



Creating A Single Global Electronic Market

1

OASIS/ebXML Registry Services Specification v1.08 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/rsV1-08.pdf>

11

12 ***Latest version:***

13 <http://www.oasis-open.org/committees/regrep/documents/rsV1-08.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 Bruce Bargmeyer, EPA
23 Kathryn Breininger, Boeing
24 Dan Chang, IBM
25 Joseph M. Chiusano, LMI
26 Suresh Damodaran, Sterling Commerce
27 Mike DeNicola, Fujitsu
28 John Evdemon, Vitria Technologies
29 Anne Fischer, Drummond Group
30 Sally Fuger, AIAG
31 Len Gallagher, NIST
32 Michael Joya, XMLGlobal
33 Chaemee Kim, KTNET
34 Jong Kim, InnoDigital
35 Kyu-Chul Lee, Chungnam National University
36 Joel Munter, Intel
37 Farrukh Najmi, Sun Microsystems Inc.
38 Joel Neu, Vitria Technologies
39 Sanjay Patil, IONA
40 Waqar Sadiq, EDS
41 Neal Smith, ChevronTexaco
42 Nikola Stojanovic, Encoda Systems Inc.
43 David Webber, XMLGlobal
44 Prasad Yendluri, webmethods
45 Yutaka Yoshida, Sun Microsystems Inc.

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

91	7.3	The Submit Objects Protocol.....	26
92	7.3.1	Universally Unique ID Generation.....	26
93	7.3.2	ID Attribute And Object References	27
94	7.3.3	Audit Trail	27
95	7.3.4	Submitting Organization.....	27
96	7.3.5	Error Handling	27
97	7.3.6	Sample SubmitObjectsRequest.....	28
98	7.4	The Update Objects Protocol.....	31
99	7.4.1	Audit Trail	32
100	7.4.2	Submitting Organization.....	32
101	7.4.3	Error Handling	32
102	7.5	The Add Slots Protocol.....	32
103	7.6	The Remove Slots Protocol	33
104	7.7	The Approve Objects Protocol.....	33
105	7.7.1	Audit Trail	34
106	7.7.2	Submitting Organization.....	34
107	7.7.3	Error Handling	34
108	7.8	The Deprecate Objects Protocol	35
109	7.8.1	Audit Trail	35
110	7.8.2	Submitting Organization.....	35
111	7.8.3	Error Handling	35
112	7.9	The Remove Objects Protocol.....	36
113	7.9.1	Deletion Scope DeleteRepositoryItemOnly	36
114	7.9.2	Deletion Scope DeleteAll	36
115	7.9.3	Error Handling	37
116	8	Query Management Service.....	38
117	8.1	Ad Hoc Query Request/Response	38
118	8.1.1	Query Response Options	39
119	8.2	Filter Query Support	40
120	8.2.1	FilterQuery.....	42
121	8.2.2	RegistryObjectQuery	43
122	8.2.3	RegistryEntryQuery.....	57
123	8.2.4	AuditableEventQuery	60
124	8.2.5	ClassificationNodeQuery.....	63
125	8.2.6	ClassificationSchemeQuery.....	68
126	8.2.7	RegistryPackageQuery	69
127	8.2.8	ExtrinsicObjectQuery	71
128	8.2.9	OrganizationQuery	72
129	8.2.10	ServiceQuery	76
130	8.2.11	Registry Filters.....	78
131	8.2.12	XML Clause Constraint Representation.....	81
132	8.3	SQL Query Support	86
133	8.3.1	SQL Query Syntax Binding To [ebRIM]	86
134	8.3.2	Semantic Constraints On Query Syntax.....	87
135	8.3.3	SQL Query Results	88
136	8.3.4	Simple Metadata Based Queries	88
137	8.3.5	RegistryObject Queries.....	88
138	8.3.6	RegistryEntry Queries	89
139	8.3.7	Classification Queries	89

140	8.3.8 Association Queries	90
141	8.3.9 Package Queries.....	91
142	8.3.10 ExternalLink Queries	91
143	8.3.11 Audit Trail Queries	91
144	8.4 Content Retrieval.....	91
145	8.4.1 Identification Of Content Payloads	92
146	8.4.2 GetContentResponse Message Structure	92
147	9 Registry Security	93
148	9.1 Security Concerns	93
149	9.2 Integrity of Registry Content	94
150	9.2.1 Message Payload Signature	94
151	9.2.2 Payload Signature Requirements	95
152	9.3 Authentication.....	97
153	9.3.1 Message Header Signature	97
154	9.4 Key Distribution and KeyInfo Element.....	99
155	9.5 Confidentiality.....	100
156	9.5.1 On-the-wire Message Confidentiality.....	100
157	9.5.2 Confidentiality of Registry Content	100
158	9.6 Authorization.....	100
159	9.6.1 Actions	100
160	9.7 Access Control.....	100
161	Appendix A Web Service Architecture	102
162	Registry Service Abstract Specification.....	102
163	Registry Service SOAP Binding.....	102
164	Appendix B ebXML Registry Schema Definitions	103
165	RIM Schema	103
166	Query Schema.....	103
167	Registry Services Interface Schema	103
168	Examples of Instance Documents.....	103
169	Appendix C Interpretation of UML Diagrams	104
170	UML Class Diagram.....	104
171	UML Sequence Diagram	104
172	Appendix D SQL Query	105
173	SQL Query Syntax Specification.....	105
174	Non-Normative BNF for Query Syntax Grammar	105
175	Relational Schema For SQL Queries.....	107
176	Appendix E Non-normative Content Based Ad Hoc Queries.....	108
177	Automatic Classification of XML Content	108
178	Index Definition.....	108
179	Example Of Index Definition	108
180	Proposed XML Definition.....	109
181	Example of Automatic Classification.....	109
182	Appendix F Security Implementation Guideline	110
183	Authentication.....	110
184	Authorization.....	110
185	Registry Bootstrap	110
186	Content Submission – Client Responsibility	110
187	Content Submission – Registry Responsibility	110

188	Content Delete/Deprecate – Client Responsibility	111
189	Content Delete/Deprecate – Registry Responsibility	111
190	Using ds:KeyInfo Field	111
191	Appendix G Native Language Support (NLS)	113
192	Definitions	113
193	Coded Character Set (CCS):.....	113
194	Character Encoding Scheme (CES):.....	113
195	Character Set (charset):.....	113
196	NLS And Request / Response Messages	113
197	NLS And Storing of RegistryObject	113
198	Character Set of <i>LocalizedString</i>	114
199	Language Information of <i>LocalizedString</i>	114
200	NLS And Storing of Repository Items	114
201	Character Set of Repository Items	114
202	Language information of repository item.....	114
203	Appendix H Terminology Mapping	115
204	References.....	116
205	Disclaimer.....	117
206	Contact Information.....	118
207	Copyright Statement	119
208		

209 **Table of Figures**

210	☞ Figure 1: Actor Relationships	15
211	☞ Figure 2: ebXML Registry Service Architecture	17
212	☞ Figure 3: The Abstract ebXML Registry Service	18
213	☞ Figure 4: A Concrete ebXML Registry Service	18
214	☞ Figure 5: Registry Architecture Supports Flexible Topologies	22
215	☞ Figure 6: Life Cycle of a Repository Item.....	25
216	☞ Figure 7: Submit Objects Sequence Diagram.....	26
217	☞ Figure 8: Update Objects Sequence Diagram.....	31
218	☞ Figure 9: Add Slots Sequence Diagram.....	33
219	☞ Figure 10: Remove Slots Sequence Diagram	33
220	☞ Figure 11: Approve Objects Sequence Diagram	34
221	☞ Figure 12: Deprecate Objects Sequence Diagram	35
222	☞ Figure 13: Remove Objects Sequence Diagram	37
223	☞ Figure 14: Submit Ad Hoc Query Sequence Diagram.....	39
224	☞ Figure 15: Example ebRIM Binding.....	41
225	☞ Figure 16: ebRim Binding for RegistryObjectQuery.....	44
226	☞ Figure 17: ebRIM Binding for RegistryEntryQuery.....	57
227	☞ Figure 18: ebRim binding for AuditableEventQuery	60
228	☞ Figure 19: ebRim binding for ClassificationNodeQuery.....	64
229	☞ Figure 20: ebRIM Binding for ClassificationSchemeQuery	68
230	☞ Figure 21: ebRim binding for RegistryPackageQuery.....	69
231	☞ Figure 22: ebRIM Binding for ExtrinsicObjectQuery	72
232	☞ Figure 23: ebRim Binding for OrganizationQuery.....	73
233	☞ Figure 24: ebRIM Binding for ServiceQuery	77
234	☞ Figure 25: The Clause Structure	82
235		

236 Table of Tables

237	☞ Table 1: Registry Users.....	14
238	☞ Table 2: LifeCycle Manager Summary.....	21
239	☞ Table 3: Query Manager	21
240	☞ Table 4: RegistryClient Summary	24
241	☞ Table 5 Submit Objects Error Handling	27
242	☞ Table 6: Update Objects Error Handling	32
243	☞ Table 7: Approve Objects Error Handling.....	34
244	☞ Table 8: Deprecate Objects Error Handling.....	36
245	☞ Table 9: Remove Objects Error Handling.....	37
246	☞ Table 10: Path Filter Expressions for Use Cases	66
247	☞ Table 11: Default Access Control Policies	101
248	☞ Table 12: Terminology Mapping Table	115
249		

250 **3 Introduction**

251 **3.1 Summary of Contents of Document**

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

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

257 **3.2 General Conventions**

258 The following conventions are used throughout this document:

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

261 The term "*repository item*" is used to refer to an object that has resides in a repository for storage
262 and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the
263 Registry by a RegistryObject instance.

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

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

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

270 **3.3 Audience**

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

272 ?? Implementers of ebXML Registry Services

273 ?? Implementers of ebXML Registry Clients

274 **3.3.1.1.1 Related Documents**

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

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

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

278 c) *ebXML Business Process Specification Schema* [ebBPM]

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

280 **4 Design Objectives**

281 **4.1 Goals**

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

283 ?? Communicate functionality of Registry services to software developers

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

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

286 ?? Be compatible with other ebXML specifications

287 **4.2 Caveats and Assumptions**

288 The Registry Services specification is first in a series of phased deliverables. Later versions of
289 the document will include additional functionality planned for future development. It is

290 assumed that:

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

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

298 **5 System Overview**

299 **5.1 What The ebXML Registry Does**

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

304 **5.2 How The ebXML Registry Works**

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

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

314 **5.2.1 Schema Documents Are Submitted**

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

318 **5.2.2 Business Process Documents Are Submitted**

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

322 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

327 **5.2.4 Buyer Discovers The Seller**

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

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

333 **5.2.5 CPA Is Established**

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

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

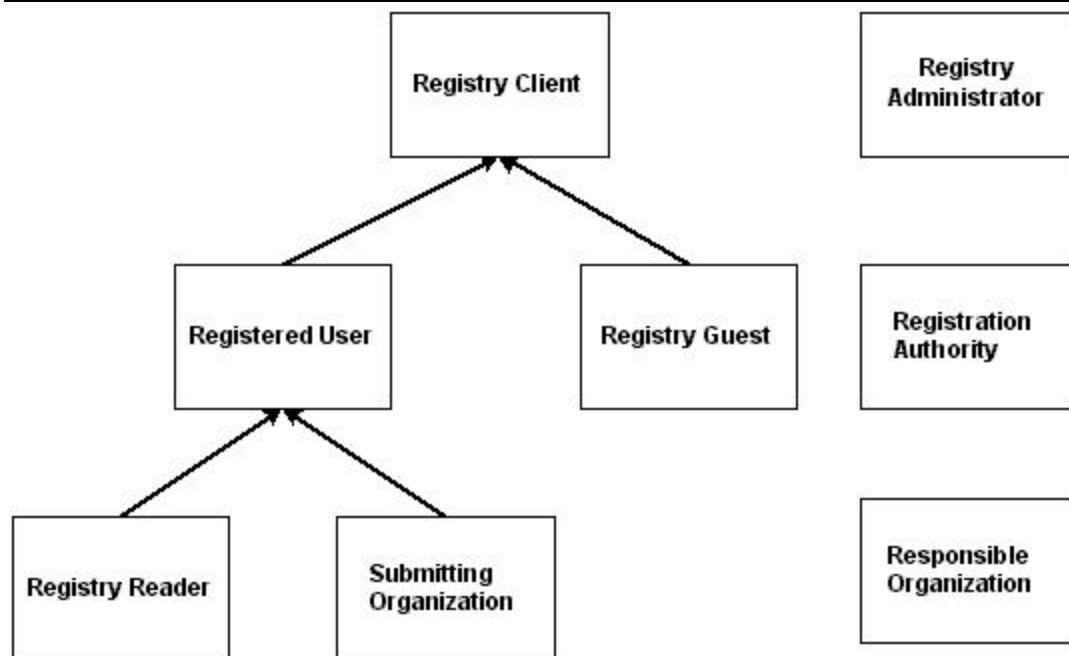
340 **5.3 Registry Users**

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

346

☞☞ **Table 1: Registry Users**

Actor	Function	ISO/IEC 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY have the same identity as Registration Authority
Registered User	Has a <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		



347
348

☞☞ Figure 1: Actor Relationships

349 **Note:**

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

351 ?? A Submitting Organization and a Responsible Organization are the same.

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

353 ?? A Registry Administrator and Registration Authority are the same.

354 **5.4 Where the Registry Services May Be Implemented**

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

357 **5.5 Implementation Conformance**

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

365 **5.5.1 Conformance as an ebXML Registry**

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

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

372 5.5.2 Conformance as an ebXML Registry Client

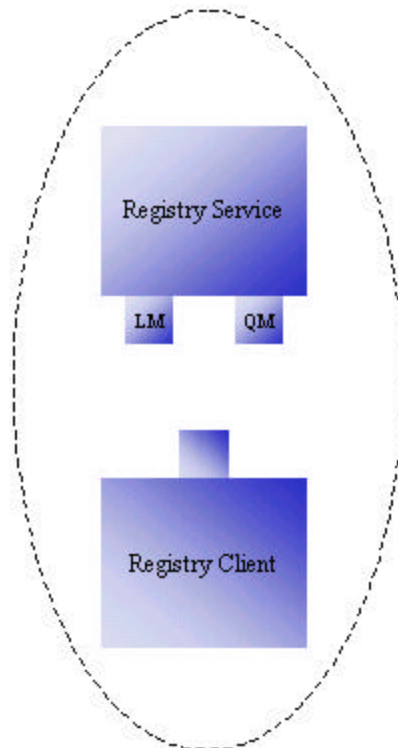
373 An implementation conforms to this specification, as an ebXML Registry Client if it meets the
374 following conditions:

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

379

380 **6 ebXML Registry Architecture**

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



384
385

☞☞ **Figure 2: ebXML Registry Service Architecture**

386 **6.1 Registry Service Described**

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

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

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

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

398 **6.2 Abstract Registry Service**

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

- 400 1. A set of interfaces that must be supported by the registry.
- 401 2. The set of methods that must be supported by each interface.

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



409
410 ☞☞ **Figure 3: The Abstract ebXML Registry Service**

411 Appendix 0 describes the abstract service definition in the Web Service Description Language
 412 (WSDL) syntax.

413 **6.3 Concrete Registry Services**

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

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

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

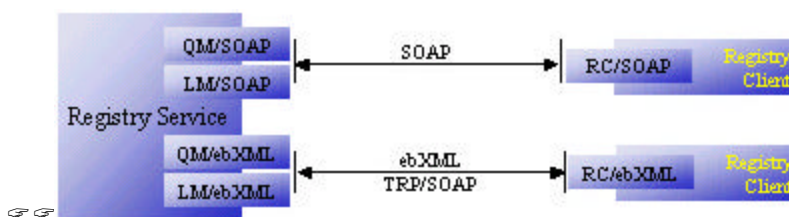
418 This specification describes two concrete bindings for the abstract registry service:

419 ?? A SOAP binding using the HTTP protocol

420 ?? An ebXML Messaging Service (ebMS) binding

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

423



424
425 ☞☞ **Figure 4: A Concrete ebXML Registry Service**

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

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

¹ Known as ObjectQueryManager in V1.0

² Known as ObjectManager in V1.0

432 protocol preference.

433 6.3.1 SOAP Binding

434 6.3.1.1 WSDL Terminology Primer

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

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

447 6.3.1.2 Concrete Binding for SOAP

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

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

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

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

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

466
467
468 The complete WSDL description for the SOAP binding is described in Appendix 0

469 6.3.2 ebXML Message Service Binding

470 6.3.2.1 Service and Action Elements

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

473 ?? The value of the Service element in the MessageHeader is an ebXML Registry Service
474 interface name (e.g., “LifeCycleManager”). The type attribute of the Service element should
475 have a value of “ebXMLRegistry”.

476 ?? The value of the Action element in the MessageHeader is an ebXML Registry Service
477 method name (e.g., “submitObjects”).
478

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

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

485 **6.3.2.2 Synchronous and Asynchronous Responses**

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

487 **Asynchronous response**

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

491 ?? MessageHeader;

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

493 ?? status attribute with value **Unavailable**.

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

499 ?? MessageHeader;

500 ?? RegistryResponse element including;

501 ?? Status attribute (Success, Failure);

502 ?? Optional RegistryErrorList.

503 **Synchronous response**

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

506 ?? MessageHeader;

507 ?? RegistryResponse element including;

508 ?? Status attribute (Success, Failure);

509 ?? Optional RegistryErrorList.

510 **6.3.2.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

519 The CPA between clients and the Registry should describe the interfaces that the Registry and

520 the client expose to each other for Registry-specific interactions. The definition of the Registry
 521 CPP template and a Registry Client CPP template are beyond the scope of this document.

522 6.4 LifeCycleManager Interface

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

528 ☞☞

529 ☞☞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.

530 6.5 QueryManager Interface

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

534 ☞☞

535 ☞☞Table 3: Query Manager

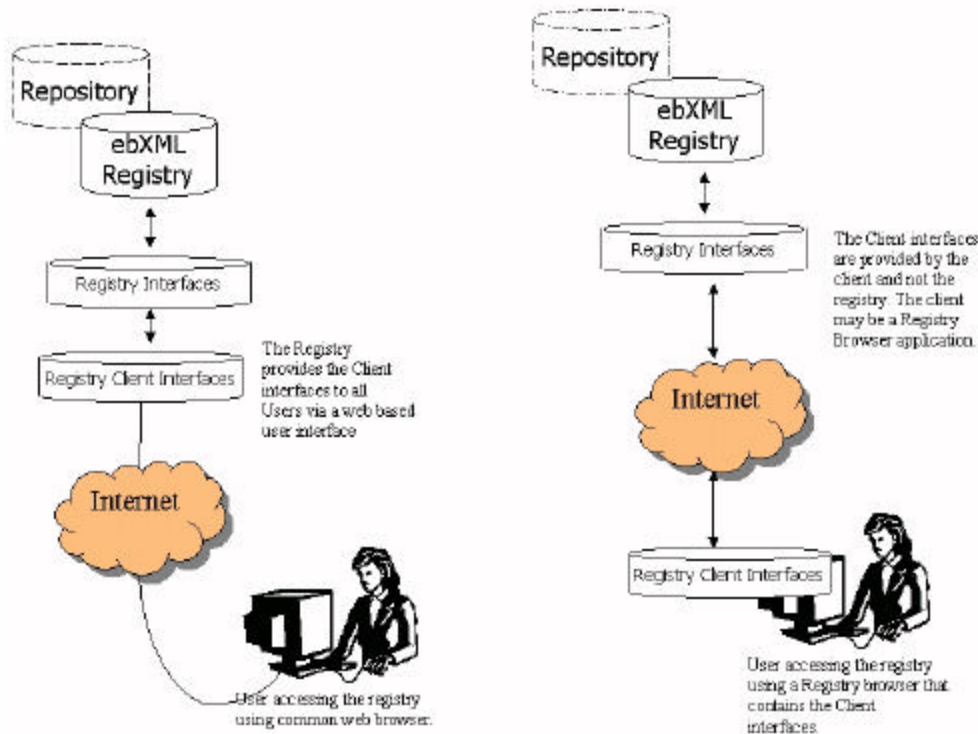
Method Summary of QueryManager	
RegistryResponse	submitAdhocQuery (AdhocQueryRequest req) Submit an ad hoc query request.

536 **6.6 Registry Clients**

537 **6.6.1 Registry Client Described**

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

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



553
 554

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

555 **6.6.2 Registry Communication Bootstrapping**

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

562 ?? In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available
563 in a WSDL description for the registry. This WSDL conforms to the template WSDL
564 description in Appendix 0. This WSDL description may be discovered in a public registry
565 such as UDDI.

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

570 **6.6.2.1 Communication Bootstrapping for SOAP Binding**

571 Each ebXML Registry must provide a WSDL description for its RegistryService as defined by
572 Appendix 0. A client uses the WSDL description to determine the address information of the
573 RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the
574 RegistryService may be accessed via a URL specified in the WSDL for the registry.

