARCHAR(64),
5721     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5722     objectType          VARCHAR(32) DEFAULT 'RegistryPackage' CHECK (objectType =
5723 'RegistryPackage'),
5724
5725 --RegistryEntry attributes
5726     expirationDate      TIMESTAMP,
5727     majorVersion        INT DEFAULT 0 NOT NULL,
5728     minorVersion        INT DEFAULT 1 NOT NULL,
5729
5730     --??stability and status in RIM1.1 are LongName. Should be ShortName
5731     stability           VARCHAR(128),
5732     status              VARCHAR(128) NOT NULL,
5733     userVersion         VARCHAR(64)
5734
5735 --RegistryPackage attributes: currently none defined
5736 );
5737
5738 CREATE TABLE PostalAddress (
5739
5740     city                VARCHAR(64),
5741     country             VARCHAR(64),
5742     postalCode          VARCHAR(64),
5743     state               VARCHAR(64),
5744     street              VARCHAR(64),
5745     streetNumber        INTEGER,
5746
5747 --The parent object that this is an address for
5748     parent              VARCHAR(64) PRIMARY KEY NOT NULL
5749 );
5750
5751
5752 CREATE TABLE Service (
5753 --Object Attributes
5754     accessControlPolicy VARCHAR(64),
5755     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
5756     objectType          VARCHAR(32) DEFAULT 'Service' CHECK (objectType = 'Service')
5757
5758 --Service attributes: currently none defined
5759 );
5760
5761 CREATE TABLE ServiceBinding (
5762 --Object Attributes
5763     accessControlPolicy VARCHAR(64),
5764     id                   VARCHAR(64) NOT NULL PRIMARY KEY,
```

```
5765     objectType      VARCHAR(32) DEFAULT 'ServiceBinding' CHECK (objectType =
5766 'ServiceBinding'),
5767
5768 --ServiceBinding attributes
5769     service         VARCHAR(64) NOT NULL
5770 );
5771
5772 --Multiple rows of Slot make up a single Slot
5773 CREATE TABLE Slot (
5774     name            VARCHAR(128) NOT NULL,
5775
5776 --??Do we need this or should we get rid of it from RIM
5777     slotType       VARCHAR(128),
5778     value          VARCHAR(64),
5779
5780 --The parent RegistryObject that this is a Slot for
5781     parent         VARCHAR(64) NOT NULL,
5782
5783     PRIMARY KEY (parent, name)
5784 );
5785
5786 CREATE TABLE SpecificationLink (
5787 --Object Attributes
5788     accessControlPolicy VARCHAR(64),
5789     id                  VARCHAR(64) NOT NULL PRIMARY KEY,
5790     objectType         VARCHAR(32) DEFAULT 'SpecificationLink' CHECK (objectType =
5791 'SpecificationLink'),
5792
5793 --SpecificationLink attributes
5794 --??Need to tell Sally to add these two attribute to RIM
5795     service           VARCHAR(64) NOT NULL,
5796     serviceBinding    VARCHAR(64) NOT NULL
5797 );
5798
5799 CREATE TABLE TelephoneNumber (
5800     areaCode        VARCHAR(4),
5801     countryCode     VARCHAR(4),
5802     extension       VARCHAR(8),
5803     number          VARCHAR(8),
5804     phoneType       VARCHAR(128),
5805
5806 --??Remove from RIM and here
5807     url             VARCHAR(256),
5808
5809     parent         VARCHAR(64) NOT NULL PRIMARY KEY
5810 );
5811
5812 CREATE TABLE User_ (
5813 --Object Attributes
5814     accessControlPolicy VARCHAR(64),
5815     id                  VARCHAR(64) NOT NULL PRIMARY KEY,
5816     objectType         VARCHAR(32) DEFAULT 'User' CHECK (objectType = 'User'),
5817
5818 --User attributes
5819 --address is in PostalAddress table
5820
5821     email           VARCHAR(128) NOT NULL,
5822     organization    VARCHAR(64) NOT NULL,
5823
5824 --personName flattened
5825     personName_firstName VARCHAR(64),
5826     personName_middleName VARCHAR(64),