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

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

580 **6.6.2.2 Communication Bootstrapping for ebXML Message Service**

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

591 **6.6.3 RegistryClient Interface**

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

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

598

599

☞☞Table 4: RegistryClient Summary

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

600 6.6.4 Registry Response

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

603 6.7 Interoperability Requirements

604 6.7.1 Client Interoperability

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

609 6.7.2 Inter-Registry Cooperation

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

613 Examples include:

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

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

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

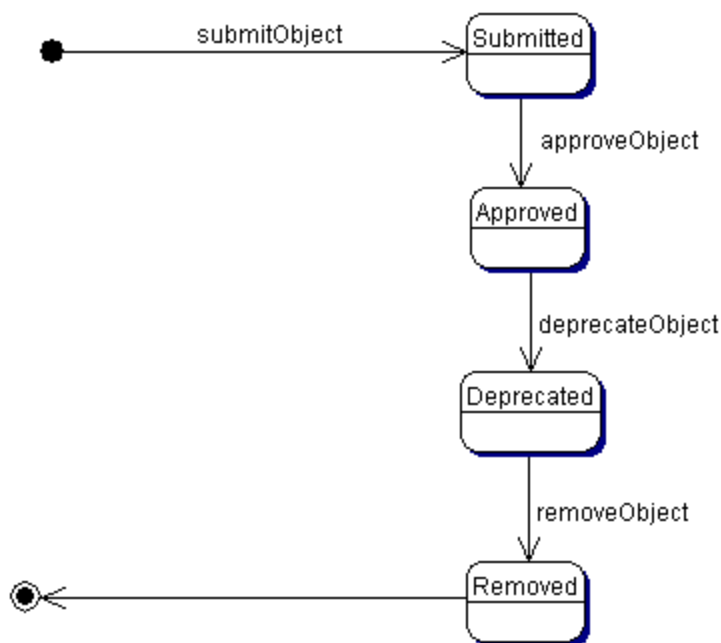
618 7 Life Cycle Management Service

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

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

628 7.1 Life Cycle of a Repository Item

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



633
634

Figure 6: Life Cycle of a Repository Item

635 7.2 RegistryObject Attributes

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

641 7.3 The Submit Objects Protocol

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



645
 646

Figure 7: Submit Objects Sequence Diagram

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

648 The SubmitObjectRequest message includes a LeafRegistryObjectList element.

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

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

655 7.3.1 Universally Unique ID Generation

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

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

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

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

669 **7.3.2 ID Attribute And Object References**

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

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

685 **7.3.3 Audit Trail**

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

688 **7.3.4 Submitting Organization**

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

692 **7.3.5 Error Handling**

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

☞☞ **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	All Classes	Warning

minorVersion are set by the RS, and ignored if supplied.		
--	--	--

701 7.3.6 Sample SubmitObjectsRequest

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

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

```

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

```

766 <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
767 sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
768
769 <!--
770 The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
771 where the RegistryPackage and the ExtrinsicObject are already in registry
772 -->
773
774 <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
775 <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
776
777 <!-- id is unspecified implying that registry must create a uuid for this object -->
778
779 <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
780 123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
781
782 <!--
783 The following 3 objects externally link specified ExtrinsicObject using
784 specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
785 are being submitted
786 -->
787
788 <rim:ExternalLink id = "acmeLink1" >
789 <rim:Name>
790 <rim:LocalizedString value = "Link #1"/>
791 </rim:Name>
792 <rim:Description>
793 <rim:LocalizedString value = "ACME's Link #1"/>
794 </rim:Description>
795 </rim:ExternalLink>
796
797 <rim:ExtrinsicObject id = "acmeCPP2" >
798 <rim:Name>
799 <rim:LocalizedString value = "Sprockets Profile" />
800 </rim:Name>
801 <rim:Description>
802 <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
803 </rim:Description>
804 </rim:ExtrinsicObject>
805
806 <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
807 sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
808
809 <!--
810 The following 2 objects externally link specified ExtrinsicObject using specified
811 ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
812 is already in registry. Note that the targetObject points to an ObjectRef in a
813 previous line
814 -->
815
816 <rim:ExternalLink id = "acmeLink2">
817 <rim:Name>
818 <rim:LocalizedString value = "Link #2"/>
819 </rim:Name>
820 <rim:Description>
821 <rim:LocalizedString value = "ACME's Link #2"/>
822 </rim:Description>
823 </rim:ExternalLink>
824
825 <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
826 "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
827 123456789012"/>
828
829 <!--
830 The following 3 objects externally identify specified ExtrinsicObject using specified
831 ExternalIdentifier, where the ExternalIdentifier is being submitted and the
832 ExtrinsicObject is already in registry. Note that the targetObject points to an
833 ObjectRef in a previous line
834 -->
835
836 <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
837 <rim:Name>
838 <rim:LocalizedString value = "DUNS"/>
839 </rim:Name>
840

```

```

841     <rim:Description>
842       <rim:LocalizedString value = "This is the DUNS scheme"/>
843     </rim:Description>
844   </rim:ClassificationScheme>
845
846   <rim:ExternalIdentifier id = "acmeDUNSId"  identificationScheme="DUNS-id" value =
847 "13456789012">
848     <rim:Name>
849       <rim:LocalizedString value = "DUNS" />
850     </rim:Name>
851     <rim:Description>
852       <rim:LocalizedString value = "DUNS ID for ACME"/>
853     </rim:Description>
854   </rim:ExternalIdentifier>
855
856   <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType =
857 "ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
858 123456789012"/>
859
860   <!--
861     The following show submission of a brand new classification scheme in its entirety
862     -->
863   <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
864     <rim:Name>
865       <rim:LocalizedString value = "Geography"/>
866     </rim:Name>
867
868     <rim:Description>
869       <rim:LocalizedString value = "This is a sample Geography scheme"/>
870     </rim:Description>
871
872     <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
873 "NorthAmerica" >
874       <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
875 "UnitedStates" />
876       <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
877     </rim:ClassificationNode>
878
879     <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
880       <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
881         <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
882       </rim:ClassificationNode>
883     </rim:ClassificationNode>
884   </rim:ClassificationScheme>
885
886   <!--
887     The following show submission of a Automotive sub-tree of ClassificationNodes that
888     gets added to an existing classification scheme named 'Industry'
889     that is already in the registry
890     -->
891
892   <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
893   <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
894 123456789012">
895     <rim:Name>
896       <rim:LocalizedString value = "Automotive" />
897     </rim:Name>
898     <rim:Description>
899       <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
900     </rim:Description>
901   </rim:ClassificationNode>
902
903   <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
904     <rim:Name>
905       <rim:LocalizedString value = "Parts Supplier" />
906     </rim:Name>
907     <rim:Description>
908       <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
909     </rim:Description>
910   </rim:ClassificationNode>
911
912   <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
913     <rim:Name>
914       <rim:LocalizedString value = "Engine Supplier" />
915

```

916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946

```

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

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

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

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

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

947 **7.4 The Update Objects Protocol**

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



951
952

☞ **Figure 8: Update Objects Sequence Diagram**

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

958 to NULL.

959 **7.4.1 Audit Trail**


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

962 **7.4.2 Submitting Organization**

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

969 **7.4.3 Error Handling**

970 An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success,
971 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
972 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
973 the event of an immediate response for an asynchronous request, the registry sends a
974 RegistryResponse with a status of "Unavailable" back to the client. Failure occurs when one or
975 more Error conditions are raised in the processing of the updated objects. Warning messages do
976 not result in failure of the request. The following business rules 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
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 LifecycleManager. Slots provide a dynamic
981 mechanism for extending registry entries as defined by [ebRIM].



982
983

Figure 9: 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 LifeCycleManager.



990
991

Figure 10: 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 LifeCycleManager. Once a repository item
995 is approved it will become available for use by business parties (e.g. during the assembly of new
996 CPAs and Collaboration Protocol Profiles).



997
998

Figure 11: Approve Objects Sequence Diagram

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 SubmitterOf 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. In
1014 the event of an immediate response for an asynchronous request, the registry sends a
1015 RegistryResponse with a status of “Unavailable” back to the client. Failure occurs when one or
1016 more Error conditions are raised in the processing of the object reference list. Warning messages
1017 do not result in failure of the request. The following business rules apply:

1018 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	All Classes other	Error

"approved".	than RegistryEntry classes	
Object status is already "Approved".	RegistryEntry Classes	Warning

1019 **7.8 The Deprecate Objects Protocol**

1020 This section describes the protocol of the Registry Service that allows a client to deprecate one or
 1021 more previously submitted repository items using the LifeCycleManager. Once an object is
 1022 deprecated, no new references (e.g. *new* Associations, Classifications and ExternalLinks) to that
 1023 object can be submitted. However, existing references to a deprecated object continue to function
 1024 normally.



1025
 1026 **Figure 12: Deprecate Objects Sequence Diagram**

1027 For details on the schema for the business documents shown in this process refer to Appendix B.

1028 **7.8.1 Audit Trail**

1029 The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject
 1030 deprecated via a Deprecate Objects request.


1031 **7.8.2 Submitting Organization**

1032 The RS must maintain an Association of type SubmitterOf between the submitting organization
 1033 and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request
 1034 is accepted from a different submitting organization, then the RS must delete the original
 1035 association object and create a new one. Of course, the AccessControlPolicy may prohibit this
 1036 sort of Deprecate Objects request in the first place. (Submitting organization is determined from
 1037 the organization attribute of the User who submits a Deprecate Objects request.)

1038 **7.8.3 Error Handling**

1039 A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of
 1040 success, the registry sends a RegistryResponse with a status of "Success" back to the client. In
 1041 the event of failure, the registry sends a RegistryResponse with a status of "Failure" back to the
 1042 client. In the event of an immediate response for an asynchronous request, the registry sends a

1043 RegistryResponse with a status of “Uavailable” back to the client. Failure occurs when one or
 1044 more Error conditions are raised in the processing of the object reference list. Warning messages
 1045 do not result in failure of the request. The following business rules apply:

1046  **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

1047 **7.9 The Remove Objects Protocol**

1048 This section describes the protocol of the Registry Service that allows a client to remove one or
 1049 more RegistryObject instances and/or repository items using the LifeCycleManager.

1050 The RemoveObjectsRequest message is sent by a client to remove RegistryObject instances
 1051 and/or repository items. The RemoveObjectsRequest element includes an XML attribute called
 1052 *deletionScope* which is an enumeration that can have the values as defined by the following
 1053 sections.

1054 **7.9.1 Deletion Scope DeleteRepositoryItemOnly**

1055 This deletionScope specifies that the request should delete the repository items for the specified
 1056 registry entries but not delete the specified registry entries. This is useful in keeping references to
 1057 the registry entries valid.

1058 **7.9.2 Deletion Scope DeleteAll**

1059 This deletionScope specifies that the request should delete both the RegistryObject and the
 1060 repository item for the specified registry entries. Only if all references (e.g. Associations,
 1061 Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject
 1062 then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to
 1063 remove a RegistryObject while it still has references raises an error condition:

1064 InvalidRequestError.

1065 The remove object protocol is expressed in UML notation as described in Appendix C.



1066
1067

☞☞Figure 13: Remove Objects Sequence Diagram

1068 For details on the schema for the business documents shown in this process refer to Appendix B.

1069 **7.9.3 Error Handling**

1070 A Remove Objects request is atomic and either succeeds or fails in total. In the event of success,
1071 the registry sends a RegistryResponse with a status of “Success” back to the client. In the event
1072 of failure, the registry sends a RegistryResponse with a status of “Failure” back to the client. In
1073 the event of an immediate response for an asynchronous request, the registry sends a
1074 RegistryResponse with a status of “Unavailable” back to the client. Failure occurs when one or
1075 more Error conditions are raised in the processing of the object reference list. Warning messages
1076 do not result in failure of the request. The following business rules apply:

1077

☞☞Table 9: Remove Objects Error Handling

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

1078

1079 **8 Query Management Service**

1080 This section describes the capabilities of the Registry Service that allow a client
1081 (QueryManagerClient) to search for or query different kind of registry objects in the ebXML
1082 Registry using the QueryManager interface of the Registry. The Registry supports the following
1083 query capabilities:

1084 1. Filter Query

1085 2. SQL Query

1086 The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry
1087 implementation. The SQL Query mechanism is an optional feature and MAY be provided by a
1088 registry implementation. However, if a vendor provides an SQL query capability to an ebXML
1089 Registry it SHALL conform to this document. As such this capability is a normative yet optional
1090 capability.

1091 In a future version of this specification, the W3C XQuery syntax may be considered as another
1092 query syntax.

1093 The Registry will hold a self-describing **capability profile** that identifies all supported
1094 AdhocQuery options. This profile is described in Section **Error! Reference source not found.**

1095 **8.1 Ad Hoc Query Request/Response**

1096 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
1097 AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry
1098 query mechanisms.

1099 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously
1100 back to the client. The AdhocQueryResponse returns a collection of objects whose element type
1101 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects
1102 representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate
1103 classes in [ebRIM] such as RegistryObject and RegistryEntry.

1104 Any errors in the query request messages are indicated in the corresponding query response
1105 message.



1106

1107

Figure 14: Submit Ad Hoc Query Sequence Diagram

1108 For details on the schema for the business documents shown in this process refer to Appendix 0.

1109 Definition

```

1110
1111 <element name="AdhocQueryRequest">
1112   <complexType>
1113     <sequence>
1114       <element ref="tns:ResponseOption" minOccurs="1" maxOccurs="1" />
1115       <choice minOccurs="1" maxOccurs="1">
1116         <element ref="tns:FilterQuery" />
1117         <element ref="tns:SQLQuery" />
1118       </choice>
1119     </sequence>
1120   </complexType>
1121 </element>
1122
1123 <element name="AdhocQueryResponse">
1124   <complexType>
1125     <choice minOccurs="1" maxOccurs="1">
1126       <element ref="tns:FilterQueryResult" />
1127       <element ref="tns:SQLQueryResult" />
1128     </choice>
1129   </complexType>
1130 </element>
1131

```

1132 8.1.1 Query Response Options

1133 Purpose

1134 A QueryManagerClient may specify what an ad hoc query must return within an
1135 AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest.

1136 ResponseOption element has an attribute "returnType" and its values are:

1137 ?? ObjectRef - This option specifies that the AdhocQueryResponse must contain a collection of
1138 ObjectRef XML elements as defined in [RIM schema]. Purpose of this option is to return just
1139 the identifiers of the registry objects.

- 1140 ?? RegistryObject - This option specifies that the AdhocQueryResponse must contain a
 1141 collection of RegistryObject XML elements as defined in [RIM schema]. In this case all
 1142 attributes of the registry objects are returned (objectType, name, description, ...) in addition
 1143 to id attribute.
- 1144 ?? RegistryEntry - This option specifies that the AdhocQueryResponse must contain a
 1145 collection of RegistryEntry XML elements as defined in [RIM schema], which correspond to
 1146 RegistryEntry attributes.
- 1147 ?? LeafClass - This option specifies that the AdhocQueryResponse must contain a collection of
 1148 XML elements that correspond to leaf classes as defined in [RIM schema].
- 1149 ?? LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse must
 1150 contain a collection of ExtrinsicObject XML elements as defined in [RIM schema]
 1151 accompanied with their repository items. Linking of ExtrinsicObject and its repository item
 1152 is done via contentURI as explained in [XXX – Content Retrieval section].
- 1153 ResponseOption element also has an attribute “returnComposedObjects”. It specifies whether or
 1154 not the whole hierarchy of composed objects are returned with the registry objects.
- 1155 If “returnType” is higher then the RegistryObject option, then the highest option that satisfies the
 1156 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return
 1157 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass
 1158 option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then
 1159 RegistryObject metadata will be returned.

1160 Definition

```

1161
1162 <complexType name="ResponseOptionType">
1163   <attribute name="returnType" default="RegistryObject">
1164     <simpleType>
1165       <restriction base="NMTOKEN">
1166         <enumeration value="ObjectRef" />
1167         <enumeration value="RegistryObject" />
1168         <enumeration value="RegistryEntry" />
1169         <enumeration value="LeafClass" />
1170         <enumeration value="LeafClassWithRepositoryItem" />
1171       </restriction>
1172     </simpleType>
1173   </attribute>
1174   <attribute name="returnComposedObjects" type="boolean" default="false" />
1175 </complexType>
1176 <element name="ResponseOption" type="tns:ResponseOptionType" />
1177

```

1178 8.2 Filter Query Support

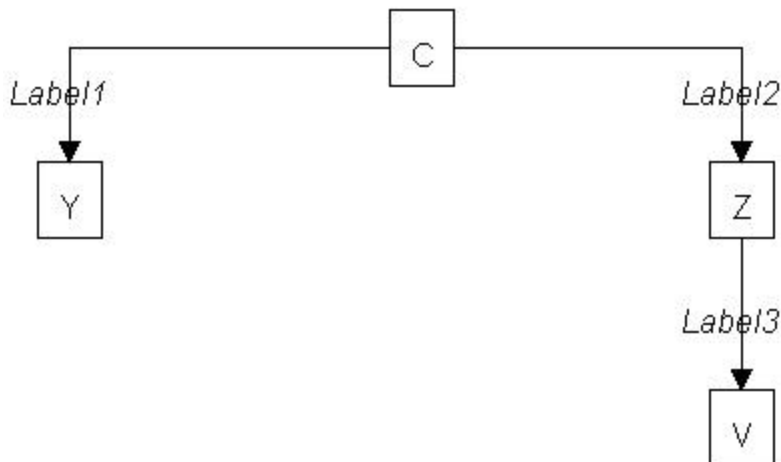
1179 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
 1180 conforming Registry implementation. Each query alternative is directed against a single class
 1181 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter queries
 1182 depending on which classes are queried on.

1183 ?? Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic
 1184 queries that might return different subclasses of the class that is queried on. The result of
 1185 such a query is a set of XML elements that correspond to instances of any class that satisfies
 1186 the responseOption defined previously in [Section 8.1.1](#). An example might be that
 1187 RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances
 1188 that satisfy the query. This implies that response might return XML elements that correspond
 1189 to classes like ClassificationScheme, RegistryPackage, Organization and Service.

1190 ?? Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact
 1191 traversals of these classes. Responses to these queries are accordingly constrained.

1192 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an
 1193 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified
 1194 herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified
 1195 in [Section 8.1](#).

1196 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of
 1197 classes derived from a single class and its associations with other classes as defined by ebRIM.
 1198 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For
 1199 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a
 1200 class that is associated with Z. The ebRIM Binding for C might be as in Figure 15



1201
 1202

☞ **Figure 15: Example ebRIM Binding**

1203 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and
 1204 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to
 1205 which ebRIM association is intended. The name of the query is determined by the root class, i.e.
 1206 this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances
 1207 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are
 1208 limited to instances that are linked to their parent node by the identified association.

1209 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a
 1210 restricted *predicate clause* over the attributes of a single class. Class methods that are defined in
 1211 ebRIM and that return simple types constitute “visible attributes” that are valid choices for
 1212 predicate clauses. Names of those attributes will be same as name of the corresponding method
 1213 just without the prefix ‘get’. For example, in case of “getLevelNumber” method the
 1214 corresponding visible attribute is “levelNumber”. The supported class filters are specified in
 1215 [Section 8.2.11](#) and the supported predicate clauses are defined in [Section 8.2.12](#). A FilterQuery
 1216 will be composed of elements that traverse the tree to determine which branches satisfy the
 1217 designated class filters, and the query result will be the set of instances that support such a
 1218 branch.

1219 In the above example, the CQuery element will have three subelements, one a CFilter on the C
1220 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on
1221 the Y class to eliminate branches from C to Y where the target of the association does not satisfy
1222 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third
1223 element is called a *branch* element because it allows class filters on each class along the path
1224 from C to V. In general, a branch element will have subelements that are themselves class filters,
1225 other branch elements, or a full-blown query on the class in the path.

1226 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one
1227 branch, filter or query element on Y is allowed. However, if the association is one-to-many, then
1228 multiple branch, filter or query elements are allowed. This allows one to specify that an instance
1229 of C must have associations with multiple instances of Y before the instance of C is said to
1230 satisfy the branch element.

1231 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be
1232 stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then
1233 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax
1234 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their
1235 respective superclass queries. Structures of XML elements that match the ebRIM classes are
1236 explained in [RIM Schema]. Names of Filters, Queries and Branches correspond to names in
1237 ebRIM whenever possible.

1238 **The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.10 below identify the virtual**
1239 **hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative**
1240 **specify the effect of that binding on query semantics.**

1241 8.2.1 FilterQuery

1242 Purpose

1243 To identify a set of queries that traverse specific registry class. Each alternative assumes a
1244 specific binding to ebRIM. The status is a success indication or a collection of warnings and/or
1245 exceptions.