5827     personName_lastName VARCHAR(64),
5828
5829 --telephoneNumbers is in TelephoneNumber table
5830
5831     url             VARCHAR(256)
5832 );
5833
5834 CREATE VIEW RegistryObject (
```

```
5840 --Object Attributes
5841   accessControlPolicy,
5842   id,
5843   objectType
5844
5845 ) AS
5846 SELECT
5847 --Object Attributes
5848   accessControlPolicy,
5849   id,
5850   objectType
5851
5852 FROM Association
5853 UNION
5854
5855 SELECT
5856 --Object Attributes
5857   accessControlPolicy,
5858   id,
5859   objectType
5860
5861 FROM AuditableEvent
5862 UNION
5863
5864 SELECT
5865 --Object Attributes
5866   accessControlPolicy,
5867   id,
5868   objectType
5869
5870 FROM Classification
5871 UNION
5872
5873 SELECT
5874 --Object Attributes
5875   accessControlPolicy,
5876   id,
5877   objectType
5878
5879 FROM ClassificationNode
5880 UNION
5881
5882 SELECT
5883 --Object Attributes
5884   accessControlPolicy,
5885   id,
5886   objectType
5887
5888 FROM ClassificationScheme
5889 UNION
5890
5891 SELECT
5892 --Object Attributes
5893   accessControlPolicy,
5894   id,
5895   objectType
5896
5897 FROM ExternalIdentifier
5898 UNION
5899
5900 SELECT
5901 --Object Attributes
5902   accessControlPolicy,
5903   id,
5904   objectType
5905
5906 FROM ExternalLink
5907 UNION
5908
5909 SELECT
5910 --Object Attributes
5911   accessControlPolicy,
5912   id,
5913   objectType
5914
```

```
5915 FROM ExtrinsicObject
5916 UNION
5917
5918 SELECT
5919 --Object Attributes
5920     accessControlPolicy,
5921     id,
5922     objectType
5923
5924 FROM Organization
5925 UNION
5926
5927 SELECT
5928 --Object Attributes
5929     accessControlPolicy,
5930     id,
5931     objectType
5932
5933 FROM RegistryPackage
5934 UNION
5935
5936 SELECT
5937 --Object Attributes
5938     accessControlPolicy,
5939     id,
5940     objectType
5941
5942 FROM Service
5943 UNION
5944
5945 SELECT
5946 --Object Attributes
5947     accessControlPolicy,
5948     id,
5949     objectType
5950
5951 FROM ServiceBinding
5952 UNION
5953
5954 SELECT
5955 --Object Attributes
5956     accessControlPolicy,
5957     id,
5958     objectType
5959
5960 FROM SpecificationLink
5961 UNION
5962
5963 SELECT
5964 --Object Attributes
5965     accessControlPolicy,
5966     id,
5967     objectType
5968
5969 FROM User_
5970 ;
5971
5972 CREATE VIEW RegistryEntry (
5973 --Object Attributes
5974     accessControlPolicy,
5975     id,
5976     objectType,
5977
5978 --RegistryEntry attributes
5979     expirationDate,
5980     majorVersion,
5981     minorVersion,
5982     stability,
5983     status,
5984     userVersion
5985
5986 ) AS
5987 SELECT
5988 --Object Attributes
5989     accessControlPolicy,
```

```
5990     id,
5991     objectType,
5992
5993 --RegistryEntry attributes
5994     expirationDate,
5995     majorVersion,
5996     minorVersion,
5997     stability,
5998     status,
5999     userVersion
6000
6001 FROM ClassificationScheme
6002 UNION
6003
6004 SELECT
6005 --Object Attributes
6006     accessControlPolicy,
6007     id,
6008     objectType,
6009
6010 --RegistryEntry attributes
6011     expirationDate,
6012     majorVersion,
6013     minorVersion,
6014     stability,
6015     status,
6016     userVersion
6017
6018 FROM ExtrinsicObject
6019 UNION
6020
6021 SELECT
6022 --Object Attributes
6023     accessControlPolicy,
6024     id,
6025     objectType,
6026
6027 --RegistryEntry attributes
6028     expirationDate,
6029     majorVersion,
6030     minorVersion,
6031     stability,
6032     status,
6033     userVersion
6034
6035 FROM RegistryPackage;
6036
6037
6038 --Following is a partial list of indexes. Will need to add more.
6039
6040 --id index
6041 CREATE INDEX id_ASSOCIATION_index ON ASSOCIATION(id);
6042 CREATE INDEX id_AuditableEvent_index ON AuditableEvent(id);
6043 CREATE INDEX id_Classification_index ON Classification(id);
6044 CREATE INDEX id_ClassificationNode_index ON ClassificationNode(id);
6045 CREATE INDEX id_ClassificationScheme_index ON ClassificationScheme(id);
6046 CREATE INDEX id_ExternalIdentifier_index ON ExternalIdentifier(id);
6047 CREATE INDEX id_ExternalLink_index ON ExternalLink(id);
6048 CREATE INDEX id_ExtrinsicObject_index ON ExtrinsicObject(id);
6049 CREATE INDEX id_Organization_index ON Organization(id);
6050 CREATE INDEX id_RegistryPackage_index ON RegistryPackage(id);
6051 CREATE INDEX id_Service_index ON Service(id);
6052 CREATE INDEX id_ServiceBinding_index ON ServiceBinding(id);
6053 CREATE INDEX id_SpecificationLink_index ON SpecificationLink(id);
6054 CREATE INDEX id_User_index ON User_(id);
6055
6056 --name index
6057 CREATE INDEX value_Name_index ON Name(value);
6058 CREATE INDEX lang_value_Name_index ON Name(lang, value);
6059
6060 --description index
6061 CREATE INDEX value_Description_index ON Description(value);
6062 CREATE INDEX lang_value_Description_index ON Description(lang, value);
6063
6064 --Indexes on Association
```



```
6065
6066 CREATE INDEX sourceObject_Association_index ON Association(sourceObject);
6067 CREATE INDEX targetObject_Association_index ON Association(targetObject);
6068 CREATE INDEX associationType_Association_index ON Association(associationType);
6069
6070
6071 --Indexes on Classification
6072
6073 CREATE INDEX classifiedObject_Classification_index ON Classification(classifiedObject);
6074 CREATE INDEX classificationNode_Classification_index ON Classification(classificationNode);
6075
6076 --Indexes on ClassificationNode
6077
6078 CREATE INDEX parent_ClassificationNode_index ON ClassificationNode(parent);
6079 CREATE INDEX code_ClassificationNode_index ON ClassificationNode(code);
6080
6081 --Indexes on ExternalIdentifier
6082
6083 CREATE INDEX registryObject_ExternalIdentifier_index ON ExternalIdentifier(registryObject);
6084
6085 --Indexes on ExternalLink
6086
6087 CREATE INDEX externalURI_ExternalLink_index ON ExternalLink(externalURI);
6088
6089 --Indexes on ExtrinsicObject
6090
6091 CREATE INDEX status_ExtrinsicObject_index ON ExtrinsicObject(status);
6092
6093 --Indexes on Organization
6094
6095 CREATE INDEX parent_Organization_index ON Organization(parent);
6096
6097 --Indexes on PostalAddress
6098
6099 CREATE INDEX parent_PostalAddress_index ON PostalAddress(parent);
6100 CREATE INDEX city_PostalAddress_index ON PostalAddress(city);
6101 CREATE INDEX country_PostalAddress_index ON PostalAddress(country);
6102 CREATE INDEX postalCode_PostalAddress_index ON PostalAddress(postalCode);
6103
6104
6105 --Indexes on ServiceBinding
6106
6107 CREATE INDEX service_ServiceBinding_index ON ServiceBinding(service);
6108
6109 --Indexes on Slot
6110
6111 CREATE INDEX parent_Slot_index ON Slot(parent);
6112 CREATE INDEX name_Slot_index ON Slot(name);
6113
6114 --Indexes on SpecificationLink
6115
6116 CREATE INDEX service_SpecificationLink_index ON SpecificationLink(service);
6117 CREATE INDEX serviceBinding_SpecificationLink_index ON SpecificationLink(serviceBinding);
6118
6119 --Indexes on TelephoneNumber
6120
6121 CREATE INDEX parent_TelephoneNumber_index ON TelephoneNumber(parent);
6122
6123 --Indexes on User
6124
6125 CREATE INDEX organization_User_index ON User_(organization);
6126 CREATE INDEX personName_lastName_User_index ON User_(personName_lastName);
6127
6128 --Defines Stored procedures that map to RIM class methods
6129
6130 --Procedures on RegistryObject
6131 CREATE PROCEDURE RegistryObject_associatedObjects(registryEntryId) {
6132 --Must return a collection of UUIDs for related RegistryObject instances
6133 }
6134
6135 CREATE PROCEDURE RegistryObject_auditTrail(registryEntryId) {
6136 --Must return an collection of UUIDs for AuditableEvents related to the RegistryObject.