1246 Definition

```
1247 <element name="FilterQuery">
1248   <complexType>
1249     <choice minOccurs="1" maxOccurs="1">
1250       <element ref="tns:RegistryObjectQuery" />
1251       <element ref="tns:RegistryEntryQuery" />
1252       <element ref="tns:AuditableEventQuery" />
1253       <element ref="tns:ClassificationNodeQuery" />
1254       <element ref="tns:ClassificationSchemeQuery" />
1255       <element ref="tns:RegistryPackageQuery" />
1256       <element ref="tns:ExtrinsicObjectQuery" />
1257       <element ref="tns:OrganizationQuery" />
1258       <element ref="tns:ServiceQuery" />
1259     </choice>
1260   </complexType>
1261 </element>
1262
1263 <element name="FilterQueryResult">
1264   <complexType>
1265     <choice minOccurs="1" maxOccurs="1">
1266       <element ref="tns:RegistryObjectQueryResult" />
1267       <element ref="tns:RegistryEntryQueryResult" />
1268     </choice>
1269   </complexType>
1270 </element>
```

```
1269     <element ref="tns:AuditableEventQueryResult" />
1270     <element ref="tns:ClassificationNodeQueryResult" />
1271     <element ref="tns:ClassificationSchemeQueryResult" />
1272     <element ref="tns:RegistryPackageQueryResult" />
1273     <element ref="tns:ExtrinsicObjectQueryResult" />
1274     <element ref="tns:OrganizationQueryResult" />
1275     <element ref="tns:ServiceQueryResult" />
1276   </choice>
1277 </complexType>
1278 </element>
1279
```

1280 **Semantic Rules**

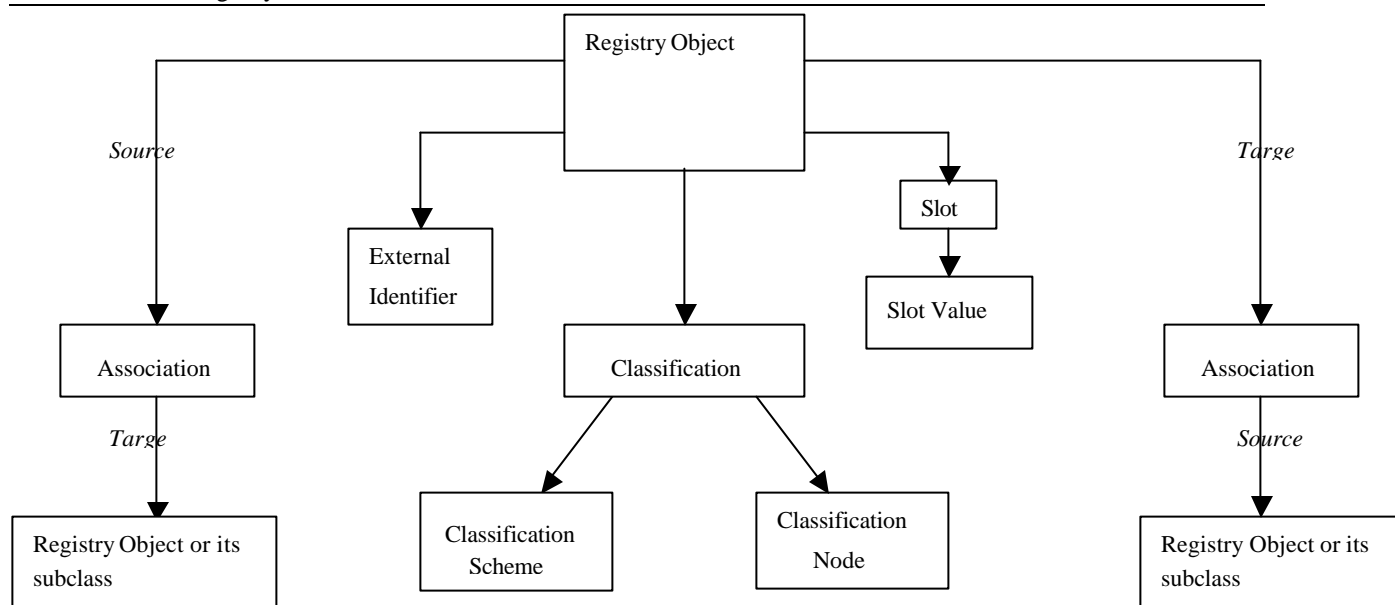
- 1281 3. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 1282 4. Semantic rules specify the procedure for implementing the evaluation of Filter Queries.
1283 Implementations do not necessarily have to follow the same procedure provided that the
1284 same effect is achieved.
- 1285 5. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
1286 Each XML attribute carries a value derived from the value of an attribute specified in the
1287 Registry Information Model [RIM Schema].
- 1288 6. For each FilterQuery subelement there is only one corresponding FilterQueryResult
1289 subelement that must be returned as a response. Class name of the FilterQueryResult
1290 subelement has to match the class name of the FilterQuery subelement.
- 1291 7. If an error condition is raised during any part of the execution of a FilterQuery, then the
1292 status attribute of the XML RegistryResult is set to “failure” and no query result element is
1293 returned; instead, a RegistryErrorList element must be returned with its highestSeverity
1294 element set to “error”. At least one of the RegistryError elements in the RegistryErrorList
1295 will have its severity attribute set to “error”.
- 1296 8. If no error conditions are raised during execution of a FilterQuery, then the status attribute of
1297 the XML RegistryResult is set to “success” and an appropriate query result element must be
1298 included. If a RegistryErrorList is also returned, then the highestSeverity attribute of the
1299 RegistryErrorList is set to “warning” and the serverity attribute of each RegistryError is set to
1300 “warning”.

1301 **8.2.2 RegistryObjectQuery**

1302 **Purpose**

1303 To identify a set of registry object instances as the result of a query over selected registry
1304 metadata.

1305 **ebRIM Binding**



1306

Figure 16: ebRim Binding for RegistryObjectQuery

1307

Definition

1308

```
<complexType name="RegistryObjectQueryType">
```

1309

```
<sequence>
```

1310

```
<element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
```

1311

```
<element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
```

1312

```
<element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
```

1313

```
<element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
```

1314

```
<element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
```

1315

```
<element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
```

1316

```
<element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
```

1317

```
<element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
```

1318

```
<element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
```

1319

```
</sequence>
```

1320

```
</complexType>
```

1321

```
<element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
```

1322

1323

```
<complexType name="LeafRegistryObjectListType">
```

1324

```
<choice minOccurs="0" maxOccurs="unbounded">
```

1325

```
<element ref="tns:ObjectRef" />
```

1326

```
<element ref="tns:Association" />
```

1327

```
<element ref="tns:AuditableEvent" />
```

1328

```
<element ref="tns:Classification" />
```

1329

```
<element ref="tns:ClassificationNode" />
```

1330

```
<element ref="tns:ClassificationScheme" />
```

1331

```
<element ref="tns:ExternalIdentifier" />
```

1332

```
<element ref="tns:ExternalLink" />
```

1333

```
<element ref="tns:ExtrinsicObject" />
```

1334

```
<element ref="tns:Organization" />
```

1335

```
<element ref="tns:RegistryPackage" />
```

1336

```
<element ref="tns:Service" />
```

1337

```
<element ref="tns:ServiceBinding" />
```

1338

```
<element ref="tns:SpecificationLink" />
```

1339

```
<element ref="tns:User" />
```

1340

```
</choice>
```

1341

```
</complexType>
```

1342

1343

```
<complexType name="RegistryObjectListType">
```

```
1344 <complexContent>
1345 <extension base="tns:LeafRegistryObjectListType">
1346 <choice minOccurs="0" maxOccurs="unbounded">
1347 <element ref="tns:RegistryEntry" />
1348 <element ref="tns:RegistryObject" />
1349 </choice>
1350 </extension>
1351 </complexContent>
1352 </complexType>
1353 <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1354
1355 <complexType name="InternationalStringBranchType">
1356 <sequence>
1357 <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1358 </sequence>
1359 </complexType>
1360
1361 <complexType name="AssociationBranchType">
1362 <sequence>
1363 <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1364 <choice minOccurs="0" maxOccurs="1">
1365 <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1366 <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1367 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1368 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1369 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1370 <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1371 <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1372 <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1373 <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1374 <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1375 <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1376 <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
1377 <element ref="tns:ClassificationBranch" minOccurs="0" maxOccurs="1" />
1378 <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1379 <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1380 <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="1" />
1381 <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="1" />
1382 </choice>
1383 </sequence>
1384 </complexType>
1385 <element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
1386 <element name="TargetAssociationBranch" type="tns:AssociationBranchType" />
1387
1388 <element name="ClassifiedByBranch">
1389 <complexType>
1390 <sequence>
1391 <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
1392 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1393 <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1394 </sequence>
1395 </complexType>
1396 </element>
1397
1398 <element name="ClassificationBranch">
1399 <complexType>
1400 <sequence>
1401 <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
1402 <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
```

```

1403     <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1404     <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1405     <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1406   </sequence>
1407 </complexType>
1408 </element>
1409
1410 <element name="SlotBranch">
1411   <complexType>
1412     <sequence>
1413       <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
1414       <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
1415     </sequence>
1416   </complexType>
1417 </element>
1418
1419 <element name="UserBranch">
1420   <complexType>
1421     <sequence>
1422       <element ref="tns:UserFilter" minOccurs="0" maxOccurs="1"/>
1423       <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1"/>
1424       <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded"/>
1425       <element ref="tns:EmailAddressFilter" minOccurs="0" maxOccurs="unbounded"/>
1426       <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1"/>
1427     </sequence>
1428   </complexType>
1429 </element>
1430
1431 <complexType name="ServiceBindingBranchType">
1432   <sequence>
1433     <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
1434     <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
1435     <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
1436   </sequence>
1437 </complexType>
1438 <element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType" />
1439 <element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType" />
1440
1441 <element name="SpecificationLinkBranch">
1442   <complexType>
1443     <sequence>
1444       <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
1445       <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1446       <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1447     </sequence>
1448   </complexType>
1449 </element>
1450

```

1451 Semantic Rules

- 1452 9. Let RO denote the set of all persistent RegistryObject instances in the Registry. The
 1453 following steps will eliminate instances in RO that do not satisfy the conditions of the
 1454 specified filters.
- 1455 a) If RO is empty then continue below.

- 1456 b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a
1457 registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from
1458 RO. If RO is empty then continue below.
- 1459 c) If an ExternalIdentifierFilter element is not specified, or if RO is empty, then continue
1460 below; otherwise, let x be a remaining registry object in RO. If x is not linked to some
1461 ExternalIdentifier instance, then remove x from RO; otherwise, treat each
1462 ExternalIdentifierFilter element separately as follows: Let EI be the set of
1463 ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If
1464 EI is empty, then remove x from RO. If RO is empty then continue below.
- 1465 d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a
1466 remaining registry object in RO. If x doesn't have an auditable event that satisfy
1467 AuditableEventQuery as specified in Section 8.2.4 then remove x from RO. If RO is
1468 empty then continue below.
- 1469 e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining
1470 registry object in RO. If x does not have a name then remove x from RO. If RO is empty
1471 then continue below; otherwise treat NameBranch as follows: If any
1472 LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings
1473 that constitute the name of the registry object then remove x from RO. If RO is empty
1474 then continue below.
- 1475 f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a
1476 remaining registry object in RO. If x does not have a name then remove x from RO. If
1477 RO is empty then continue below; otherwise treat DescriptionBranch as follows: If any
1478 LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings
1479 that constitute the description of the registry object then remove x from RO. If RO is
1480 empty then continue below.
- 1481 g) If a ClassifiedByBranch element is not specified, or if RO is empty, then continue below;
1482 otherwise, let x be a remaining registry object in RO. If x is not the classifiedObject of
1483 some Classification instance, then remove x from RO; otherwise, treat each
1484 ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified
1485 within the ClassifiedByBranch, then let CL be the set of all Classification instances that
1486 have x as the classifiedObject; otherwise, let CL be the set of Classification instances that
1487 satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then
1488 remove x from RO and continue below. Otherwise, if CL is not empty, and if a
1489 ClassificationSchemeQuery is specified, then replace CL by the set of remaining
1490 Classification instances in CL whose defining classification scheme satisfies the
1491 ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and
1492 continue below. Otherwise, if CL remains not empty, and if a ClassificationNodeQuery is
1493 specified, then replace CL by the set of remaining Classification instances in CL for
1494 which a classification node exists and for which that classification node satisfies the
1495 ClassificationNodeQuery. If the new CL is empty, then remove x from RO. If RO is
1496 empty then continue below.

- 1497 h) If a SlotBranch element is not specified, or if RO is empty, then continue below;
1498 otherwise, let x be a remaining registry object in RO. If x is not linked to some Slot
1499 instance, then remove x from RO. If RO is empty then continue below; otherwise, treat
1500 each SlotBranch element separately as follows: If a SlotFilter is not specified within the
1501 SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set
1502 of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,
1503 then remove x from RO and continue below. Otherwise, if SL remains not empty, and if a
1504 SlotValueFilter is specified, replace SL by the set of remaining Slot instances in SL for
1505 which every specified SlotValueFilter is valid. If SL is empty, then remove x from RO. If
1506 RO is empty then continue below.
- 1507 i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,
1508 let x be a remaining registry object in RO. If x is not the source object of some
1509 Association instance, then remove x from RO. If RO is empty then continue below;
1510 otherwise, treat each SourceAssociationBranch element separately as follows:
1511 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be
1512 the set of all Association instances that have x as a source object; otherwise, let AF be the
1513 set of Association instances that satisfy the AssociationFilter and have x as the source
1514 object. If AF is empty, then remove x from RO.
- 1515
1516 If RO is empty then continue below.
- 1517
1518 If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT
1519 be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target
1520 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1521 then continue below.
- 1522
1523 If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let
1524 ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1525 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1526 RO is empty then continue below.
- 1527
1528 If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT
1529 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1530 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1531 empty then continue below.
- 1532
1533 If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT
1534 be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the
1535 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1536 empty then continue below.
- 1537
1538 If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let
1539 ROT be the set of ClassificationScheme instances that satisfy the
1540 ClassificationSchemeQuery and are the target object of some element of AF. If ROT is
1541 empty, then remove x from RO. If RO is empty then continue below.
- 1542

1543 If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let
1544 ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1545 and are the target object of some element of AF. If ROT is empty, then remove x from
1546 RO. If RO is empty then continue below.
1547

1548 If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT
1549 be the set of Organization instances that satisfy the OrganizationQuery and are the target
1550 object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1551 then continue below.
1552

1553 If an AuditableEventQuery is specified within the SourceAssociationBranch, then let
1554 ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1555 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1556 is empty then continue below.
1557

1558 If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let
1559 ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1560 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1561 RO is empty then continue below.
1562

1563 If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let
1564 ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1565 the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1566 is empty then continue below.
1567

1568 If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the
1569 set of Service instances that satisfy the ServiceQuery and are the target object of some
1570 element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue
1571 below.
1572

1573 If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set
1574 of User instances that are the target object of some element of AF. If ROT is empty, then
1575 remove x from RO. If RO is empty then continue below. Let u be the member of ROT. If
1576 a UserFilter element is specified within the UserBranch, and if u does not satisfy that
1577 filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is
1578 empty then continue below. If a PostalAddressFilter element is specified within the
1579 UserBranch, and if the postal address of u does not satisfy that filter, then remove u from
1580 ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. If
1581 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
1582 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
1583 remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then
1584 continue below. If an OrganizationQuery element is specified within the UserBranch,
1585 then let o be the Organization instance that is identified by the organization that u is
1586 affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then
1587 remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then
1588 continue below.
1589

1590 If a ClassificationBranch is specified within the SourceAssociationBranch then let ROT
1591 be the set of Classification instances that are the target object of some element of AF. If
1592 ROT is empty, then remove x from RO. If RO is empty then continue below. Let cb be
1593 the member of ROT. If ClassificationFilter element is specified within the
1594 ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROT. If
1595 ROT is empty, then remove x from RO. If RO is empty then continue below. If a
1596 ClassificationSchemeQuery element is specified within the ClassificationBranch then
1597 replace ROT by the set of remaining Classification instances in ROT whose defining
1598 classification scheme satisfies the ClassificationSchemeQuery. If ROT is empty, then
1599 remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery
1600 element is specified within the ClassificationBranch, then replace ROT by the set of
1601 remaining Classification instances in ROT for which a classification node exists and for
1602 which that classification node satisfies the ClassificationNodeQuery. If ROT is empty,
1603 then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1604 element is specified within the ClassificationBranch element then let cb be a remaining
1605 classification in ROT. Treat RegistryObjectQuery element as follows: Let ROQ be the
1606 result set of the RegistryObjectQuery as defined in [Section 8.2.2](#). If cb is not a
1607 classification for some registry object in ROQ, then remove cb from ROT. If ROT is
1608 empty, then remove x from RO. If RO is empty then continue below. If a
1609 RegistryEntryQuery element is specified within the ClassificationBranch element then let
1610 cb be a remaining classification in ROT. Treat RegistryEntryQuery element as follows:
1611 Let REQ be the result set of the RegistryEntryQuery as defined in [Section 8.2.3](#). If cb is
1612 not a classification for some registry entry in REQ, then remove cb from ROT. If ROT is
1613 empty, then remove x from RO. If RO is empty then continue below.
1614

1615 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1616 ROT be the set of ServiceBinding instances that are the target object of some element of
1617 AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let
1618 sb be the member of ROT. If a ServiceBindingFilter element is specified within the
1619 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROT. If
1620 ROT is empty then remove x from RO. If RO is empty then continue below. If a
1621 SpecificationLinkBranch is specified within the ServiceBindingBranch then consider
1622 each SpecificationLinkBranch element separately as follows:

1623 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link
1624 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1625 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1626 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then
1627 remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1628 element is specified within the SpecificationLinkBranch then let sl be a remaining
1629 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
1630 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
1631 specification link for some registry object in RO, then remove sl from SL. If SL is empty
1632 then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then
1633 continue below. If a RegistryEntryQuery element is specified within the
1634 SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
1635 RegistryEntryQuery element as follows: Let RE be the result set of the
1636 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
1637 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROT.
1638 If ROT is empty then remove x from RO. If RO is empty then continue below. If a
1639 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT
1640 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and
1641 are the target service binding of some element of ROT. If SBT is empty then remove sb
1642 from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue
1643 below.

1644

1645 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
1646 ROT be the set of SpecificationLink instances that are the target object of some element
1647 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.
1648 Let sl be the member of ROT. If a SpecificationLinkFilter element is specified within the
1649 SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROT.
1650 If ROT is empty then remove x from RO. If RO is empty then continue below. If a
1651 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1652 be a remaining specification link in ROT. Treat RegistryObjectQuery element as follows:
1653 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
1654 not a specification link for some registry object in RO, then remove sl from ROT. If ROT
1655 is empty then remove x from RO. If RO is empty then continue below. If a
1656 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
1657 be a remaining specification link in ROT. Treat RegistryEntryQuery element as follows:
1658 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
1659 a specification link for some registry entry in RE, then remove sl from ROT. If ROT is
1660 empty then remove x from RO. If RO is empty then continue below.

1661

1662 If a SourceAssociationBranch is specified within the SourceAssociationBranch, then let
1663 ROT be the set of RegistryObject instances that satisfy the SourceAssociationBranch and
1664 are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1665 RO is empty then continue below.

1666

1667 If a TargetAssociationBranch is specified within the SourceAssociationBranch, then let
1668 ROT be the set of RegistryObject instances that satisfy the TargetAssociationBranch and
1669 are the source object of some element of AF. If ROT is empty, then remove x from RO.
1670 If RO is empty then continue below.

1671 j) If a TargetAssociationBranch element is not specified then go to the next step; otherwise,
1672 let x be a remaining registry object in RO. If x is not the target object of some
1673 Association instance, then remove x from RO. If RO is empty then continue below;
1674 otherwise, treat each TargetAssociationBranch element separately as follows:
1675

1676 If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be
1677 the set of all Association instances that have x as a target object; otherwise, let AF be the
1678 set of Association instances that satisfy the AssociationFilter and have x as the target
1679 object. If AF is empty, then remove x from RO. If RO is empty then continue below.
1680

1681 If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be
1682 the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source
1683 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1684 then continue below.
1685

1686 If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let
1687 ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1688 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1689 RO is empty then continue below.
1690

1691 If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS
1692 be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1693 source object of some element of AF. If ROS is empty, then remove x from RO. If RO is
1694 empty then continue below
1695

1696 If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS
1697 be the set of
1698 RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of
1699 some element of AF. If ROS is empty, then remove x from RO. If RO is empty then
1700 continue below.
1701

1702 If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let
1703 ROS be the set of ClassificationScheme instances that satisfy the
1704 ClassificationSchemeQuery and are the source object of some element of AF. If ROS is
1705 empty, then remove x from RO. If RO is empty then continue below.
1706

1707 If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let
1708 ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1709 and are the source object of some element of AF. If ROS is empty, then remove x from
1710 RO. If RO is empty then continue below.
1711

1712 If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS
1713 be the set of Organization instances that satisfy the OrganizationQuery and are the source
1714 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1715 then continue below.
1716

1717 If an AuditableEventQuery is specified within the TargetAssociationBranch, then let
1718 ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1719 the source object of some element of AF. If ROS is empty, then remove x from RO. If
1720 RO is empty then continue below.
1721

1722 If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let
1723 ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1724 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1725 RO is empty then continue below.
1726

1727 If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let
1728 ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1729 the source object of some element of AF. If ROS is empty, then remove x from RO. If
1730 RO is empty then continue below.
1731

1732 If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the
1733 set of Service instances that satisfy the ServiceQuery and are the source object of some
1734 element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue
1735 below.
1736

1737 If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set
1738 of User instances that are the source object of some element of AF. If ROS is empty, then
1739 remove x from RO. If RO is empty then continue below. Let u be the member of ROS. If
1740 a UserFilter element is specified within the UserBranch, and if u does not satisfy that
1741 filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is
1742 empty then continue below. If a PostalAddressFilter element is specified within the
1743 UserBranch, and if the postal address of u does not satisfy that filter, then remove u from
1744 ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. If
1745 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
1746 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
1747 remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then
1748 continue below. If an OrganizationQuery element is specified within the UserBranch,
1749 then let o be the Organization instance that is identified by the organization that u is
1750 affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then
1751 remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then
1752 continue below.
1753

1754 If a ClassificationBranch is specified within the TargetAssociationBranch then let ROS
1755 be the set of Classification instances that are the source object of some element of AF. If
1756 ROS is empty, then remove x from RO. If RO is empty then continue below. Let cb be
1757 the member of ROS. If ClassificationFilter element is specified within the
1758 ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROS. If
1759 ROS is empty, then remove x from RO. If RO is empty then continue below. If a
1760 ClassificationSchemeQuery element is specified within the ClassificationBranch then
1761 replace ROS by the set of remaining Classification instances in ROS whose defining
1762 classification scheme satisfies the ClassificationSchemeQuery. If ROS is empty, then
1763 remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery
1764 element is specified within the ClassificationBranch, then replace ROS by the set of
1765 remaining Classification instances in ROS for which a classification node exists and for
1766 which that classification node satisfies the ClassificationNodeQuery. If ROS is empty,
1767 then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1768 element is specified within the ClassificationBranch element then let cb be a remaining
1769 classification in ROT. Treat RegistryObjectQuery element as follows: Let ROQ be the
1770 result set of the RegistryObjectQuery as defined in [Section 8.2.2](#). If cb is not a
1771 classification for some registry object in ROQ, then remove cb from ROT. If ROT is
1772 empty, then remove x from RO. If RO is empty then continue below. If a
1773 RegistryEntryQuery element is specified within the ClassificationBranch element then let
1774 cb be a remaining classification in ROT. Treat RegistryEntryQuery element as follows:
1775 Let REQ be the result set of the RegistryEntryQuery as defined in [Section 8.2.3](#). If cb is
1776 not a classification for some registry entry in REQ, then remove cb from ROT. If ROT is
1777 empty, then remove x from RO. If RO is empty then continue below.