6137 --Collection must be in ascending order by timestamp
6138 }
6139
```

```
6140 CREATE PROCEDURE RegistryObject_externalLinks(registryEntryId) {
6141 --Must return a collection of UUIDs for ExternalLinks annotating this RegistryObject.
6142 }
6143
6144 CREATE PROCEDURE RegistryObject_externalIdentifiers(registryEntryId) {
6145 --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryObject.
6146 }
6147
6148 CREATE PROCEDURE RegistryObject_classifications(registryEntryId) {
6149 --Must return a collection of UUIDs for Classifications classifying this RegistryObject.
6150 }
6151
6152 CREATE PROCEDURE RegistryObject_packages(registryEntryId) {
6153 --Must return a collection of UUIDs for Packages that this RegistryObject belongs to.
6154 }
6155
6156
6157 --Procedures on RegistryPackage
6158 CREATE PROCEDURE RegistryPackage_memberObjects(packageId) {
6159 --Must return a collection of UUIDs for RegistryObjects that are members of this Package.
6160 }
6161
6162 --Procedures on ExternalLink
6163 CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
6164 --Must return a collection of UUIDs for objects in this relationship
6165 }
6166
6167 --Procedures on ClassificationNode
6168 CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
6169 --Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
6170 }
```

6171 **Appendix E Non-normative Content Based Ad Hoc Queries**

6172 The Registry SQL query capability supports the ability to search for content based not only on
6173 metadata that catalogs the content but also the data contained within the content itself. For
6174 example it is possible for a client to submit a query that searches for all Collaboration Party
6175 Profiles that define a role named “seller” within a RoleName element in the CPP document itself.
6176 Currently content-based query capability is restricted to XML content.

6177 **E.1.1 Automatic Classification of XML Content**

6178 Content-based queries are indirectly supported through the existing classification mechanism
6179 supported by the Registry.

6180 A submitting organization may define logical indexes on any XML schema or DTD when it is
6181 submitted. An instance of such a logical index defines a link between a specific attribute or
6182 element node in an XML document tree and a ClassificationNode in a classification scheme
6183 within the registry.

6184 The registry utilizes this index to automatically classify documents that are instances of the
6185 schema at the time the document instance is submitted. Such documents are classified according
6186 to the data contained within the document itself.

6187 Such automatically classified content may subsequently be discovered by clients using the
6188 existing classification-based discovery mechanism of the Registry and the query facilities of the
6189 ObjectQueryManager.

6190 [Note] This approach is conceptually similar to the way databases support
6191 indexed retrieval. DBAs define indexes on tables in the schema. When
6192 data is added to the table, the data gets automatically indexed.

6193 **E.1.2 Index Definition**

6194 This section describes how the logical indexes are defined in the SubmittedObject element
6195 defined in the Registry DTD. The complete Registry DTD is specified in Appendix A.

6196 A SubmittedObject element for a schema or DTD may define a collection of
6197 ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList
6198 is ignored if the content being submitted is not of the SCHEMA objectType.

6199 The ClassificationIndex element inherits the attributes of the base class RegistryObject in
6200 [ebRIM]. It then defines specialized attributes as follows:

- 6201 1. classificationNode: This attribute references a specific ClassificationNode by its ID.
- 6202 2. contentIdentifier: This attribute identifies a specific data element within the document
6203 instances of the schema using an XPATH expression as defined by [XPT].

6204 **E.1.3 Example Of Index Definition**

6205 To define an index that automatically classifies a CPP based upon the roles defined within its
6206 RoleName elements, the following index must be defined on the CPP schema or DTD:

```
6207 <ClassificationIndex  
6208   classificationNode='id-for-role-classification-scheme'  
6209   contentIdentifier='/Role//RoleName'  
6210 />  
6211  
6212
```

6213 **E.1.4 Proposed XML Definition**

```
6214 <!--  
6215 A ClassificationIndexList is specified on ExtrinsicObjects of objectType  
6216 'Schema' to define an automatic Classification of instance objects of the  
6217 schema using the specified classificationNode as parent and a  
6218 ClassificationNode created or selected by the object content as selected  
6219 by the contentIdentifier  
6220 -->  
6221 <!ELEMENT ClassificationIndex EMPTY>  
6222 <!ATTLIST ClassificationIndex  
6223     %ObjectAttributes;  
6224     classificationNode IDREF #REQUIRED  
6225     contentIdentifier CDATA #REQUIRED  
6226 >  
6227 <!-- ClassificationIndexList contains new ClassificationIndexes -->  
6228 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>  
6229  
6230  
6231
```

6232 **E.1.5 Example of Automatic Classification**

6233 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the CPP is
6234 submitted it will automatically be classified by two ClassificationNodes named “buyer” and
6235 “seller” that are both children of the ClassificationNode (e.g. a node named Role) specified in the
6236 classificationNode attribute of the ClassificationIndex. If either of the two ClassificationNodes
6237 named “buyer” and “seller” did not previously exist, the ObjectManager would automatically
6238 create these ClassificationNodes.

6239 **Appendix F Security Implementation Guideline**

6240 This section provides a suggested blueprint for how security processing may be implemented in
6241 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have
6242 different implementations as long as they support the default security roles and authorization
6243 rules described in this document.

6244 **F.1 Authentication**

- 6245 1. As soon as a message is received, the first work is the authentication. A principal object is
6246 created.
- 6247 2. If the message is signed, it is verified (including the validity of the certificate) and the DN of
6248 the certificate becomes the identity of the principal. Then the Registry is searched for the
6249 principal and if found, the roles and groups are filled in.
- 6250 3. If the message is not signed, an empty principal is created with the role RegistryGuest. This
6251 step is for symmetry and to decouple the rest of the processing.
- 6252 4. Then the message is processed for the command and the objects it will act on.

6253 **F.2 Authorization**

6254 For every object, the access controller will iterate through all the AccessControlPolicy objects
6255 with the object and see if there is a chain through the permission objects to verify that the
6256 requested method is permitted for the Principal. If any of the permission objects which the object
6257 is associated with has a common role, or identity, or group with the principal, the action is
6258 permitted.

6259 **F.3 Registry Bootstrap**

6260 When a Registry is newly created, a default Principal object should be created with the identity
6261 of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message
6262 signed by the Registry Admin will get all the privileges.

6263 When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the
6264 default AccessControlPolicy. This includes the creation of the necessary Permission instances as
6265 well as the Privileges and Privilege attributes.

6266 **F.4 Content Submission – Client Responsibility**

6267 The Registry client has to sign the contents before submission – otherwise the content will be
6268 rejected.

6269 **F.5 Content Submission – Registry Responsibility**

- 6270 1. As with any other request, the client will first be authenticated. In this case, the Principal
6271 object will get the DN from the certificate.
- 6272 2. As per the request in the message, the RegistryEntry will be created.
- 6273 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 6274 4. If a principal with the identity of the SO is not available, an identity object with the SO's DN
6275 is created.

6276 5. A principal with this identity is created.

6277 **F.6 Content Delete/Deprecate – Client Responsibility**

6278 The Registry client has to sign the payload (not entire message) before submission, for
6279 authentication purposes; otherwise, the request will be rejected

6280 **F.7 Content Delete/Deprecate – Registry Responsibility**

- 6281 1. As with any other request, the client will first be authenticated. In this case, the Principal
6282 object will get the DN from the certificate. As there will be a principal with this identity in
6283 the Registry, the Principal object will get all the roles from that object
- 6284 2. As per the request in the message (delete or deprecate), the appropriate method in the
6285 RegistryObject class will be accessed.
- 6286 3. The access controller performs the authorization by iterating through the Permission objects
6287 associated with this object via the singleton default AccessControlPolicy.
- 6288 4. If authorization succeeds then the action will be permitted. Otherwise an error response is
6289 sent back with a suitable AuthorizationException error message.

6290 **Appendix G Native Language Support (NLS)**

6291 **G.1 Definitions**

6292 Although this section discusses only character set and language, the following terms have to be
6293 defined clearly.