1778
1779 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1780 ROS be the set of ServiceBinding instances that are the source object of some element of
1781 AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let
1782 sb be the member of ROS. If a ServiceBindingFilter element is specified within the
1783 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROS. If
1784 ROS is empty then remove x from RO. If RO is empty then continue below. If a
1785 SpecificationLinkBranch is specified within the ServiceBindingBranch then consider
1786 each SpecificationLinkBranch element separately as follows:

1787 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link
1788 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1789 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1790 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then
1791 remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery
1792 element is specified within the SpecificationLinkBranch then let sl be a remaining
1793 specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
1794 result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
1795 specification link for some registry object in RO, then remove sl from SL. If SL is empty
1796 then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then
1797 continue below. If a RegistryEntryQuery element is specified within the
1798 SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
1799 RegistryEntryQuery element as follows: Let RE be the result set of the
1800 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
1801 registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.
1802 If ROS is empty then remove x from RO. If RO is empty then continue below.

1803
 1804 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
 1805 ROS be the set of SpecificationLink instances that are the source object of some element
 1806 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.
 1807 Let sl be the member of ROS. If a SpecificationLinkFilter element is specified within the
 1808 SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROS.
 1809 If ROS is empty then remove x from RO. If RO is empty then continue below. If a
 1810 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
 1811 be a remaining specification link in ROS. Treat RegistryObjectQuery element as follows:
 1812 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
 1813 not a specification link for some registry object in RO, then remove sl from ROS. If ROS
 1814 is empty then remove x from RO. If RO is empty then continue below. If a
 1815 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
 1816 be a remaining specification link in ROS. Treat RegistryEntryQuery element as follows:
 1817 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
 1818 a specification link for some registry entry in RE, then remove sl from ROS. If ROS is
 1819 empty then remove x from RO. If RO is empty then continue below. If a
 1820 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT
 1821 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and
 1822 are the target service binding of some element of ROT. If SBT is empty then remove sb
 1823 from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue
 1824 below.

1825
 1826 If a SourceAssociationBranch is specified within the TargetAssociationBranch, then let
 1827 ROS be the set of RegistryObject instances that satisfy the SourceAssociationBranch and
 1828 are the target object of some element of AF. If ROS is empty, then remove x from RO. If
 1829 RO is empty then continue below.

1830
 1831 k) If a TargetAssociationBranch is specified within the TargetAssociationBranch, then let
 1832 ROS be the set of RegistryObject instances that satisfy the TargetAssociationBranch and
 1833 are the source object of some element of AF. If ROS is empty, then remove x from RO. If
 1834 RO is empty then continue below.

1835 10. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise,
 1836 return RO as the result of the RegistryObjectQuery.

1837 11. Return any accumulated warnings or exceptions as the StatusResult associated with the
 1838 RegistryObjectQuery.

1839 Examples

1840 A client application needs all items that are classified by two different classification schemes,
 1841 one based on "Industry" and another based on "Geography". Both schemes have been defined by
 1842 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",
 1843 respectively. The following query identifies registry entries for all registered items that are
 1844 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of
 1845 "Asia/Japan".

```
1846
1847 <AdhocQueryRequest>
1848   <ResponseOption returnType = "RegistryEntry"/>
1849   <FilterQuery>
```

```

1850 <RegistryObjectQuery>
1851   <ClassifiedByBranch>
1852     <ClassificationFilter>
1853       <Clause>
1854         <SimpleClause leftArgument = "path">
1855           <StringClause stringPredicate = "Equal">//Automotive</StringClause>
1856         </SimpleClause>
1857       </Clause>
1858     </ClassificationFilter>
1859   <ClassificationSchemeQuery>
1860     <NameBranch>
1861       <LocalizedStringFilter>
1862         <Clause>
1863           <SimpleClause leftArgument = "value">
1864             <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
1865           </SimpleClause>
1866         </Clause>
1867       </LocalizedStringFilter>
1868     </NameBranch>
1869   </ClassificationSchemeQuery>
1870 </ClassifiedByBranch>
1871 <ClassifiedByBranch>
1872   <ClassificationFilter>
1873     <Clause>
1874       <SimpleClause leftArgument = "path">
1875         <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
1876       </SimpleClause>
1877     </Clause>
1878   </ClassificationFilter>
1879 <ClassificationSchemeQuery>
1880   <NameBranch>
1881     <LocalizedStringFilter>
1882       <Clause>
1883         <SimpleClause leftArgument = "value">
1884           <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
1885         </SimpleClause>
1886       </Clause>
1887     </LocalizedStringFilter>
1888   </NameBranch>
1889 </ClassificationSchemeQuery>
1890 </ClassifiedByBranch>
1891 </RegistryObjectQuery>
1892 </FilterQuery>
1893 </AdhocQueryRequest>
1894

```

1895 A client application wishes to identify all RegistryObject instances that are classified by some
1896 internal classification scheme and have some given keyword as part of the description of one of
1897 the classification nodes of that classification scheme. The following query identifies all such

1898 RegistryObject instances. The query takes advantage of the knowledge that the classification
 1899 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode
 1900 instances.

1901
 1902
 1903
 1904
 1905
 1906
 1907
 1908
 1909
 1910
 1911
 1912
 1913
 1914
 1915
 1916
 1917
 1918
 1919
 1920
 1921
 1922

```

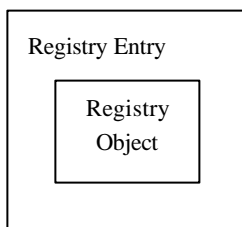
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryObject" />
  <FilterQuery>
    <RegistryObjectQuery>
      <ClassifiedByBranch>
        <ClassificationNodeQuery>
          <DescriptionBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument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

1923 **8.2.3 RegistryEntryQuery**

1924 **Purpose**

1925 To identify a set of registry entry instances as the result of a query over selected registry
 1926 metadata.

1927



1928 **ebRIM Binding**

1929 **Figure 17: ebRIM Binding for RegistryEntryQuery**

1930 **Definition**

1931
 1932
 1933
 1934
 1935
 1936

```

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

```

1937     </sequence>
1938     </extension>
1939     </complexContent>
1940 </complexType>
1941 <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
1942
1943 <element name="RegistryEntryQueryResult">
1944     <complexType>
1945         <choice minOccurs="0" maxOccurs="unbounded">
1946             <element ref="rim:ObjectRef" />
1947             <element ref="rim:ClassificationScheme" />
1948             <element ref="rim:ExtrinsicObject" />
1949             <element ref="rim:RegistryEntry" />
1950             <element ref="rim:RegistryObject" />
1951             <element ref="rim:RegistryPackage" />
1952         </choice>
1953     </complexType>
1954 </element>
1955

```

1956 Semantic Rules

- 1957 12. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
 1958 steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
- 1959 a) If RE is empty then continue below.
- 1960 b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
 1961 registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
 1962 If RE is empty then continue below.
- 1963 c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
 1964 RegistryObjectQuery over RE as explained in section 8.2.2.
- 1965 13. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return
 1966 RE as the result of the RegistryEntryQuery.
- 1967 14. Return any accumulated warnings or exceptions as the StatusResult associated with the
 1968 RegistryEntryQuery.

1969 Examples

1970 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if
 1971 they have registered any of their business documents in the Registry. The following query
 1972 returns a set of registry entry identifiers for currently registered items submitted by any
 1973 organization whose name includes the string "XYZ". It does not return any registry entry
 1974 identifiers for superseded, replaced, deprecated, or withdrawn items.

```

1976 <AdhocQueryRequest>
1977     <ResponseOption returnType = "ObjectRef"/>
1978     <FilterQuery>
1979         <RegistryEntryQuery>
1980             <TargetAssociationBranch>
1981                 <AssociationFilter>
1982                     <Clause>
1983                         <SimpleClause leftArgument = "associationType">
1984                             <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>

```

```

1985     </SimpleClause>
1986     </Clause>
1987   </AssociationFilter>
1988   <OrganizationQuery>
1989     <NameBranch>
1990       <LocalizedStringFilter>
1991         <Clause>
1992           <SimpleClause leftArgument = "value">
1993             <StringClause stringPredicate = "Contains">XYZ</StringClause>
1994           </SimpleClause>
1995         </Clause>
1996       </LocalizedStringFilter>
1997     </NameBranch>
1998   </OrganizationQuery>
1999 </TargetAssociationBranch>
2000 <RegistryEntryFilter>
2001   <Clause>
2002     <SimpleClause leftArgument = "status">
2003       <StringClause stringPredicate = "Equal">Approved</StringClause>
2004     </SimpleClause>
2005   </Clause>
2006 </RegistryEntryFilter>
2007 </RegistryEntryQuery>
2008 </FilterQuery>
2009 </AdhocQueryRequest>
2010

```

2011 A client is using the United Nations Standard Product and Services Classification (UNSPSC)
 2012 scheme and wants to identify all companies that deal with products classified as "Integrated
 2013 circuit components", i.e. UNSPSC code "321118". The client knows that companies have
 2014 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
 2015 such profile has been classified by UNSPSC according to the products the company deals with.
 2016 However, the client does not know if the UNSPSC classification scheme is internal or external to
 2017 this registry. The following query returns a set of approved registry entry instances for CPP's of
 2018 companies that deal with integrated circuit components.

```

2019
2020 <AdhocQueryRequest>
2021   <ResponseOption returnType = "RegistryEntry"/>
2022   <FilterQuery>
2023     <RegistryEntryQuery>
2024       <ClassifiedByBranch>
2025         <ClassificationFilter>
2026           <Clause>
2027             <SimpleClause leftArgument = "code">
2028               <StringClause stringPredicate = "Equal">321118</StringClause>
2029             </SimpleClause>
2030           </Clause>
2031         </ClassificationFilter>
2032       <ClassificationSchemeQuery>
2033         <NameBranch>
2034           <LocalizedStringFilter>
2035             <Clause>
2036               <SimpleClause leftArgument = "value">
2037                 <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
2038               </SimpleClause>
2039             </Clause>
2040           </LocalizedStringFilter>
2041         </NameBranch>

```

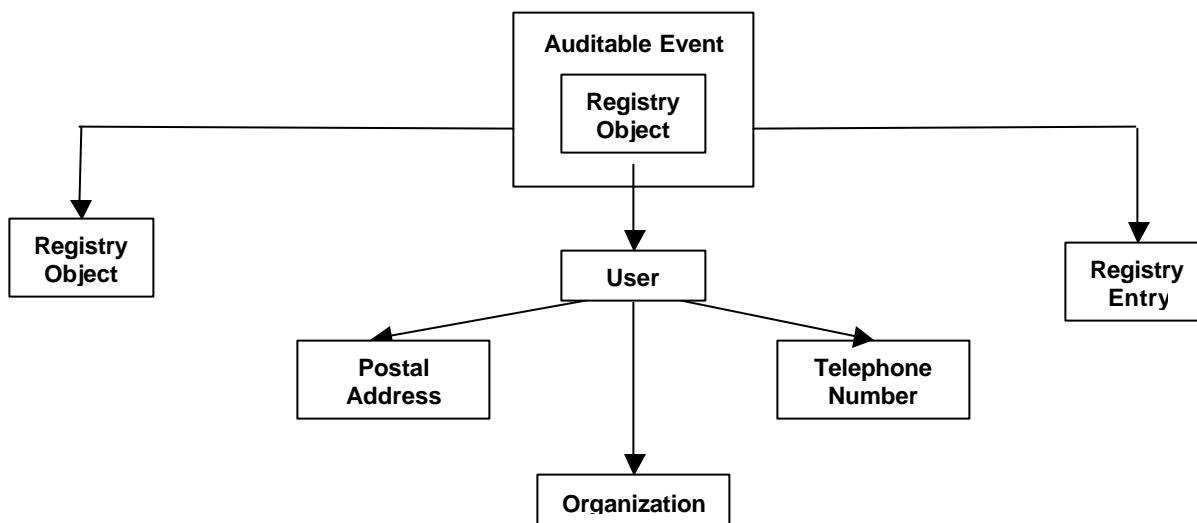
```

2042 </ClassificationSchemeQuery>
2043 </ClassifiedByBranch>
2044 <RegistryEntryFilter>
2045 <Clause>
2046 <CompoundClause connectivePredicate = "And">
2047 <Clause>
2048 <SimpleClause leftArgument = "objectType">
2049 <StringClause stringPredicate = "Equal">CPP</StringClause>
2050 </SimpleClause>
2051 </Clause>
2052 <Clause>
2053 <SimpleClause leftArgument = "status">
2054 <StringClause stringPredicate = "Equal">Approved</StringClause>
2055 </SimpleClause>
2056 </Clause>
2057 </CompoundClause>
2058 </Clause>
2059 </RegistryEntryFilter>
2060 </RegistryEntryQuery>
2061 </FilterQuery>
2062 </AdhocQueryRequest>
2063
    
```

2064 **8.2.4 AuditableEventQuery**

2065 **Purpose**

2066 To identify a set of auditable event instances as the result of a query over selected registry
 2067 metadata.



2068 **ebRIM Binding**

2069 **Figure 18: ebRim binding for AuditableEventQuery**

2070 **Definition**

```

2071 <complexType name="AuditableEventQueryType">
2072 <complexContent>
2073 <extension base="tns:RegistryObjectQueryType">
2074 <sequence>
2075
    
```

```
2076 <element ref="tns:AuditableEventFilter" minOccurs="0" />
2077 <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2078 <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2079 <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2080 </sequence>
2081 </extension>
2082 </complexContent>
2083 </complexType>
2084 <element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />
2085
2086 <element name="AuditableEventQueryResult">
2087 <complexType>
2088 <choice minOccurs="0" maxOccurs="unbounded">
2089 <element ref="rim:ObjectRef" />
2090 <element ref="rim:RegistryObject" />
2091 <element ref="rim:AuditableEvent" />
2092 </choice>
2093 </complexType>
2094 </element>
2095
```

2096 Semantic Rules

- 2097 15. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
2098 following steps will eliminate instances in AE that do not satisfy the conditions of the
2099 specified filters.
- 2100 d) If AE is empty then continue below.
- 2101 e) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an
2102 auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from
2103 AE. If AE is empty then continue below.
- 2104 f) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
2105 x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:
2106 Let RO be the result set of the RegistryObjectQuery as defined in [Section 8.2.2](#). If x is
2107 not an auditable event for some registry object in RO, then remove x from AE. If AE is
2108 empty then continue below.
- 2109 g) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
2110 be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let
2111 RE be the result set of the RegistryEntryQuery as defined in [Section 8.2.3](#). If x is not an
2112 auditable event for some registry entry in RE, then remove x from AE. If AE is empty
2113 then continue below.

- 2114 h) If a UserBranch element is not specified then go to the next step; otherwise, let x be a
 2115 remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter
 2116 element is specified within the UserBranch, and if u does not satisfy that filter, then
 2117 remove x from AE. If a PostalAddressFilter element is specified within the UserBranch,
 2118 and if the postal address of u does not satisfy that filter, then remove x from AE. If
 2119 TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
 2120 TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
 2121 remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if
 2122 any of the EmailAddressFilters isn't satisfied by some of the email addresses of u then
 2123 remove x from AE. If an OrganizationQuery element is specified within the UserBranch,
 2124 then let o be the Organization instance that is identified by the organization that u is
 2125 affiliated with. If o doesn't satisfy OrganizationQuery as defined in **Section 8.2.9** then
 2126 remove x from AE. If AE is empty then continue below.
- 2127 i) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited
 2128 RegistryObjectQuery over AE as explained in **section 8.2.2**.
- 2129 16. If AE is empty, then raise the warning: *auditable event query result is empty*; otherwise
 2130 return AE as the result of the AuditableEventQuery.
- 2131 17. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2132 AuditableEventQuery.

2133 Examples

2134 A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The
 2135 client is now interested in all events since the beginning of the year that have impacted that item.
 2136 The following query will return a set of AuditableEvent instances for all such events.

```

2137 <AdhocQueryRequest>
2138 <ResponseOption returnType = "LeafClass"/>
2139 <FilterQuery>
2140 <AuditableEventQuery>
2141 <AuditableEventFilter>
2142 <Clause>
2143 <SimpleClause leftArgument = "timestamp">
2144 <RationalClause logicalPredicate = "GE">
2145 <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2146 </RationalClause>
2147 </SimpleClause>
2148 </Clause>
2149 </AuditableEventFilter>
2150 <RegistryEntryQuery>
2151 <NameBranch>
2152 <LocalizedStringFilter>
2153 <Clause>
2154 <SimpleClause leftArgument = "value">
2155 <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
2156 </SimpleClause>
2157 </Clause>
2158 </LocalizedStringFilter>
2159 </NameBranch>
2160 </RegistryEntryQuery>
2161 </AuditableEventQuery>
2162 </FilterQuery>
2163 </AdhocQueryRequest>
2164
2165
  
```

2166 A client company has many registered objects in the Registry. The Registry allows events
 2167 submitted by other organizations to have an impact on your registered items, e.g. new
 2168 classifications and new associations. The following query will return a set of identifiers for all
 2169 auditable events, invoked by some other party, that had an impact on an item submitted by
 2170 “myorg”.

```

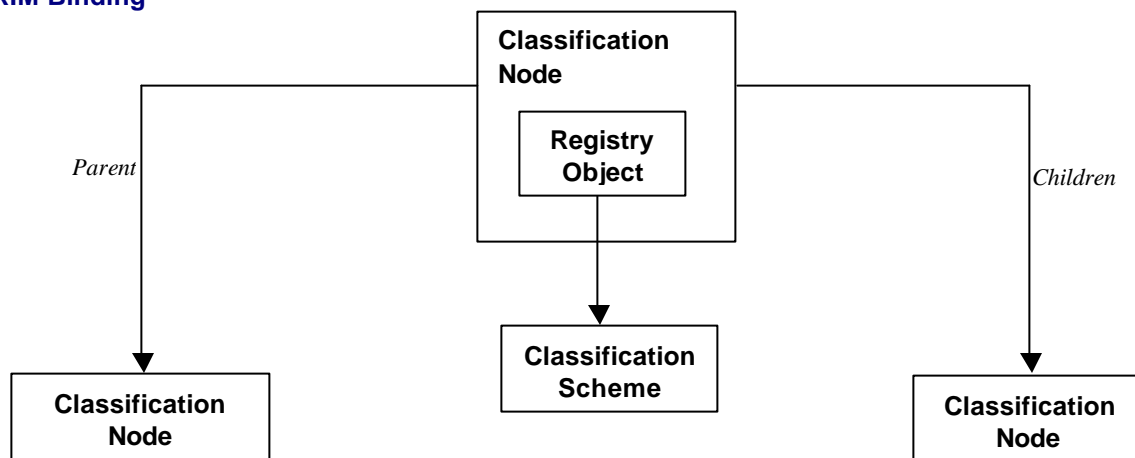
2171 <AdhocQueryRequest>
2172   <ResponseOption returnType = "LeafClass"/>
2173   <FilterQuery>
2174     <AuditableEventQuery>
2175       <RegistryEntryQuery>
2176         <TargetAssociationBranch>
2177           <AssociationFilter>
2178             <Clause>
2179               <SimpleClause leftArgument = "associationType">
2180                 <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2181               </SimpleClause>
2182             </Clause>
2183           </AssociationFilter>
2184         <OrganizationQuery>
2185           <NameBranch>
2186             <LocalizedStringFilter>
2187               <Clause>
2188                 <SimpleClause leftArgument = "value">
2189                   <StringClause stringPredicate = "Equal">myorg</StringClause>
2190                 </SimpleClause>
2191               </Clause>
2192             </LocalizedStringFilter>
2193           </NameBranch>
2194         </OrganizationQuery>
2195       </TargetAssociationBranch>
2196     </RegistryEntryQuery>
2197   <UserBranch>
2198     <OrganizationQuery>
2199       <NameBranch>
2200         <LocalizedStringFilter>
2201           <Clause>
2202             <SimpleClause leftArgument = "value">
2203               <StringClause stringPredicate = "-Equal">myorg</StringClause>
2204             </SimpleClause>
2205           </Clause>
2206         </LocalizedStringFilter>
2207       </NameBranch>
2208     </OrganizationQuery>
2209   </UserBranch>
2210 </AuditableEventQuery>
2211 </FilterQuery>
2212 </AdhocQueryRequest>
2213
2214
```

2215 8.2.5 ClassificationNodeQuery

2216 Purpose

2217 To identify a set of classification node instances as the result of a query over selected registry
 2218 metadata.

2219

ebRIM Binding

2220

Figure 19: ebRim binding for ClassificationNodeQuery

2221

Definition

2222

2223

2224

2225

2226

2227

2228

2229

2230

2231

2232

2233

2234

2235

2236

2237

2238

2239

2240

2241

2242

2243

2244

2245

2246

2247

2248

```

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

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

2249

Semantic Rules

2250

2251

2252

2253

2254

2255

2256

18. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The following steps will eliminate instances in CN that do not satisfy the conditions of the specified filters.

j) If CN is empty then continue below.

k) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove x from CN. If CN is empty then continue below.

- 2257 l) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
 2258 be a remaining classification node in CN. If the defining classification scheme of x does
 2259 not satisfy the ClassificationSchemeQuery as defined in [section 8.2.6](#), then remove x
 2260 from CN. If CN is empty then continue below.
- 2261 m) If a ClassificationNodeParentBranch element is not specified, then go to the next step;
 2262 otherwise, let x be a remaining classification node in CN and execute the following
 2263 paragraph with n=x.
 2264 Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base
 2265 level node), then remove x from CN and go to the next step; otherwise, let p be the parent
 2266 node of n. If a ClassificationNodeFilter element is directly contained in the
 2267 ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,
 2268 then remove x from CN. If CN is empty then continue below. If a
 2269 ClassificationSchemeQuery element is directly contained in the
 2270 ClassificationNodeParentBranch and if defining classification scheme of p does not
 2271 satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then
 2272 continue below.
 2273 If another ClassificationNodeParentBranch element is directly contained within this
 2274 ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.
- 2275 n) If a ClassificationNodeChildrenBranch element is not specified, then continue below;
 2276 otherwise, let x be a remaining classification node in CN. If x is not the parent node of
 2277 some ClassificationNode instance, then remove x from CN and if CN is empty continue
 2278 below; otherwise, treat each ClassificationNodeChildrenBranch element separately and
 2279 execute the following paragraph with n = x.
 2280 Let n be a classification node instance. If a ClassificationNodeFilter element is not
 2281 specified within the ClassificationNodeChildrenBranch element then let CNC be the set
 2282 of all classification nodes that have n as their parent node; otherwise, let CNC be the set
 2283 of all classification nodes that satisfy the ClassificationNodeFilter and have n as their
 2284 parent node. If CNC is empty, then remove x from CN and if CN is empty continue
 2285 below; otherwise, let c be any member of CNC. If a ClassificationSchemeQuery element
 2286 is directly contained in the ClassificationNodeChildrenBranch and if the defining
 2287 classification scheme of c does not satisfy the ClassificationSchemeQuery then remove c
 2288 from CNC. If CNC is empty then remove x from CN. If CN is empty then continue
 2289 below; otherwise, let y be an element of CNC and continue with the next paragraph.
 2290 If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly
 2291 contain another ClassificationNodeChildrenBranch element, then continue below;
 2292 otherwise, repeat the previous paragraph with the new ClassificationNodeChildrenBranch
 2293 element and with n = y.
- 2294 o) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited
 2295 RegistryObjectQuery over CN as explained in [section 8.2.2](#).
- 2296 19. If CN is empty, then raise the warning: *classification node query result is empty*; otherwise
 2297 return CN as the result of the ClassificationNodeQuery.
- 2298 20. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2299 ClassificationNodeQuery.

2300 Path Filter Expression usage in ClassificationNodeFilter

2301 The path filter expression is used to match classification nodes in ClassificationNodeFilter
 2302 elements involving the path attribute of the ClassificationNode class as defined by the getPath

2303 method in [ebRIM].
 2304 The path filter expressions are based on a very small and proper sub-set of location path syntax
 2305 of XPath.

2306 The path filter expression syntax includes support for matching multiple nodes by using wild
 2307 card syntax as follows:

2308 ?? Use of '*' as a wildcard in place of any path element in the pathFilter

2309 ?? Use of '/' syntax to denote any descendent of a node in the pathFilter

2310 It is defined by the following BNF grammar:

```

2311 pathFilter ::= '/' schemeId nodePath
2312 nodePath ::= slashes nodeCode
2313           | slashes '*'
2314           | slashes nodeCode ( nodePath )?
2315 Slashes ::= '/' | '/'
2316
2317
    
```

2318 In the above grammer, schemeId is the id attribute of the ClassificationScheme instance. In the
 2319 above grammar nodeCode is defined by NCName production as defined by
 2320 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

2321 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a
 2322 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause
 2323 is a PATH Filter expression.

2324 This is illustrated in the following example that matches all second level nodes in
 2325 ClassificationScheme with id 'Geography-id' and with code 'Japan':

```

2326
2327 <ClassificationNodeQuery>
2328   <ClassificationNodeFilter>
2329     <Clause>
2330       <SimpleClause leftArgument = "path">
2331         <StringClause stringPredicate = "Equal">//Geography-id/*/Japan</StringClause>
2332       </SimpleClause>
2333     </Clause>
2334   </ClassificationNodeFilter>
2335 </ClassificationNodeQuery>
2336
    
```

2337 **Use Cases and Examples of Path Filter Expressions**

2338 The following table lists various use cases and examples using the sample Geography scheme
 2339 below:

```

2340 <ClassificationScheme id='Geography-id' name="Geography" />
2341
2342 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
2343 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
2344
2345 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
2346 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
2347 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
2348
2349
    
```

2350  **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	/Geography-id/NorthAmerica/*	Match all nodes whose first level

level node whose code is "NorthAmerica"		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'

2351 Examples

2352 A client application wishes to identify all of the classification nodes in the first three levels of a
 2353 classification scheme hierarchy. The client knows that the name of the underlying classification
 2354 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three
 2355 levels.

```

2356 <AdhocQueryRequest>
2357   <ResponseOption returnType = "LeafClass"/>
2358   <FilterQuery>
2359     <ClassificationNodeQuery>
2360       <ClassificationNodeFilter>
2361         <Clause>
2362           <SimpleClause leftArgument = "levelNumber">
2363             <RationalClause logicalPredicate = "LE">
2364               <IntClause>3</IntClause>
2365             </RationalClause>
2366           </SimpleClause>
2367         </Clause>
2368       </ClassificationNodeFilter>
2369     </ClassificationNodeQuery>
2370     <ClassificationSchemeQuery>
2371       <NameBranch>
2372         <LocalizedStringFilter>
2373           <Clause>
2374             <SimpleClause leftArgument = "value">
2375               <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
2376             </SimpleClause>
2377           </Clause>
2378         </LocalizedStringFilter>
2379       </NameBranch>
2380     </ClassificationSchemeQuery>
2381   </ClassificationNodeQuery>
2382 </FilterQuery>
2383 </AdhocQueryRequest>
2384
```

2385 If, instead, the client wishes all levels returned, they could simply delete the
 2386 ClassificationNodeFilter element from the query.

2387 The following query finds all children nodes of a first level node whose code is NorthAmerica.

```

2388 <AdhocQueryRequest>
2389
```

```

2390 <ResponseOption returnType = "LeafClass"/>
2391 <FilterQuery>
2392   <ClassificationNodeQuery>
2393     <ClassificationNodeFilter>
2394       <Clause>
2395         <SimpleClause leftArgument = "path">
2396           <StringClause stringPredicate = "Equal"/>Geography-id/NorthAmerica/*</StringClause>
2397         </SimpleClause>
2398       </Clause>
2399     </ClassificationNodeFilter>
2400   </ClassificationNodeQuery>
2401 </FilterQuery>
2402 </AdhocQueryRequest>
2403

```

2404 The following query finds all third level nodes with code of Tokyo.

```

2405 <AdhocQueryRequest>
2406   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2407   <FilterQuery>
2408     <ClassificationNodeQuery>
2409       <ClassificationNodeFilter>
2410         <Clause>
2411           <SimpleClause leftArgument = "path">
2412             <StringClause stringPredicate = "Equal"/>Geography-id/*/*Tokyo</StringClause>
2413           </SimpleClause>
2414         </Clause>
2415       </ClassificationNodeFilter>
2416     </ClassificationNodeQuery>
2417   </FilterQuery>
2418 </AdhocQueryRequest>
2419
2420

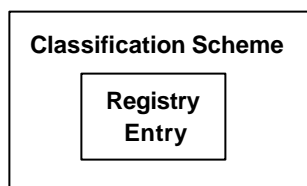
```


2421 8.2.6 ClassificationSchemeQuery

2422 Purpose

2423 To identify a set of classification scheme instances as the result of a query over selected registry
2424 metadata.

2425 ebRIM Binding



2426  Figure 20: ebRIM Binding for ClassificationSchemeQuery

2427 Definition

```

2428 <complexType name="ClassificationSchemeQueryType">
2429   <complexContent>
2430     <extension base="tns:RegistryEntryQueryType">
2431       <sequence>
2432         <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
2433       </sequence>
2434     </extension>
2435

```

```

2436 </complexContent>
2437 </complexType>
2438 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
2439
    
```

2440 **Semantic Rules**

- 2441 21. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The
 2442 following steps will eliminate instances in CS that do not satisfy the conditions of the
 2443 specified filters.
- 2444 p) If CS is empty then continue below.
 - 2445 q) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x
 2446 be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,
 2447 then remove x from CS. If CS is empty then continue below.
 - 2448 r) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited
 2449 RegistryEntryQuery over CS as explained in section 8.2.3.
- 2450 22. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,
 2451 return CS as the result of the ClassificationSchemeQuery.
- 2452 Return any accumulated warnings or exceptions as the StatusResult associated with the
 2453 ClassificationSchemeQuery.

2454 **Examples**

2455 A client application wishes to identify all classification scheme instances in the Registry.

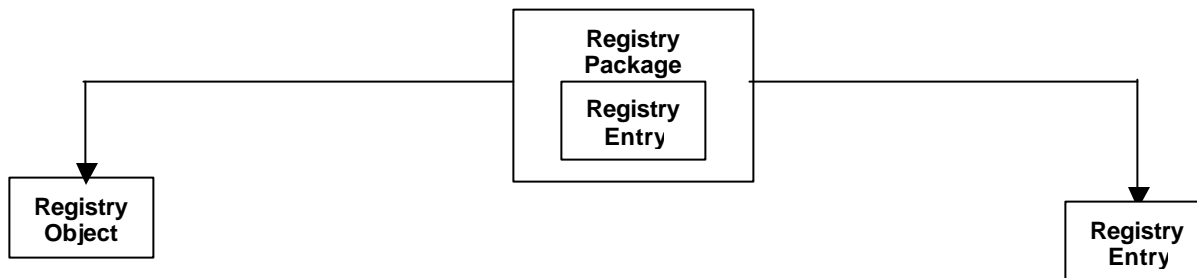
```

2456 <AdhocQueryRequest>
2457 <ResponseOption returnType = "LeafClass"/>
2458 <FilterQuery>
2459 <ClassificationSchemeQuery/>
2460 </FilterQuery>
2461 </AdhocQueryRequest>
    
```

2463 **8.2.7 RegistryPackageQuery**

2464 **Purpose**

2465 To identify a set of registry package instances as the result of a query over selected registry
 2466 metadata.



2467 **ebRIM Binding**

2468 **Figure 21: ebRim binding for RegistryPackageQuery**

2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494

Definition

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

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

Semantic Rules

- 2495
- 2496 23. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The
2497 following steps will eliminate instances in RP that do not satisfy the conditions of the
2498 specified filters.
- 2499 s) If RP is empty then continue below.
- 2500 t) If a RegistryPackageFilter is not specified, then continue below; otherwise, let x be a
2501 registry package instance in RP. If x does not satisfy the RegistryPackageFilter then
2502 remove x from RP. If RP is empty then continue below.
- 2503 u) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery
2504 element then treat each RegistryObjectQuery as follows: let RO be the set of
2505 RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2
2506 and let PO be the subset of RO that are members of the package x. If PO is empty, then
2507 remove x from RP. If RP is empty then continue below. If a RegistryEntryQuery element
2508 is directly contained in the RegistryPackageQuery element then treat each
2509 RegistryEntryQuery as follows: let RE be the set of RegistryEntry instances returned by
2510 the RegistryEntryQuery as defined in Section 8.2.3 and let PE be the subset of RE that
2511 are members of the package x. If PE is empty, then remove x from RP. If RP is empty
2512 then continue below.
- 2513 v) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited
2514 RegistryEntryQuery over RP as explained in section 8.2.3.
- 2515 24. If RP is empty, then raise the warning: *registry package query result is empty*; otherwise
2516 return RP as the result of the RegistryPackageQuery.
- 2517 25. Return any accumulated warnings or exceptions as the StatusResult associated with the
2518 RegistryPackageQuery.

2519 **Examples**

2520 A client application wishes to identify all package instances in the Registry that contain an
 2521 Invoice extrinsic object as a member of the package.

```

2522
2523 <AdhocQueryRequest>
2524   <ResponseOption returnType = "LeafClass"/>
2525   <FilterQuery>
2526     <RegistryPackageQuery>
2527       <RegistryEntryQuery>
2528         <RegistryEntryFilter>
2529           <Clause>
2530             <SimpleClause leftArgument = "objectType">
2531               <StringClause stringPredicate = "Equal">Invoice</StringClause>
2532             </SimpleClause>
2533           </Clause>
2534         </RegistryEntryFilter>
2535       </RegistryEntryQuery>
2536     </RegistryPackageQuery>
2537   </FilterQuery>
2538 </AdhocQueryRequest>
2539
  
```

2540 A client application wishes to identify all package instances in the Registry that are not empty.

```

2541
2542 <AdhocQueryRequest>
2543   <ResponseOption returnType = "LeafClass"/>
2544   <FilterQuery>
2545     <RegistryPackageQuery>
2546       <RegistryObjectQuery/>
2547     </RegistryPackageQuery>
2548   </FilterQuery>
2549 </AdhocQueryRequest>
2550
  
```

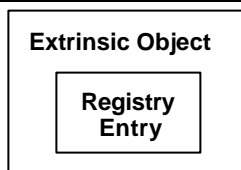
2551 A client application wishes to identify all package instances in the Registry that are empty. Since
 2552 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate
 2553 RegistryPackageQuery requests, one to find all packages and another to find all non-empty
 2554 packages, and then do the set difference themselves. Alternatively, they could do a more
 2555 complex RegistryEntryQuery and check that the packaging association between the package and
 2556 its members is non-existent.

2557 **Note:** A registry package is an intrinsic RegistryEntry instance that is completely determined by
 2558 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an
 2559 equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However,
 2560 the equivalent RegistryEntryQuery is often more complicated to write.

2561 **8.2.8 ExtrinsicObjectQuery**2562 **Purpose**

2563 To identify a set of extrinsic object instances as the result of a query over selected registry
 2564 metadata.

2565 **ebRIM Binding**



2566

☞ ☞ Figure 22: ebRIM Binding for ExtrinsicObjectQuery

2567 **Definition**

```

2568
2569 <complexType name="ExtrinsicObjectQueryType">
2570   <complexContent>
2571     <extension base="tns:RegistryEntryQueryType">
2572       <sequence>
2573         <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
2574       </sequence>
2575     </extension>
2576   </complexContent>
2577 </complexType>
2578 <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
2579
2580 <element name="ExtrinsicObjectQueryResult">
2581   <complexType>
2582     <choice minOccurs="0" maxOccurs="unbounded">
2583       <element ref="rim:ObjectRef" />
2584       <element ref="rim:RegistryEntry" />
2585       <element ref="rim:RegistryObject" />
2586       <element ref="rim:ExtrinsicObject" />
2587     </choice>
2588   </complexType>
2589 </element>
2590
  
```

2591 **Semantic Rules**

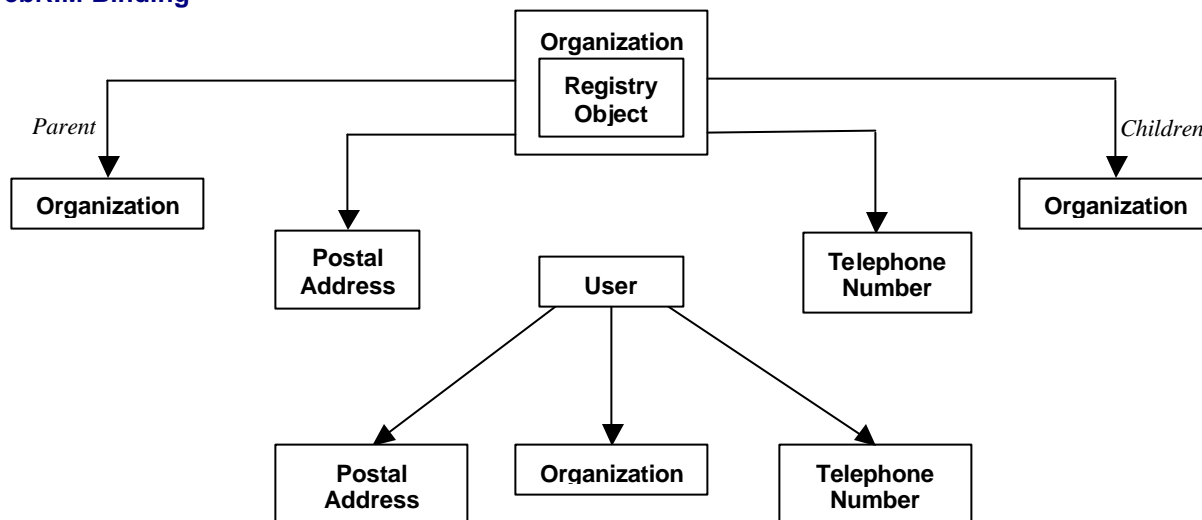
- 2592 26. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The
 2593 following steps will eliminate instances in EO that do not satisfy the conditions of the
 2594 specified filters.
- 2595 w) If EO is empty then continue below.
- 2596 x) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an
 2597 extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from
 2598 EO. If EO is empty then continue below.
- 2599 y) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited
 2600 RegistryEntryQuery over EO as explained in section 8.2.3.
- 2601 27. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise,
 2602 return EO as the result of the ExtrinsicObjectQuery.
- 2603 28. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2604 ExtrinsicObjectQuery.

2605 **8.2.9 OrganizationQuery**2606 **Purpose**

2607 To identify a set of organization instances as the result of a query over selected registry

2608 metadata.

2609 **ebRIM Binding**



2610 Figure 23: ebRim Binding for OrganizationQuery

2611 **Definition**

```

2612 <complexType name="OrganizationQueryType">
2613   <complexContent>
2614     <extension base="tns:RegistryObjectQueryType">
2615       <sequence>
2616         <element ref="tns:OrganizationFilter" minOccurs="0" maxOccurs="1" />
2617         <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
2618         <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
2619         <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2620         <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0"
2621           maxOccurs="1" />
2622         <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
2623           maxOccurs="unbounded" />
2624       </sequence>
2625     </extension>
2626   </complexContent>
2627 </complexType>
2628 <element name="OrganizationQuery" type="tns:OrganizationQueryType" />
2629
2630 <element name="OrganizationQueryResult">
2631   <complexType>
2632     <choice minOccurs="0" maxOccurs="unbounded">
2633       <element ref="rim:ObjectRef" />
2634       <element ref="rim:RegistryObject" />
2635       <element ref="rim:Organization" />
2636     </choice>
2637   </complexType>
2638 </element>
2639
2640

```

2641 **Semantic Rules**

2642 29. Let ORG denote the set of all persistent Organization instances in the Registry. The
 2643 following steps will eliminate instances in ORG that do not satisfy the conditions of the
 2644 specified filters.

- 2645 z) If ORG is empty then continue below.
- 2646 aa) If an OrganizationFilter element is not directly contained in the OrganizationQuery
2647 element, then go to the next step; otherwise let x be an organization instance in ORG. If x
2648 does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then
2649 continue below.
- 2650 bb) If a PostalAddressFilter element is not directly contained in the OrganizationQuery
2651 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal
2652 address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is
2653 empty then continue below.
- 2654 cc) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery
2655 element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of
2656 the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then
2657 remove x from ORG. If ORG is empty then continue below.
- 2658 dd) If a UserBranch element is not directly contained in the OrganizationQuery element then
2659 go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user
2660 instance that is affiliated with x. If a UserFilter element is specified within the
2661 UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a
2662 PostalAddressFilter element is specified within the UserBranch, and if the postal address
2663 of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s)
2664 are specified within the UserBranch and if any of the TelephoneNumberFilters isn't
2665 satisfied by some of the telephone numbers of x then remove x from ORG. If
2666 EmailAddressFilter(s) are specified within the UserBranch and if any of the
2667 EmailAddressFilters isn't satisfied by some of the email addresses of x then remove x
2668 from ORG. If an OrganizationQuery element is specified within the UserBranch, then let
2669 o be the Organization instance that is identified by the organization that u is affiliated
2670 with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then remove x
2671 from ORG. If ORG is empty then continue below.
- 2672 ee) If a OrganizationParentBranch element is not specified within the OrganizationQuery,
2673 then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the
2674 following paragraph with o = x:
2675 Let o be an organization instance. If an OrganizationFilter is not specified within the
2676 OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the
2677 Organization hierarchy), then remove x from ORG; otherwise, let p be the parent
2678 organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.
2679 If ORG is empty then continue below.
2680 If another OrganizationParentBranch element is directly contained within this
2681 OrganizationParentBranch element, then repeat the previous paragraph with o = p.
- 2682 ff) If a OrganizationChildrenBranch element is not specified, then continue below;
2683 otherwise, let x be a remaining organization in ORG. If x is not the parent node of some
2684 organization instance, then remove x from ORG and if ORG is empty continue below;
2685 otherwise, treat each OrganizationChildrenBranch element separately and execute the
2686 following paragraph with n = x.

- 2687 Let n be an organization instance. If an `OrganizationFilter` element is not specified within
 2688 the `OrganizationChildrenBranch` element then let `ORGC` be the set of all organizations
 2689 that have n as their parent node; otherwise, let `ORGC` be the set of all organizations that
 2690 satisfy the `OrganizationFilter` and have n as their parent node. If `ORGC` is empty, then
 2691 remove x from `ORG` and if `ORG` is empty continue below; otherwise, let c be any
 2692 member of `ORGC`. If a `PostalAddressFilter` element is directly contained in the
 2693 `OrganizationChildrenBranch` and if the postal address of c does not satisfy the
 2694 `PostalAddressFilter` then remove c from `ORGC`. If `ORGC` is empty then remove x from
 2695 `ORG`. If `ORG` is empty then continue below. If no `TelephoneNumberFilter` element is
 2696 directly contained in the `OrganizationChildrenBranch` and if any of the
 2697 `TelephoneNumberFilters` isn't satisfied by some of the telephone numbers of c then
 2698 remove c from `ORGC`. If `ORGC` is empty then remove x from `ORG`. If `ORG` is empty
 2699 then continue below; otherwise, let y be an element of `ORGC` and continue with the next
 2700 paragraph.
- 2701 If the `OrganizationChildrenBranch` element is terminal, i.e. if it does not directly contain
 2702 another `OrganizationChildrenBranch` element, then continue below; otherwise, repeat the
 2703 previous paragraph with the new `OrganizationChildrenBranch` element and with $n = y$.
- 2704 gg) Let `ORG` be the set of remaining `Organization` instances. Evaluate inherited
 2705 `RegistryObjectQuery` over `ORG` as explained in section 8.2.2.
- 2706 30. If `ORG` is empty, then raise the warning: *organization query result is empty*; otherwise return
 2707 `ORG` as the result of the `OrganizationQuery`.
- 2708 31. Return any accumulated warnings or exceptions as the `StatusResult` associated with the
 2709 `OrganizationQuery`.

2710 Examples

2711 A client application wishes to identify a set of organizations, based in France, that have
 2712 submitted a `PartyProfile` extrinsic object this year.