6294 **G.1.1 Coded Character Set (CCS):**

6295 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of
6296 CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

6297 **G.1.2 Character Encoding Scheme (CES):**

6298 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are
6299 ISO-2022, UTF-8.

6300 **G.1.3 Character Set (charset):**

- 6301 • charset is a set of rules for mapping from a sequence of octets to a sequence of
6302 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- 6303 • A list of registered character sets can be found at [IANA].

6304 **G.2 NLS And Request / Response Messages**

6305 For the accurate processing of data in both registry client and registry services, it is essential to
6306 know which character set is used. Although the body part of the transaction may contain the
6307 charset in xml encoding declaration, registry client and registry services shall specify charset
6308 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a
6309 text/xml entity is received with the charset parameter omitted, MIME processors and XML
6310 processors MUST use the default charset value of "us-ascii". For example:

```
6311 Content-Type: text/xml; charset=ISO-2022-JP  
6312  
6313
```

6314 Also, when an application/xml entity is used, the charset parameter is optional, and registry
6315 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which
6316 directly address this contingency.

6317 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

6318 **G.3 NLS And Storing of RegistryObject**

6319 This section provides NLS guidelines on how a registry should store *RegistryObject* instances.

6320 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple
6321 locales. Thus there is no language or character set associated with a specific RegistryObject
6322 instance.

6323 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows.
6324 Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in
6325 RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed
6326 as LocalizedString sub-elements within the XML element representing the I18N capable
6327 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a

6328 specific locale. Each `LocalizedString` element has a `charset` and `lang` attribute as well as a value
6329 attribute of type string.

6330 **G.3.1 Character Set of *LocalizedString***

6331 This is basically an implementation issue because the actual character set that the
6332 ***LocalizedString*** is stored with, does not affect the interface. However, it is highly recommended
6333 to use UTF-8 or UTF-16 for maximum inter-operability.

6334 **G.3.2 Language Information of *LocalizedString***

6335 The language may be specified in `xml:lang` attribute (Section 2.12 [REC-XML]).

6336 **G.4 NLS And Storing of Repository Items**

6337 This section provides NLS guidelines on how a registry should store repository items.

6338 While a single instance of an `ExtrinsicObject` is capable of supporting multiple locales, it is
6339 always associated with a single repository item. The repository item may be in a single locale or
6340 may be in multiple locales. This specification does not specify the repository item.

6341 **G.4.1 Character Set of Repository Items**

6342 The MIME **Content-Type** mime header for the mime multi-part containing the repository
6343 item MAY contain a "**charset**" attribute that specifies the character set used by the repository
6344 item. For example:

6345

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

6346

6347
6348 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset
6349 of a repository item must be preserved as it is originally specified in the transaction.

6350 **G.4.2 Language information of repository item**

6351 The Content-language mime header for the mime bodypart containing the repository item may
6352 specify the language for a locale specific repository item. The value of the Content-language
6353 mime header property must conform to [RFC 1766].

6354 This document currently specifies only the method of sending the information of character set
6355 and language, and how it is stored in a registry. However, the language information may be used
6356 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a
6357 language negotiation procedure, like registry client is asking a favorite language for messages
6358 from registry services, could be another functionality for the future revision of this document.

6359 **Appendix H Terminology Mapping**

6360 While every attempt has been made to use the same terminology used in other works there are
 6361 some terminology differences. The following table shows the terminology mapping between this
 6362 specification and that used in other specifications and working groups.

6363 **Table 12: Terminology Mapping Table**

This Document	OASIS	ISO 11179
“repository item”	RegisteredObject	
RegistryEntry	RegistryEntry	Administered Component
ExternalLink	RelatedData	N/A
Object.id	regEntryId, orgId, etc.	
ExtrinsicObject.uri	objectURL	
ExtrinsicObject.objectType	defnSource, objectType	
RegistryEntry.name	commonName	
Object.description	shortDescription, Description	
ExtrinsicObject.mimeType	objectType=“mime” fileType=“<mime type>”	
Versionable.majorVersion	userVersion only	
Versionable.minorVersion	userVersion only	
RegistryEntry.status	registrationStatus	

6364 **References**

- 6365 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 6366 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 6367 [TA] ebXML Technical Architecture
- 6368 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf
- 6369 [OAS] OASIS Information Model
- 6370 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 6371 [ISO] ISO 11179 Information Model
- 6372 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 6373
- 6374 [ebRIM] ebXML Registry Information Model
- 6375 http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
- 6376 [ebBPM] ebXML Business Process Specification Schema
- 6377 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 6378 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 6379 http://www.ebxml.org/project_teams/trade_partner/private/
- 6380 [ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
- 6381 <http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html>
- 6382 [CTB] Context table informal document from Core Components
- 6383 [ebMS] ebXML Messaging Service Specification, Version 0.21
- 6384 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
- 6385 [ERR] ebXML TRP Error Handling Specification
- 6386 http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf
- 6387 [SEC] ebXML Risk Assessment Technical Report, Version 3.6
- 6388 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 6389 [XPT] XML Path Language (XPath) Version 1.0
- 6390 <http://www.w3.org/TR/xpath>
- 6391 [SQL] Structured Query Language (FIPS PUB 127-2)
- 6392 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 6393 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 6394 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 6395 [IANA] IANA (Internet Assigned Numbers Authority).
- 6396 Official Names for Character Sets, ed. Keld Simonsen et al.
- 6397 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 6398 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
- 6399 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
- 6400 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 6401 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
- 6402 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
- 6403 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 6404 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
- 6405 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
- 6406 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 6407 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:

- 6408 XML Media Types, ed. M. Murata. 2001.
6409 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
- 6410 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
6411 <http://www.w3.org/TR/REC-xml>
- 6412 [UUID] DCE 128 bit Universal Unique Identifier
6413 http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh_20
6414 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
- 6415 [WSDL]W3C Note. Web Services Description Language (WSDL) 1.1
6416 <http://www.w3.org/TR/wsdl>
- 6417 [SOAP11]W3C Note. Simple Object Access Protocol, May 2000,
6418 <http://www.w3.org/TR/SOAP>
- 6419 [SOAPAt]W3C Note: SOAP with Attachments, Dec 2000,
6420 <http://www.w3.org/TR/SOAP-attachments>
- 6421 [XMLDSIG] XML-Signature Syntax and Processing,
6422 <http://www.w3.org/TR/2001/PR-xmlsig-core-20010820/>

6423 **Disclaimer**

6424 The views and specification expressed in this document are those of the authors and are not
6425 necessarily those of their employers. The authors and their employers specifically disclaim
6426 responsibility for any problems arising from correct or incorrect implementation or use of this
6427 design.

6428 **Contact Information**

6429 Team Leader

6430 Name: Lisa Carnahan
6431 Company:
6432 Street:
6433 City, State, Postal Code:
6434 Country: USA
6435 Phone:
6436 Email: lisa.carnahan@????

6437

6438 Vice Team Lead

6439 Name: Yutaka Yoshida
6440 Company: Sun Microsystems
6441 Street: 901 San Antonio Road, MS UMPK17-102
6442 City, State, Postal Code: Palo Alto, CA 94303
6443 Country: USA
6444 Phone: 650.786.5488
6445 Email: Yutaka.Yoshida@eng.sun.com

6446

6447 Editor

6448 Name: Anne A. Fischer
6449 Company: Drummond Group, Inc.
6450 Street: 4700 Bryant Irvin Ct., Suite 303
6451 City, State, Postal Code: Fort Worth, Texas 76107-7645
6452 Country: USA
6453 Phone: 817-371-2367
6454 Email: anne@drummondgroup.com

6455

6456 **Copyright Statement**

6457 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

6458 This document and translations of it may be copied and furnished to others, and derivative works
6459 that comment on or otherwise explain it or assist in its implementation MAY be prepared,
6460 copied, published and distributed, in whole or in part, without restriction of any kind, provided
6461 that the above copyright notice and this paragraph are included on all such copies and derivative
6462 works. However, this document itself MAY not be modified in any way, such as by removing
6463 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
6464 translate it into languages other than English.

6465 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
6466 successors or assigns.

6467 This document and the information contained herein is provided on an "AS IS" basis and
6468 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
6469 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
6470 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
6471 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.