```

2713
2714 <AdhocQueryRequest>
2715   <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2716   <FilterQuery>
2717     <OrganizationQuery>
2718       <SourceAssociationBranch>
2719         <AssociationFilter>
2720           <Clause>
2721             <SimpleClause leftArgument = "associationType">
2722               <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2723             </SimpleClause>
2724           </Clause>
2725         </AssociationFilter>
2726         <RegistryObjectQuery>
2727           <RegistryObjectFilter>
2728             <Clause>
2729               <SimpleClause leftArgument = "objectType">
2730                 <StringClause stringPredicate = "Equal">CPP</StringClause>
2731               </SimpleClause>
2732             </Clause>
2733           </RegistryObjectFilter>
2734           <AuditableEventQuery>
2735             <AuditableEventFilter>
2736               <Clause>
2737                 <SimpleClause leftArgument = "timestamp">
2738                   <RationalClause logicalPredicate = "GE">
2739                     <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2740                   </RationalClause>
2741                 </SimpleClause>
2742               </Clause>
2743             </AuditableEventFilter>
2744           </AuditableEventQuery>

```

```

2745         </RegistryObjectQuery>
2746     </SourceAssociationBranch>
2747     <PostalAddressFilter>
2748         <Clause>
2749             <SimpleClause leftArgument = "country">
2750                 <StringClause stringPredicate = "Equal">France</StringClause>
2751             </SimpleClause>
2752         </Clause>
2753     </PostalAddressFilter>
2754 </OrganizationQuery>
2755 </FilterQuery>
2756 </AdhocQueryRequest>
2757

```

2758 A client application wishes to identify all organizations that have Corporation named XYZ as a
 2759 parent.

```

2760 <AdhocQueryRequest>
2761 <ResponseOption returnType = "LeafClass" />
2762 <FilterQuery>
2763     <OrganizationQuery>
2764         <OrganizationParentBranch>
2765             <NameBranch>
2766                 <LocalizedStringFilter>
2767                     <Clause>
2768                         <SimpleClause leftArgument = "value">
2769                             <StringClause stringPredicate = "Equal">XYZ</StringClause>
2770                         </SimpleClause>
2771                     </Clause>
2772                 </LocalizedStringFilter>
2773             </NameBranch>
2774         </OrganizationParentBranch>
2775     </OrganizationQuery>
2776 </FilterQuery>
2777 </AdhocQueryRequest>
2778
2779

```

2780 **8.2.10 ServiceQuery**

2781 **Purpose**

2782
 2783 To identify a set of service instances as the result of a query over selected registry metadata.

2784 **ebRIM Binding**

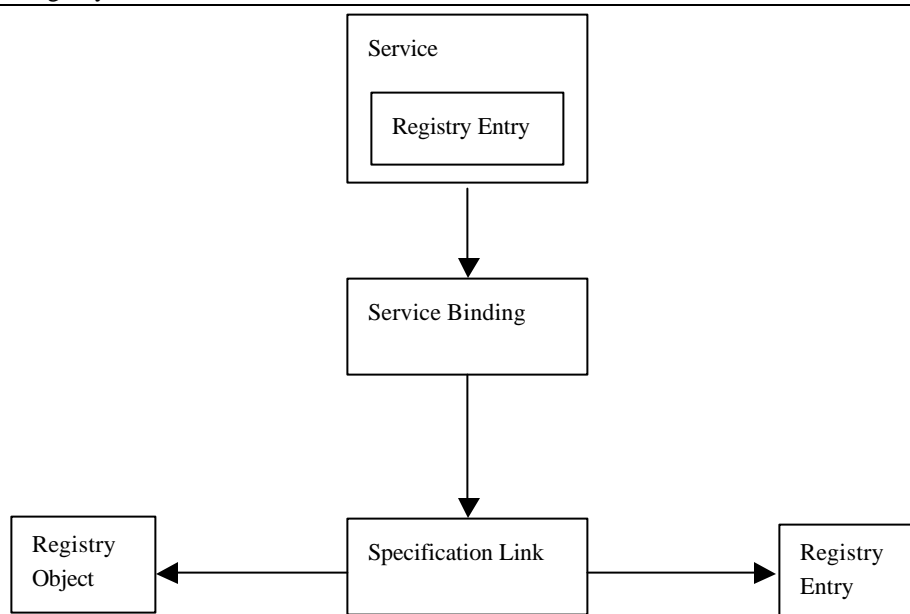


Figure 24: ebRIM Binding for ServiceQuery

2785

Definition

2786

2787

2788

2789

2790

2791

2792

2793

2794

2795

2796

2797

2798

2799

2800

2801

2802

2803

2804

2805

2806

2807

2808

2809

2810

2811

```

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

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

Semantic Rules

2812

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

2814

hh) If S is empty then continue below.

2815

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

2816

2817

2818

- 2819 jj) If a ServiceBindingBranch is not specified then continue below; otherwise, consider each
 2820 ServiceBindingBranch element separately as follows:
 2821 Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the
 2822 member of SB. If a ServiceBindingFilter element is specified within the
 2823 ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If
 2824 SB is empty then remove x from S. If S is empty then continue below. If a
 2825 SpecificationLinkBranch is not specified within the ServiceBindingBranch then continue
 2826 below; otherwise, consider each SpecificationLinkBranch element separately as follows:
 2827 Let sb be a remaining service binding in SB. Let SL be the set of all specification link
 2828 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
 2829 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
 2830 remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove
 2831 x from S. If S is empty then continue below. If a RegistryObjectQuery element is
 2832 specified within the SpecificationLinkBranch then let sl be a remaining specification link
 2833 in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the
 2834 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
 2835 registry object in RO, then remove sl from SL. If SL is empty then remove sb from SB. If
 2836 SB is empty then remove x from S. If S is empty then continue below. If a
 2837 RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl
 2838 be a remaining specification link in SL. Treat RegistryEntryQuery element as follows:
 2839 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not
 2840 a specification link for some registry entry in RE, then remove sl from SL. If SL is empty
 2841 then remove sb from SB. If SB is empty then remove x from S. If S is empty then
 2842 continue below.
- 2843 kk) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery
 2844 over AE as explained in section 8.2.3.
- 2845 33. If S is empty, then raise the warning: *service query result is empty*; otherwise return S as the
 2846 result of the ServiceQuery.
- 2847 34. Return any accumulated warnings or exceptions as the StatusResult associated with the
 2848 ServiceQuery.

2849 **Examples**

2851 **8.2.11 Registry Filters**

2852 **Purpose**

2853 To identify a subset of the set of all persistent instances of a given registry class.

2854 **Definition**

```

2855       <complexType name="FilterType">
2856           <sequence>
2857             <element ref="tns:Clause" />
2858           </sequence>
2859       </complexType>
2860       <element name="RegistryObjectFilter" type="tns:FilterType" />
2861       <element name="RegistryEntryFilter" type="tns:FilterType" />
2862       <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
  
```

```

2864 <element name="RegistryPackageFilter" type="tns:FilterType" />
2865 <element name="OrganizationFilter" type="tns:FilterType" />
2866 <element name="ClassificationNodeFilter" type="tns:FilterType" />
2867 <element name="AssociationFilter" type="tns:FilterType" />
2868 <element name="ClassificationFilter" type="tns:FilterType" />
2869 <element name="ClassificationSchemeFilter" type="tns:FilterType" />
2870 <element name="ExternalLinkFilter" type="tns:FilterType" />
2871 <element name="ExternalIdentifierFilter" type="tns:FilterType" />
2872 <element name="SlotFilter" type="tns:FilterType" />
2873 <element name="AuditableEventFilter" type="tns:FilterType" />
2874 <element name="UserFilter" type="tns:FilterType" />
2875 <element name="SlotValueFilter" type="tns:FilterType" />
2876 <element name="PostalAddressFilter" type="tns:FilterType" />
2877 <element name="TelephoneNumberFilter" type="tns:FilterType" />
2878 <element name="ServiceFilter" type="tns:FilterType" />
2879 <element name="ServiceBindingFilter" type="tns:FilterType" />
2880 <element name="SpecificationLinkFilter" type="tns:FilterType" />
2881 <element name="LocalizedStringFilter" type="tns:FilterType" />
2882

```

2883 Semantic Rules

- 2884 35. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
- 2885 36. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
 2886 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in
 2887 [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set
 2888 of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the
 2889 Clause predicate.
- 2890 37. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing
 2891 SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in
 2892 [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
 2893 returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
 2894 for the Clause predicate.
- 2895 38. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
 2896 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in
 2897 [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter
 2898 returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to
 2899 *True* for the Clause predicate.
- 2900 39. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing
 2901 SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in
 2902 [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns
 2903 a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for
 2904 the Clause predicate.
- 2905 40. For every OrganizationFilter XML element, the leftArgument attribute of any containing
 2906 SimpleClause shall identify a public attribute of the Organization or PostalAddress UML
 2907 classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The
 2908 OrganizationFilter returns a set of identifiers for Organization instances whose attribute
 2909 values evaluate to *True* for the Clause predicate.

- 2910 41. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
2911 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
2912 class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the
2913 leftAttribute is the visible attribute “path” then if stringPredicate of the StringClause is not
2914 “Equal” then raise exception: *classification node path attribute error*. The
2915 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose
2916 attribute values evaluate to *True* for the Clause predicate.
- 2917 42. For every AssociationFilter XML element, the leftArgument attribute of any containing
2918 SimpleClause shall identify a public attribute of the Association UML class defined in
2919 [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a
2920 set of identifiers for Association instances whose attribute values evaluate to *True* for the
2921 Clause predicate.
- 2922 43. For every ClassificationFilter XML element, the leftArgument attribute of any containing
2923 SimpleClause shall identify a public attribute of the Classification UML class defined in
2924 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter
2925 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*
2926 for the Clause predicate.
- 2927 44. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any
2928 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
2929 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The
2930 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances
2931 whose attribute values evaluate to *True* for the Clause predicate.
- 2932 45. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing
2933 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in
2934 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns
2935 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the
2936 Clause predicate.
- 2937 46. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing
2938 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in
2939 [ebRIM]. If not, raise exception: *external identifier attribute error*. The
2940 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose
2941 attribute values evaluate to *True* for the Clause predicate.
- 2942 47. For every SlotFilter XML element, the leftArgument attribute of any containing
2943 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If
2944 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot
2945 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2946 48. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
2947 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in
2948 [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter
2949 returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to
2950 *True* for the Clause predicate.
- 2951 49. For every UserFilter XML element, the leftArgument attribute of any containing
2952 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If
2953 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User
2954 instances whose attribute values evaluate to *True* for the Clause predicate.

- 2955 50. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one
2956 SlotValue instance for each “value” in the “values” list of a Slot instance. The visible
2957 attribute of SlotValue is “value”. It is a character string. The dynamic instances of SlotValue
2958 are derived from the “values” attribute defined in ebRIM for a Slot instance. For every
2959 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause
2960 shall identify the “value” attribute of the SlotValue class just defined. If not, raise exception:
2961 *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose “value”
2962 attribute evaluates to *True* for the Clause predicate.
- 2963 51. For every PostalAddressFilter XML element, the leftArgument attribute of any containing
2964 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in
2965 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter
2966 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*
2967 for the Clause predicate.
- 2968 52. For every TelephoneNumberFilter XML element, the leftArgument attribute of any
2969 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML
2970 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.
2971 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances
2972 whose attribute values evaluate to *True* for the Clause predicate.
- 2973 53. For every ServiceFilter XML element, the leftArgument attribute of any containing
2974 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].
2975 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for
2976 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 2977 54. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
2978 SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in
2979 [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter
2980 returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to
2981 *True* for the Clause predicate.
- 2982 55. For every SpecificationLinkFilter XML element, the leftArgument attribute of any
2983 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class
2984 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The
2985 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose
2986 attribute values evaluate to *True* for the Clause predicate.
- 2987 56. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing
2988 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in
2989 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter
2990 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to
2991 *True* for the Clause predicate.

2992 **8.2.12 XML Clause Constraint Representation**

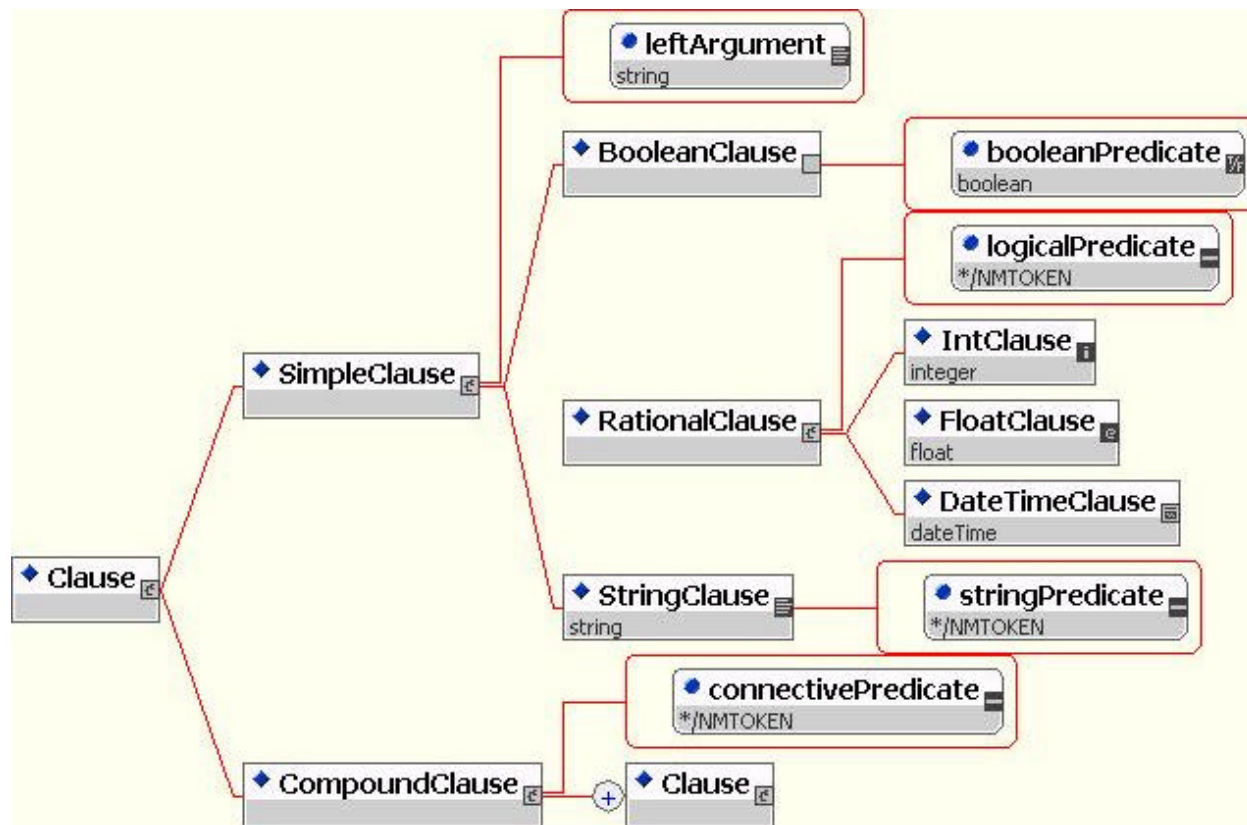
2993 **Purpose**

2994 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate Clauses*.
2995 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to
2996 simply as *Clauses* in this specification.

2997 **Conceptual Diagram**

2998 The following is a conceptual diagram outlining the Clause structure.

2999



3000
3001

Figure 25: The Clause Structure

3002 **Semantic Rules**

3003 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate - RightArgument"
3004 format to form a *Clause*. There are two types of Clauses: *SimpleClauses* and *CompoundClauses*.

3005 *SimpleClauses*

3006 A *SimpleClause* always defines the leftArgument as a text string, sometimes referred to as the
3007 *Subject* of the Clause. *SimpleClause* itself is incomplete (abstract) and must be extended.
3008 *SimpleClause* is extended to support *BooleanClause*, *StringClause*, and *RationalClause*
3009 (abstract).

3010 *BooleanClause* implicitly defines the predicate as 'equal to', with the right argument as a
3011 boolean. *StringClause* defines the predicate as an enumerated attribute of appropriate string-
3012 compare operations and a right argument as the element's text data. Rational number support is
3013 provided through a common *RationalClause* providing an enumeration of appropriate rational
3014 number compare operations, which is further extended to *IntClause* and *FloatClause*, each with
3015 appropriate signatures for the right argument.

3016 *CompoundClauses*

3017 A *CompoundClause* contains two or more Clauses (Simple or Compound) and a connective
3018 predicate. This provides for arbitrarily complex Clauses to be formed.

3019 **Definition**

3020
3021
3022

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

```
3023     <documentation xml:lang = "en">
3024 The following lines define the XML syntax for Clause.
3025
3026     </documentation>
3027 </annotation>
3028 <complexType>
3029   <choice>
3030     <element ref = "tns:SimpleClause"/>
3031     <element ref = "tns:CompoundClause"/>
3032   </choice>
3033 </complexType>
3034 </element>
3035 <element name = "SimpleClause">
3036   <complexType>
3037     <choice>
3038       <element ref = "tns:BooleanClause"/>
3039       <element ref = "tns:RationalClause"/>
3040       <element ref = "tns:StringClause"/>
3041     </choice>
3042     <attribute name = "leftArgument" use = "required" type =
3043 "string"/>
3044   </complexType>
3045 </element>
3046 <element name = "CompoundClause">
3047   <complexType>
3048     <sequence>
3049       <element ref = "tns:Clause" maxOccurs = "unbounded"/>
3050     </sequence>
3051     <attribute name = "connectivePredicate" use = "required">
3052       <simpleType>
3053         <restriction base = "NMTOKEN">
3054           <enumeration value = "And"/>
3055           <enumeration value = "Or"/>
3056         </restriction>
3057       </simpleType>
3058     </attribute>
3059   </complexType>
3060 </element>
3061 <element name = "BooleanClause">
3062   <complexType>
3063     <attribute name = "booleanPredicate" use = "required" type =
3064 "boolean"/>
3065   </complexType>
3066 </element>
3067 <element name = "RationalClause">
3068   <complexType>
3069     <choice>
3070       <element ref = "tns:IntClause"/>
3071       <element ref = "tns:FloatClause"/>
3072       <element ref = "tns:DateTimeClause"/>
3073     </choice>
3074     <attribute name = "logicalPredicate" use = "required">
3075       <simpleType>
3076         <restriction base = "NMTOKEN">
3077           <enumeration value = "LE"/>
3078           <enumeration value = "LT"/>
3079           <enumeration value = "GE"/>
3080           <enumeration value = "GT"/>
3081           <enumeration value = "EQ"/>
3082           <enumeration value = "NE"/>
```

```

3083     </restriction>
3084     </simpleType>
3085     </attribute>
3086   </complexType>
3087 </element>
3088 <element name = "IntClause" type = "integer"/>
3089 <element name = "FloatClause" type = "float"/>
3090 <element name = "DateTimeClause" type = "dateTime"/>
3091
3092 <element name = "StringClause">
3093   <complexType>
3094     <simpleContent>
3095       <extension base = "string">
3096         <attribute name = "stringPredicate" use = "required">
3097           <simpleType>
3098             <restriction base = "NMTOKEN">
3099               <enumeration value = "Contains"/>
3100               <enumeration value = "-Contains"/>
3101               <enumeration value = "StartsWith"/>
3102               <enumeration value = "-StartsWith"/>
3103               <enumeration value = "Equal"/>
3104               <enumeration value = "-Equal"/>
3105               <enumeration value = "EndsWith"/>
3106               <enumeration value = "-EndsWith"/>
3107             </restriction>
3108           </simpleType>
3109         </attribute>
3110       </extension>
3111     </simpleContent>
3112   </complexType>
3113 </element>
3114

```

3115 Examples

3116 Simple BooleanClause: "Smoker" = True

```

3117
3118 <Clause>
3119   <SimpleClause leftArgument="Smoker">
3120     <BooleanClause booleanPredicate="True"/>
3121   </SimpleClause>
3122 </Clause>
3123

```

3124 Simple StringClause: "Smoker" contains "mo"

```

3125
3126 <Clause>
3127   <SimpleClause leftArgument = "Smoker">
3128     <StringClause stringPredicate = "Contains">mo</StringClause>
3129   </SimpleClause>
3130 </Clause>

```

3131 Simple IntClause: "Age" >= 7

```

3132
3133 <Clause>
3134   <SimpleClause leftArgument="Age">
3135     <RationalClause logicalPredicate="GE">
3136       <IntClause>7</IntClause>
3137     </RationalClause>
3138   </SimpleClause>

```

3139 </Clause>
 3140

Simple FloatClause: "Size" = 4.3

3141
 3142
 3143 <Clause>
 3144 <SimpleClause leftArgument="Size">
 3145 <RationalClause logicalPredicate="Equal">
 3146 <FloatClause>4.3</FloatClause>
 3147 </RationalClause>
 3148 </SimpleClause>
 3149 </Clause>
 3150

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

3151
 3152
 3153 <Clause>
 3154 <CompoundClause connectivePredicate="And">
 3155 <Clause>
 3156 <SimpleClause leftArgument="Smoker">
 3157 <BooleanClause booleanPredicate="False"/>
 3158 </SimpleClause>
 3159 </Clause>
 3160 <Clause>
 3161 <SimpleClause leftArgument="Age">
 3162 <RationalClause logicalPredicate="LE">
 3163 <IntClause>45</IntClause>
 3164 </RationalClause>
 3165 </SimpleClause>
 3166 </Clause>
 3167 </CompoundClause>
 3168 </Clause>
 3169

3170 Compound with one Simple and one Compound

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

3171
 3172
 3173 <Clause>
 3174 <CompoundClause connectivePredicate="And">
 3175 <Clause>
 3176 <SimpleClause leftArgument="Smoker">
 3177 <BooleanClause booleanPredicate="False"/>
 3178 </SimpleClause>
 3179 </Clause>
 3180 <Clause>
 3181 <CompoundClause connectivePredicate="Or">
 3182 <Clause>
 3183 <SimpleClause leftArgument="Age">
 3184 <RationalClause logicalPredicate="LE">
 3185 <IntClause>45</IntClause>
 3186 </RationalClause>
 3187 </SimpleClause>
 3188 </Clause>
 3189 <Clause>
 3190 <SimpleClause leftArgument="American">
 3191 <BooleanClause booleanPredicate="True"/>
 3192 </SimpleClause>
 3193 </Clause>
 3194 </CompoundClause>

```
3195     </Clause>
3196     </CompoundClause>
3197 <Clause>
3198
```

3199 **8.3 SQL Query Support**

3200 The Registry may optionally support an SQL based query capability that is designed for Registry
3201 clients that demand more advanced query capability. The optional SQLQuery element in the
3202 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query
3203 language.

3204 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of
3205 the “SELECT” statement of Entry level SQL defined by ISO/IEC 9075:1992, Database
3206 Language SQL [SQL], extended to include <sql invoked routines> (also known as
3207 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined
3208 in template form in Appendix 0. The syntax of the Registry query language is defined by the
3209 BNF grammar in 0.

3210 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use
3211 relational databases in a Registry implementation.

3212 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

3213 SQL Queries are defined based upon the query syntax in in Appendix 0 and a fixed relational
3214 schema defined in Appendix 0. The relational schema is an algorithmic binding to [ebRIM] as
3215 described in the following sections.

3216 **8.3.1.1 Class Binding**

3217 A subset of the class names defined in [ebRIM] map to table names that may be queried by an
3218 SQL query. Appendix 0 defines the names of the ebRIM classes that may be queried by an SQL
3219 query.

3220 The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix 0 is
3221 as follows:

3222 ?? Classes that have concrete instances are mapped to relational tables. In addition entity classes
3223 (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.

3224 ?? The intermediate classes in the inheritance hierarchy, namely RegistryObject and
3225 RegistryEntry, map to relational views.

3226 ?? The names of relational tables and views are the same as the corresponding [ebRIM] class
3227 name. However, the name binding is case insensitive.

3228 ?? Each [ebRIM] class that maps to a table in Appendix 0 includes column definitions in
3229 Appendix 0 where the column definitions are based on a subset of attributes defined for that
3230 class in [ebRIM]. The attributes that map to columns include the inherited attributes for the
3231 [ebRIM] class. Comments in Appendix 0 indicate which ancestor class contributed which
3232 column definitions.

3233 An SQLQuery against a table not defined in Appendix 0 may raise an error condition:
3234 InvalidQueryException.

3235 The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn
3236 definitions.

3237 **8.3.1.2 Primitive Attributes Binding**

3238 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same
3239 way as column names in SQL. Again the exact attribute names are defined in the class
3240 definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is
3241 therefore valid for a query to contain attribute names that do not exactly match the case defined
3242 in [ebRIM].

3243 **8.3.1.3 Reference Attribute Binding**

3244 A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a
3245 class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject
3246 class returns a reference to an instance of an AccessControlPolicy object.

3247 In such cases the reference maps to the `id` attribute for the referenced object. The name of the
3248 resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data
3249 type for the column is VARCHAR(64) as defined in Appendix 0.

3250 When a reference attribute value holds a null reference, it maps to a null value in the SQL
3251 binding and may be tested with the `<null specification>` (“IS [NOT] NULL” syntax) as defined
3252 by [SQL].

3253 Reference attribute binding is a special case of a primitive attribute mapping.

3254 **8.3.1.4 Complex Attribute Binding**

3255 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of
3256 a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type
3257 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

3258 The SQL query schema does not map complex attributes as columns in the table for the class for
3259 which the attribute is defined. Instead the complex attributes are mapped to columns in the table
3260 for the domain class that represents the data type for the complex attribute (e.g.
3261 TelephoneNumber). A column links the row in the domain table to the row in the parent table
3262 (e.g. User). An additional column named ‘attribute_name’ identifies the attribute name in the
3263 parent class, in case there are multiple attributes with the same complex attribute type.

3264 This mapping also easily allows for attributes that are a collection of a complex type. For
3265 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the
3266 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier
3267 and an attribute_name.

3268 **8.3.1.5 Binding of Methods Returning Collections**

3269 Several of the [ebRIM] classes define methods in addition to attributes, where these methods
3270 return collections of references to instances of classes defined by [ebRIM]. For example, the
3271 getPackages method of the ManagedObject class returns a Collection of references to instances
3272 of Packages that the object is a member of.

3273 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in
3274 Appendix 0 such that these stored procedures return a collection of `id` attribute values. The
3275 returned value of these stored procedures can be treated as the result of a table sub-query in SQL.
3276 These stored procedures may be used as the right-hand-side of an SQL IN clause to test for
3277 membership of an object in such collections of references.

3278 **8.3.2 Semantic Constraints On Query Syntax**

3279 This section defines simplifying constraints on the query syntax that cannot be expressed in the

- 3280 BNF for the query syntax. These constraints must be applied in the semantic analysis of the
3281 query.
- 3282 1. Class names and attribute names must be processed in a case insensitive manner.
 - 3283 2. The syntax used for stored procedure invocation must be consistent with the syntax of an
3284 SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
 - 3285 3. For this version of the specification, the SQL select column list consists of exactly one
3286 column, and must always be t.i.d, where t is a table reference in the FROM clause.
 - 3287 4. Join operations must be restricted to simple joins involving only those columns that have an
3288 index defined within the normative SQL schema. This constraint is to prevent queries that
3289 may be computationally too expensive.

3290 **8.3.3 SQL Query Results**

3291 The result of an SQL query resolves to a collection of objects within the registry. It never
3292 resolves to partial attributes. The objects related to the result set may be returned as an
3293 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the
3294 responseOption parameter specified by the client on the AdHocQueryRequest. The entire result
3295 set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section **Error!**
3296 **Reference source not found..**

3297 **8.3.4 Simple Metadata Based Queries**

3298 The simplest form of an SQL query is based upon metadata attributes specified for a single class
3299 within [ebRIM]. This section gives some examples of simple metadata based queries.

3300 For example, to get the collection of ExtrinsicObjects whose name contains the word ‘Acme’
3301 and that have a version greater than 1.3, the following query must be submitted:

```
3302 SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND  
3303 eo.id = nm.parent AND  
3304 eo.majorVersion >= 1 AND  
3305 (eo.majorVersion >= 2 OR eo.minorVersion > 3);  
3306  
3307
```

3308 Note that the query syntax allows for conjugation of simpler predicates into more complex
3309 queries as shown in the simple example above.

3310 **8.3.5 RegistryObject Queries**

3311 The schema for the SQL query defines a special view called RegistryObject that allows doing a
3312 polymorphic query against all RegistryObject instances regardless of their actual concrete type or
3313 table name.

3314 The following example is the similar to that in Section 8.3.4 except that it is applied against all
3315 RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id
3316 for all qualifying RegistryObject instances whose name contains the word ‘Acme’ and whose
3317 description contains the word “bicycle”.

```
3318 SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND  
3319 d.value LIKE '%bicycle%' AND  
3320 ro.id = nm.parent AND ro.id = d.parent;  
3321  
3322
```


3323 **8.3.6 RegistryEntry Queries**

3324 The schema for the SQL query defines a special view called RegistryEntry that allows doing a
3325 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or
3326 table name.

3327 The following example is the same as Section 8.3.4 except that it is applied against all
3328 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id
3329 for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a
3330 version greater than 1.3.

```
3331
3332 SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
3333 re.id = nm.parent AND
3334 re.majorVersion >= 1 AND
3335 (re.majorVersion >= 2 OR re.minorVersion > 3);
3336
```

3337 **8.3.7 Classification Queries**

3338 This section describes the various classification related queries that must be supported.

3339 **8.3.7.1 Identifying ClassificationNodes**

3340 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may
3341 also be identified as a path attribute that specifies an XPATH expression [XPT] from a root
3342 classification node to the specified classification node in the XML document that would
3343 represent the ClassificationNode tree including the said ClassificationNode.

3344 **8.3.7.2 Getting ClassificationSchemes**

3345 To get the collection of ClassificationSchemes the following query predicate must be supported:

```
3346
3347 SELECT scheme.id FROM ClassificationScheme scheme;
3348
```

3349 The above query returns all ClassificationSchemes. Note that the above query may also specify
3350 additional predicates (e.g. name, description etc.) if desired.

3351 **8.3.7.3 Getting Children of Specified ClassificationNode**

3352 To get the children of a ClassificationNode given the ID of that node the following style of query
3353 must be supported:

```
3354
3355 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
3356
```

3357 The above query returns all ClassificationNodes that have the node specified by <id> as their
3358 parent attribute.

3359 **8.3.7.4 Getting Objects Classified By a ClassificationNode**

3360 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the
3361 following style of query must be supported:

```
3362
3363 SELECT id FROM ExtrinsicObject
3364 WHERE
3365 id IN (SELECT classifiedObject FROM Classification
3366 WHERE
3367 classificationNode IN (SELECT id FROM ClassificationNode
3368 WHERE path = '/Geography/Asia/Japan'))
3369 AND
3370 id IN (SELECT classifiedObject FROM Classification
3371 WHERE
3372 classificationNode IN (SELECT id FROM ClassificationNode
3373 WHERE path = '/Industry/Automotive'))
```

3374

3375 The above query gets the collection of ExtrinsicObjects that are classified by the Automotive
3376 Industry and the Japan Geography. Note that according to the semantics defined for
3377 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by
3378 descendents of the specified ClassificationNodes.

3379 **8.3.7.5 Getting Classifications That Classify an Object**

3380 To get the collection of Classifications that classify a specified Object the following style of
3381 query must be supported:

3382
3383
3384
3385

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

3386 **8.3.8 Association Queries**

3387 This section describes the various Association related queries that must be supported.

3388 **8.3.8.1 Getting All Association With Specified Object As Its Source**

3389 To get the collection of Associations that have the specified Object as its source, the following
3390 query must be supported:

3391
3392
3393

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

3394 **8.3.8.2 Getting All Association With Specified Object As Its Target**

3395 To get the collection of Associations that have the specified Object as its target, the following
3396 query must be supported:

3397
3398
3399

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

3400 **8.3.8.3 Getting Associated Objects Based On Association Attributes**

3401 To get the collection of Associations that have specified Association attributes, the following
3402 queries must be supported:

3403 Select Associations that have the specified name.

3404
3405
3406

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

3407 Select Associations that have the specified association type, where association type is a string
3408 containing the corresponding field name described in [ebRIM].

3409
3410
3411
3412

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

3413 **8.3.8.4 Complex Association Queries**

3414 The various forms of Association queries may be combined into complex predicates. The
3415 following query selects Associations that have a specific sourceObject, targetObject and
3416 associationType:

3417
3418
3419
3420
3421
3422

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

3423 8.3.9 Package Queries

3424 To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
3425  
3426 SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));  
3427
```

3428 8.3.9.1 Complex Package Queries

3429 The following query gets all Packages that a specified object belongs to, that are not deprecated
3430 and where name contains "RosettaNet."

```
3431  
3432 SELECT id FROM Package p, Name n WHERE  
3433 p.id IN (RegistryObject_packages(<id>)) AND  
3434 nm.value LIKE '%RosettaNet%' AND nm.parent = p.id AND  
3435 p.status <> 'Deprecated'  
3436
```

3437 8.3.10 ExternalLink Queries

3438 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is
3439 specified:

```
3440  
3441 SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))  
3442
```

3443 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is
3444 specified:

```
3445  
3446 SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))  
3447
```

3448 8.3.10.1 Complex ExternalLink Queries

3449 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that
3450 contain the word 'legal' in their description and have a URL for their externalURI.

```
3451  
3452 SELECT id FROM ExternalLink WHERE  
3453 id IN (RegistryObject_externalLinks(<id>)) AND  
3454 description LIKE '%legal%' AND  
3455 externalURI LIKE '%http://%'  
3456
```

3457 8.3.11 Audit Trail Queries

3458 To get the complete collection of AuditableEvent objects for a specified ManagedObject, the
3459 following query is specified:

```
3460  
3461 SELECT id FROM AuditableEvent WHERE registryObject = <id>  
3462
```

3463 8.4 Content Retrieval

3464 A client retrieves content via the Registry by sending the GetContentRequest to the
3465 QueryManager. The GetContentRequest specifies a list of Object references for Objects that
3466 need to be retrieved. The QueryManager returns the specified content by sending a
3467 GetContentResponse message to the RegistryClient interface of the client. If there are no errors
3468 encountered, the GetContentResponse message includes the specified content as additional
3469 payloads within the message. In addition to the GetContentResponse payload, there is one
3470 additional payload for each content that was requested. If there are errors encountered, the
3471 RegistryResponse payload includes an error and there are no additional content specific
3472 payloads.

3473 8.4.1 Identification Of Content Payloads

3474 Since the GetContentResponse message may include several repository items as additional
 3475 payloads, it is necessary to have a way to identify each payload in the message. To facilitate this
 3476 identification, the Registry must do the following:

3477 ?? Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mime-
 3478 part that contains the corresponding repository item for the ExtrinsicObject

3479 ?? In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the
 3480 repository item in the Reference element for that object in the Manifest element of the
 3481 ebXMLHeader.

3482 8.4.2 GetContentResponse Message Structure

3483 The following message fragment illustrates the structure of the GetContentResponse Message
 3484 that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs
 3485 for the requested objects.

```

3486 Content-type: multipart/related; boundary="Boundary"; type="text/xml";
3487
3488 --Boundary
3489 Content-ID: <GetContentRequest@example.com>
3490 Content-Type: text/xml
3491
3492 <?xml version="1.0" encoding="UTF-8"?>
3493 <SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
3494   xmlns:eb='http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
3495 <SOAP-ENV:Header>
3496
3497   ...ebMS header goes here if using ebMS
3498
3499 </SOAP-ENV:Header>
3500 <SOAP-ENV:Body>
3501
3502   ...ebMS manifest goes here if using ebMS
3503
3504 <?xml version="1.0" encoding="UTF-8"?>
3505
3506 <GetContentRequest>
3507   <ObjectRefList>
3508     <ObjectRef id="d8163dfb-f45a-4798-81d9-88aca29c24ff" .../>
3509     <ObjectRef id="212c3a78-1368-45d7-acc9-a935197e1e4f" .../>
3510   </ObjectRefList>
3511 </GetContentRequest>
3512
3513 </SOAP-ENV:Body>
3514 </SOAP-ENV:Envelope>
3515
3516 --Boundary
3517 Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
3518 Content-Type: text/xml
3519
3520 <?xml version="1.0" encoding="UTF-8"?>
3521 <CPP>
3522   ....
3523 </CPP>
3524
3525 --Boundary--
3526 Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f
3527 Content-Type: text/xml
3528
3529 <CPP>
3530   ....
3531 </CPP>
3532
3533 --Boundary--
3534
3535
3536
```

3537 **9 Registry Security**

3538 This chapter describes the security features of the ebXML Registry. It is assumed that the reader
3539 is familiar with the security related classes in the Registry information model as described in
3540 [ebRIM]. Security glossary terms can be referenced from RFC 2828.

3541 **9.1 Security Concerns**

3542 The security risks broadly stem from the following concerns. After a description of these
3543 concerns and potential solutions, we identify the concerns that we address in the current
3544 specification

- 3545 1. Is the content of the registry (data) trustworthy?
 - 3546 a) How to make sure “what is in the registry” is “what is put there” by a submitting
3547 organization? This concern can be addressed by ensuring that the publisher is
3548 authenticated using digital signature (Source Integrity), message is not corrupted during
3549 transfer using digital signature (Data Integrity), and the data is not altered by
3550 unauthorized subjects based on access control policy (Authorization)
 - 3551 b) How to protect data while in transmission?
3552 Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data
3553 Confidentiality that can be addressed by encrypting the data in transmission. How to
3554 protect against a replay attack.
 - 3555 c) Is the content up to date? The versioning as well as any time stamp processing, when
3556 done securely will ensure the “latest content” is guaranteed to be the latest content.
 - 3557 d) How to ensure only bona fide responsible organizations add contents to registry?
3558 Ensuring Source Integrity (as in 1a).
 - 3559 e) How to ensure that bona fide publishers add contents to registry only at authorized
3560 locations? (System Integrity)
 - 3561 f) What if the publishers deny modifying certain content after-the-fact? To prevent this
3562 (Nonrepudiation) audit trails may be kept which contain signed message digests.
 - 3563 g) What if the reader denies getting information from the registry?
- 3564 2. How to provide selective access to registry content? The broad answer is, by using an access
3565 control policy – applies to (a), (b), and (c) directly.
 - 3566 a) How does a submitting organization restrict access to the content to only specific registry
3567 readers?
 - 3568 b) How can a submitting organization allow some “partners” (fellow publishers) to modify
3569 content?
 - 3570 c) How to provide selective access to partners the registry usage data?
 - 3571 d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw
3572 failure of the registry security components? The solution to this problem is by having
3573 System Integrity.
 - 3574 e) Data confidentiality of RegistryObject

- 3575 3. How do we make “who can see what” policy itself visible to limited parties, even excluding
3576 the administrator (self & confidential maintenance of access control policy). By making sure
3577 there is an access control policy for accessing the policies themselves.
- 3578 4. How to transfer credentials? The broad solution is to use credentials assertion (such as being
3579 worked on in SAML). Currently, Registry does not support the notion of a session.
3580 Therefore, some of these concerns are not relevant to the current specification.
- 3581 a) How to transfer credentials (authorization/authentication) to federated registries?
3582 b) How do aggregators get credentials (authorization/authentication) transferred to them?
3583 c) How to store credentials through a session?
- 3584 In the current version of this specification, we address data integrity, source integrity (item 1,
3585 above). We have used a minimalist approach to address the access control concern as in item 2,
3586 above. Essentially, “any *known* entity (Submitting Organization) can publish content and *anyone*
3587 can view published content.” The Registry information model has been designed to allow more
3588 sophisticated security policies in future versions of this specification.

3589 **9.2 Integrity of Registry Content**

3590 It is assumed that most business registries do not have the resources to validate the veracity of
3591 the content submitted to them. Registry must ensure that any tampering to the content submitted
3592 by a Submitting Organization (SO) can be detected. Furthermore, Registry must make it possible
3593 to identify the Responsible Organization for any Registry content unambiguously. Note that in
3594 the discussions in this section we assume a Submitting Organization to be also the Responsible
3595 Organization. Future version of this specification may provide more examples and scenarios
3596 where a Submitting Organization and Responsible Organization are different.

3597 **9.2.1 Message Payload Signature**

3598 Integrity of Registry content requires that all submitted content be signed by the Registry client.
3599 The signature on the submitted content ensures that:

- 3600 ?? Any tampering of the content can be detected.
3601 ?? The content’s veracity can be ascertained by its association with a specific Submitting
3602 Organization.

3603 This section specifies the requirements for generation, packaging and validation of payload
3604 signatures. A payload signature is packaged with the payload. Therefore the requirements apply
3605 regardless of whether the Registry Client and the Registration Authority communicate over
3606 vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML
3607 Messaging Service does not specify the generation, validation and packaging of payload
3608 signatures. The specification of payload signatures is left upto the application (such as Registry).
3609 So the requirements on the payload signatures augment the [ebMS] specification.

3610 **Use Case**

3611 This Use Case illustrates the use of header and payload signatures (we discuss header signatures
3612 later).

- 3613 ?? RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the
3614 content along with the payload signature to the Registry.
3615 ?? RC2 (Registry Client 2) retrieves RC1’s content from the Registry.
3616 ?? RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
3617 the content, the response from the Registration Authority to RC2 contains the following:

- 3618 ?? Payload containing the content that has been published by RC1.
- 3619 ?? RC1's payload signature (represented by a ds:Signature element) over RC1's published
3620 content.
- 3621 ?? The public key for validating RC1's payload signature in ds:Signature element (using the
3622 KeyInfo element as specified in [XMLDSIG]) so RC2 can obtain the public key for
3623 signature (e.g. retrieve a certificate containing the public key for RC1).
- 3624 ?? A ds:Signature element containing the header signature. Note that the Registration
3625 Authority (not RC1) generates this signature.

3626 9.2.2 Payload Signature Requirements

3627 9.2.2.1 Payload Signature Packaging Requirements

3628 A payload signature is represented by a ds:Signature element. The payload signature must be
3629 packaged with the payload as specified here. This packaging assumes that the payload is always
3630 signed.

3631 ?? The payload and its signature must be enclosed in a MIME multipart message with a
3632 Content-Type of multipart/Related.

3633 ?? The first body part must contain the XML signature as specified in the section "Payload
3634 Signature Generation Requirements".

3635 ?? The second through nth body part must be the content.

3636 The packaging of the payload signature with one payload is as follows:

```
3637
3638 MIME-Version: 1.0
3639 Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
3640 Content-Description: ebXML Message
3641
3642 -- MIME_boundary
3643 Content-Type: text/xml; charset=UTF-8
3644 Content-Transfer-Encoding: 8bit
3645 Content-ID: http://claiming-it.com/claim061400a.xml
3646
3647 <?xml version='1.0' encoding="utf-8"?>
3648 <SOAP-ENV: Envelope>
3649 ...
3650 SOAP-ENV: Envelope>
3651
3652 --MIME_boundary
3653 Content-Type: multipart/Related; boundary=PAYLOAD_boundary
3654
3655 --PAYLOAD_boundary
3656 Content-Type: text/xml; charset=UTF-8
3657 Content-Transfer-Encoding: 8bit
3658 Content-ID: payload1
3659 <ds:Signature>
3660 ... Payload signature
3661 </ds: Signature>
3662
```

```

3663 --PAYLOAD_boundary
3664 Content-Type: text/xml; charset=UTF-8
3665 Content-Transfer-Encoding: 8bit
3666 Content-ID: payload2
3667 <SubmitObjectsRequest>...</SubmitObjectsRequest>
3668 --MIME_boundary
3669

```

3670 9.2.2.2 Payload Signature Generation Requirements

3671 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in
 3672 this section. Note: the “ds” name space reference is to <http://www.w3.org/2000/09/xmlsig#>

3673 ?? ds:SignatureMethod must be present. The signing algorithm can be valid any algorithm
 3674 permitted in [XMLDSIG], though we suggest using the following Algorithm attribute while
 3675 signing for interoperability: <http://www.w3.org/2000/09/xmlsig/#dsa-sha1>

3676 ?? The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. . The
 3677 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
 3678 <http://www.w3.org/TR/2001/REC-xml-c14n-2001315>

3679 ?? One ds:Reference element to reference each of the payloads that needs to be signed must be
 3680 created. The ds:Reference element:

3681 ?? Must identify the payload to be signed using the URI attribute of the ds:Reference
 3682 element.

3683 ?? Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be
 3684 support the following digest algorithm:

3685 <http://www.w3.org/2000/09/xmlsig/#sha1>

3686 ?? Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

3687 The ds:SignedValue must be generated as specified in [XMLDSIG].

3688 The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject
 3689 to the requirements stated in the “KeyDistribution and KeyInfo element” section of this
 3690 document.

3691 9.2.2.3 Message Payload Signature Validation

3692 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

3693 9.2.2.4 Payload Signature Example

3694 The following example shows the format of the payload signature:

```

3695 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmlsig#">
3696 <ds:SignedInfo>
3697   <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmlsig/#dsa-sha1" />
3698   <ds:CanonicalizationMethod>
3699     Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
3700   </ds:CanonicalizationMethod>
3701   <ds:Reference URI=#Payload1>
3702     <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmlsig/#sha1">
3703     <ds:DigestValue> ... </ds:DigestValue>
3704   </ds:Reference>
3705 </ds:SignedInfo>
3706

```



```
3707 <ds:SignatureValue> ... </ds:SignatureValue>
3708 </ds:Signature>
3709
```

3710 9.3 Authentication

3711 The Registry must be able to authenticate the identity of the Principal associated with client
3712 requests. *Authentication* is required to identify the ownership of content as well as to identify
3713 what “privileges” a Principal can be assigned with respect to the specific objects in the Registry.

3714 The Registry must perform authentication on a per message basis. From a security point of view,
3715 all messages are independent and there is no concept of a session encompassing multiple
3716 messages or conversations. Session support may be added as an optimization feature in future
3717 versions of this specification.

3718 It is important to note that the message header signature can only guarantee data integrity and it
3719 may be used for Authentication knowing that it is vulnerable to replay types of attacks. True
3720 support for authentication requires timestamps or nonce (nonrecurring series of numbers to
3721 identify each message) that are signed.

3722 9.3.1 Message Header Signature

3723 Message headers are signed to provide data integrity while the message is in transit. Note that the
3724 signature within the message header also signs the digests of the payloads.

3725 Header Signature Requirements

3726 Message headers can be signed and are referred to as a header signature. This section specifies
3727 the requirements for generation, packaging and validation of a header signature. These
3728 requirements apply when the Registry Client and Registration Authority communicate using
3729 vanilla SOAP with Attachments. When ebXML MS is used for communication, then the [ebMS]
3730 specifies the generation, packaging and validation of XML signatures in the SOAP header.
3731 Therefore the header signature requirements do not apply when the ebXML MS is used for
3732 communication. However, payload signature generation requirements (specified elsewhere in
3733 this document) do apply whether vanilla SOAP with Attachments or ebXML MS is used for
3734 communication.

3735 9.3.1.1 Packaging Requirements

3736 A header signature is represented by a ds:Signature element. The ds:Signature element generated
3737 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature
3738 element in the SOAP header field is shown below.
3739

```
3740 MIME-Version: 1.0
3741 Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3742 Content-Description: ebXML Message
3743
3744 -- MIME_boundary
3745 Content-Type: text/xml; charset=UTF-8
3746 Content-Transfer-Encoding: 8bit
3747 Content-ID: http://claiming-it.com/claim061400a.xml
3748
3749 <?xml version='1.0' encoding="utf-8"?>
3750 <SOAP-ENV:Envelope
```

```

3751 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3752 <SOAP-ENV:Header>
3753   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3754     ...signature over soap envelope
3755   </ds:Signature>
3756 </SOAP-ENV:Header>
3757 <SOAP-ENV:Body>
3758   ...
3759 </SOAP-ENV:Body>
3760 </SOAP-ENV:Envelope>
3761

```

3762 9.3.1.2 Header Signature Generation Requirements

3763 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in
 3764 this section. A ds:Signature element contains:

- 3765 ?? ds:SignedInfo
- 3766 ?? ds:SignatureValue
- 3767 ?? ds:KeyInfo

3768 The ds:SignedInfo element must be generated as follows:

- 3769 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
 3770 using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,
 3771 a client must be capable of signing using only the following Algorithm attribute:
 3772 <http://www.w3.org/2000/09/xmldsig/#dsa-sha1> This algorithm is being chosen because all
 3773 XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
 - 3774 2. The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The
 3775 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
 3776 <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>
 - 3777 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.
 3778 This signs the entire ds:Reference element and:
 - 3779 ?? Must include the following ds:Transform:
 - 3780 <http://www.w3.org/2000/09/xmldsig#enveloped-signature>
 - 3781 This ensures that the signature (which is embedded in the <SOAP-ENV:Header>
 - 3782 element) is not included in the signature calculation.
 - 3783 ?? Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the
 - 3784 ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) .
 - 3785 The URI attribute must be "".
 - 3786 ?? Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support
 - 3787 the following digest algorithm: <http://www.w3.org/2000/09/xmldsig/#sha1>
 - 3788 ?? Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].
- 3789 The ds:SignedValue must be generated as specified in [XMLDSIG].
- 3790 The ds:KeyInfo element may be present But when present, it is subject to the requirements stated
 3791 in the "KeyDistribution and KeyInfo element" section of this document.

3792 9.3.1.3 Header Signature Validation Requirements

3793 The ds:Signature element for the ebXML message header must be validated by the recipient as

3794 specified by [XMLDSIG].

3795 9.3.1.4 Header Signature Example

3796 The following example shows the format of a header signature:

3797

3798 `<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">`

3799 `<ds:SignedInfo>`

3800 `<SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>`

3801 `<ds:CanonicalizationMethod>`

3802 `Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">`

3803 `</ds:CanonicalizationMethod>`

3804 `<ds:Reference URI= "">`

3805 `<ds:Transform>`

3806 `http://www.w3.org/2000/09/xmldsig#enveloped-signature`

3807 `</ds:Transform>`

3808 `<ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">`

3809 `<ds:DigestValue> ... </ds:DigestValue>`

3810 `</ds:Reference>`

3811 `</ds:SignedInfo>`

3812 `<ds:SignatureValue> ... </ds:SignatureValue>`

3813 `</ds:Signature>`

3814

3815 9.4 Key Distribution and KeyInfo Element

3816 To validate a signature, the recipient of the signature needs the public key corresponding to the
 3817 signer's public key. The participants may use the KeyInfo field of ds:Signature, or distribute the
 3818 public keys out-of-band. In this section we consider the case when the public key is sent in the
 3819 KeyInfo field. The following use cases need to be handled:

3820 ?? Registration Authority needs the public key of the Registry Client to validate the signature

3821 ?? Registry Client needs the public key of the Registration Authority to validate the Registry's
 3822 signature.

3823 ?? Registry Client RC1 needs the public key of Registry Client (RC2) to validate the content
 3824 signed by RC1.

3825 [XMLDSIG] provides a ds:KeyInfo element that can be used to pass the recipient information
 3826 for retrieving the public key. ds:KeyInfo is an optional element as specified in [XMLDSIG].

3827 This field together with the procedures outlined in this section is used to securely pass the public
 3828 key to a recipient. ds:Keyinfo can be used to pass information such as keys, certificates, names
 3829 etc. The intended usage of KeyInfo field is to send the X509 Certificate, and subsequently
 3830 extract the public key from the certificate. Therefore, the KeyInfo field must contain a X509
 3831 Certificate, if the KeyInfo field is present.

3832 The following assumptions are also made:

3833 1. A Certificate is associated both with the Registration Authority and a Registry Client.

3834 2. A Registry Client registers its certificate with the Registration Authority. The mechanism
 3835 used for this is not specified here.

3836 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local
 3837 key store. The mechanism is not specified here.

3838 Couple of scenarios on the use of KeyInfo field is in Appendix F.8.

3839 **9.5 Confidentiality**

3840 **9.5.1 On-the-wire Message Confidentiality**

3841 It is suggested but not required that message payloads exchanged between clients and the
3842 Registry be encrypted during transmission. Payload encryption must abide by any restrictions set
3843 forth in [SEC].

3844 **9.5.2 Confidentiality of Registry Content**

3845 In the current version of this specification, there are no provisions for confidentiality of Registry
3846 content. All content submitted to the Registry may be discovered and read by *any* client. This
3847 implies that the Registry and the client need to have an a priori agreement regarding encryption
3848 algorithm, key exchange agreements, etc. This service is not addressed in this specification.

3849 **9.6 Authorization**

3850 The Registry must provide an authorization mechanism based on the information model defined
3851 in [ebRIM]. In this version of the specification the authorization mechanism is based on a default
3852 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of
3853 this specification will allow for custom Access Control Policies to be defined by the Submitting
3854 Organization. The authorization is going to be applied on a specific set of privileges. A
3855 privilege is the ability to carry a specific action.

3856 **9.6.1 Actions**

3857 Life Cycle Actions

3858 submitObjects
3859 updateObjects
3860 addSlots
3861 removeSlots
3862 approveObjects
3863 deprecateObjects
3864 removeObjects

3865 Read Actions

3866 The various getXXX() methods in QueryManagement Service.

3867 **9.7 Access Control**

3868 The Registry must create a default AccessControlPolicy object that grants the default
3869 permissions to Registry users based upon their assigned role. The following table defines the
3870 Permissions granted by the Registry to the various pre-defined roles for Registry users based
3871 upon the default AccessControlPolicy. Note that we have “ContentOwner” as a role. This role
3872 maps to the Submitting Organization in the current version of the specification.

3873

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).

- 3874 The following list summarizes the default role-based AccessControlPolicy:
- 3875 ?? The Registry must implement the default AccessControlPolicy and associate it with all
3876 Objects in the Registry
- 3877 ?? Anyone can publish content, but needs to be a Registered User
- 3878 ?? Anyone can access the content without requiring authentication
- 3879 ?? The ContentOwner has access to all methods for Registry Objects created by it.
- 3880 ?? The RegistryAdministrator has access to all methods on all Registry Objects
- 3881 ?? Unauthenticated clients can access all read-only (getXXX) methods
- 3882 ?? At the time of content submission, the Registry must assign the default ContentOwner role to
3883 the Submitting Organization (SO) as authenticated by the credentials in the submission
3884 message. In the current version of this specification, the Submitting Organization will be the
3885 DN as identified by the certificate
- 3886 ?? Clients that browse the Registry need not use certificates. The Registry must assign the
3887 default RegistryGuest role to such clients.

3888 **Appendix A Web Service Architecture**

3889 **Registry Service Abstract Specification**

3890 The normative definition of the Abstract Registry Service in WSDL is defined at the following
3891 location on the web:

3892

3893 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/services/Registry.wsdl>

3894

3895 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

3896 **Registry Service SOAP Binding**

3897 The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined
3898 at the following location on the web:

3899

3900 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/services/RegistrySOAPBinding.wsdl>

3901

3902 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

3903 **Appendix B ebXML Registry Schema Definitions**

3904 **RIM Schema**

3905 The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the
3906 following location on the web:

3907

3908 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/schema/rim.xsd>

3909

3910 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

3911 **Query Schema**

3912 The normative XML Schema definition for the XML query syntax for the registry service
3913 interface can be found at the following location on the web:

3914

3915 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/schema/query.xsd>

3916

3917 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

3918 **Registry Services Interface Schema**

3919 The normative XML Schema definition that defines the XML requests and responses supported
3920 by the registry service interfaces in this document can be found at the following location on the
3921 web:

3922

3923 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/schema/rs.xsd>

3924

3925 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

3926 **Examples of Instance Documents**

3927 A growing number of non-normative XML instance documents that conform to the normative
3928 Schema definitions described earlier may be found at the following location on the web:

3929

3930 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/>

3931

3932 **Appendix C Interpretation of UML Diagrams**

3933 This section describes in *abstract terms* the conventions used to define ebXML business process
3934 description in UML.

3935 **UML Class Diagram**

3936 A UML class diagram is used to describe the Service Interfaces required to implement an
3937 ebXML Registry Services and clients. The UML class diagram contains:

- 3938
- 3939 1. A collection of UML interfaces where each interface represents a Service Interface for a
3940 Registry service.
 - 3941 2. Tabular description of methods on each interface where each method represents an
3942 Action (as defined by [ebCPP]) within the Service Interface representing the UML
3943 interface.
 - 3944 3. Each method within a UML interface specifies one or more parameters, where the type of
3945 each method argument represents the ebXML message type that is exchanged as part of
3946 the Action corresponding to the method. Multiple arguments imply multiple payload
3947 documents within the body of the corresponding ebXML message.

3948 **UML Sequence Diagram**

3949 A UML sequence diagram is used to specify the business protocol representing the interactions
3950 between the UML interfaces for a Registry specific ebXML business process. A UML sequence
3951 diagram provides the necessary information to determine the sequencing of messages, request to
3952 response association as well as request to error response association.

3953 Each sequence diagram shows the sequence for a specific conversation protocol as method calls
3954 from the requestor to the responder. Method invocation may be synchronous or asynchronous
3955 based on the UML notation used on the arrow-head for the link. A half arrow-head represents
3956 asynchronous communication. A full arrow-head represents synchronous communication.

3957 Each method invocation may be followed by a response method invocation from the responder to
3958 the requestor to indicate the ResponseName for the previous Request. Possible error response is
3959 indicated by a conditional response method invocation from the responder to the requestor. See
3960 Figure 7 on page 26 for an example.

3961 Appendix D SQL Query

3962 SQL Query Syntax Specification

3963 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The
 3964 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax
 3965 conforms to the <query specification>, modulo the restrictions identified below:

- 3966 1. A <select list> may contain at most one <select sublist>.
- 3967 2. In a <select list> must be is a single column whose data type is UUID, from the table in the
 3968 <from clause>.
- 3969 3. A <derived column> may not have an <as clause>.
- 3970 4. <table expression> does not contain the optional <group by clause> and <having clause>
 3971 clauses.
- 3972 5. A <table reference> can only consist of <table name> and <correlation name>.
- 3973 6. A <table reference> does not have the optional AS between <table name> and
 3974 <correlation name>.
- 3975 7. There can only be one <table reference> in the <from clause>.
- 3976 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows
 3977 for the right hand side of the <in predicate> to be limited to a restricted <query
 3978 specification> as defined above.
- 3979 9. A <search condition> within the <where clause> may not include a <query expression>.
- 3980 10. Simple joins are allowed only if they are based on indexed columns within the relational
 3981 schema.
- 3982 11. The SQL query syntax allows for the use of <sql invoked routines> invocation from
 3983 [SQL/PSM] as the RHS of the <in predicate>.

3984 Non-Normative BNF for Query Syntax Grammar

3985 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as
 3986 an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative
 3987 syntax. For the normative syntax rules see Appendix 0.

```

3988 /*****
3989 * The Registry Query (Subset of SQL-92) grammar starts here
3990 *****/
3991
3992 RegistryQuery = SQLSelect [ ";" ]
3993
3994 SQLSelect = "SELECT" [ "DISTINCT" ] SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
3995
3996 SQLSelectCols = ID
3997
3998 SQLTableList = SQLTableRef
3999
4000 SQLTableRef = ID
4001
4002 SQLWhere = "WHERE" SQLOrExpr
4003
4004 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*
4005
```

```

4006
4007 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr)*
4008
4009 SQLNotExpr = [ "NOT" ] SQLCompareExpr
4010
4011 SQLCompareExpr =
4012     (SQLColRef "IS") SQLIsClause
4013     | SQLSumExpr [ SQLCompareExprRight ]
4014
4015
4016 SQLCompareExprRight =
4017     SQLLikeClause
4018     | SQLInClause
4019     | SQLCompareOp SQLSumExpr
4020
4021 SQLCompareOp =
4022     "="
4023     | "<>"
4024     | ">"
4025     | ">="
4026     | "<"
4027     | "<="
4028
4029 SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
4030
4031 SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
4032
4033 SQLLValueElement = "NULL" | SQLSelect
4034
4035 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
4036
4037 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
4038
4039 SQLPattern = STRING_LITERAL
4040
4041 SQLLiteral =
4042     STRING_LITERAL
4043     | INTEGER_LITERAL
4044     | FLOATING_POINT_LITERAL
4045
4046 SQLColRef = SQLLvalue
4047
4048 SQLLvalue = SQLLvalueTerm
4049
4050 SQLLvalueTerm = ID ( "." ID )*
4051
4052 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
4053
4054 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr )*
4055
4056 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
4057
4058 SQLTerm = "(" SQLOrExpr ")"
4059     | SQLColRef
4060     | SQLLiteral
4061
4062 INTEGER_LITERAL = ([ "0"-"9" ])+
4063
4064 FLOATING_POINT_LITERAL =
4065     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ ( EXPONENT)?
4066     | "." ([ "0"-"9" ])+ ( EXPONENT)?
4067     | ([ "0"-"9" ])+ EXPONENT
4068     | ([ "0"-"9" ])+ ( EXPONENT)?
4069
4070 EXPONENT = [ "e", "E" ] ( [ "+" , "-" ] )? ([ "0"-"9" ])+
4071
4072 STRING_LITERAL: "'" (~[ "'" ])* ( "''" (~[ "'" ])* )* "'"
4073
4074 ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
4075 LETTER = [ "A"-"Z", "a"-"z" ]
4076 DIGIT = [ "0"-"9" ]

```

4077 **Relational Schema For SQL Queries**

4078 The normative Relational Schema definition for SQL queries can be found at the following
4079 location on the web:

4080

4081 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/sql/database.sql>

4082

4083 The stored procedures that must be supported by the SQL query feature are defined at the following
4084 location on the web:

4085 <http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/sql/storedProcedures.sql>

4086

4087 **In the final V2.0 version of this document the URL will point to an OASIS web site location.**

4088 **Appendix E Non-normative Content Based Ad Hoc Queries**

4089 The Registry SQL query capability supports the ability to search for content based not only on
4090 metadata that catalogs the content but also the data contained within the content itself. For
4091 example it is possible for a client to submit a query that searches for all Collaboration Party
4092 Profiles that define a role named “seller” within a RoleName element in the CPP document itself.
4093 Currently content-based query capability is restricted to XML content.

4094 **Automatic Classification of XML Content**

4095 Content-based queries are indirectly supported through the existing classification mechanism
4096 supported by the Registry.

4097 A submitting organization may define logical indexes on any XML schema or DTD when it is
4098 submitted. An instance of such a logical index defines a link between a specific attribute or
4099 element node in an XML document tree and a ClassificationNode in a classification scheme
4100 within the registry.

4101 The registry utilizes this index to automatically classify documents that are instances of the
4102 schema at the time the document instance is submitted. Such documents are classified according
4103 to the data contained within the document itself.

4104 Such automatically classified content may subsequently be discovered by clients using the
4105 existing classification-based discovery mechanism of the Registry and the query facilities of the
4106 QueryManager.

4107 [Note] This approach is conceptually similar to the way databases support
4108 indexed retrieval. DBAs define indexes on tables in the schema. When
4109 data is added to the table, the data gets automatically indexed.

4110 **Index Definition**

4111 This section describes how the logical indexes are defined in the SubmittedObject element
4112 defined in the Registry DTD. The complete Registry DTD is specified in Appendix A.

4113 A SubmittedObject element for a schema or DTD may define a collection of
4114 ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList
4115 is ignored if the content being submitted is not of the SCHEMA objectType.

4116 The ClassificationIndex element inherits the attributes of the base class RegistryObject in
4117 [ebRIM]. It then defines specialized attributes as follows:

- 4118 1. classificationNode: This attribute references a specific ClassificationNode by its ID.
- 4119 2. contentIdentifier: This attribute identifies a specific data element within the document
4120 instances of the schema using an XPATH expression as defined by [XPT].

4121 **Example Of Index Definition**

4122 To define an index that automatically classifies a CPP based upon the roles defined within its
4123 RoleName elements, the following index must be defined on the CPP schema or DTD:

```
4124 <ClassificationIndex  
4125     classificationNode='id-for-role-classification-scheme'  
4126     contentIdentifier='//Role//RoleName'  
4127 />  
4128  
4129
```

4130 Proposed XML Definition

```
4131 <!--  
4132 A ClassificationIndexList is specified on ExtrinsicObjects of objectType  
4133 'Schema' to define an automatic Classification of instance objects of the  
4134 schema using the specified classificationNode as parent and a  
4135 ClassificationNode created or selected by the object content as selected  
4136 by the contentIdentifier  
4137 -->  
4138 <!--  
4139 <!ELEMENT ClassificationIndex EMPTY>  
4140 <!ATTLIST ClassificationIndex  
4141     %ObjectAttributes;  
4142     classificationNode IDREF #REQUIRED  
4143     contentIdentifier CDATA #REQUIRED  
4144 >  
4145 <!-- ClassificationIndexList contains new ClassificationIndexes -->  
4146 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>  
4147  
4148
```

4149 Example of Automatic Classification

4150 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the CPP is
4151 submitted it will automatically be classified by two ClassificationNodes named “buyer” and
4152 “seller” that are both children of the ClassificationNode (e.g. a node named Role) specified in the
4153 classificationNode attribute of the ClassificationIndex. If either of the two ClassificationNodes
4154 named “buyer” and “seller” did not previously exist, the LifeCycleManager would automatically
4155 create these ClassificationNodes.

4156 **Appendix F Security Implementation Guideline**

4157 This section provides a suggested blueprint for how security processing may be implemented in
4158 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have
4159 different implementations as long as they support the default security roles and authorization
4160 rules described in this document.

4161 **Authentication**

- 4162 1. As soon as a message is received, the first work is the authentication. A principal object is
4163 created.
- 4164 2. If the message is signed, it is verified (including the validity of the certificate) and the DN of
4165 the certificate becomes the identity of the principal. Then the Registry is searched for the
4166 principal and if found, the roles and groups are filled in.
- 4167 3. If the message is not signed, an empty principal is created with the role RegistryGuest. This
4168 step is for symmetry and to decouple the rest of the processing.
- 4169 4. Then the message is processed for the command and the objects it will act on.

4170 **Authorization**

4171 For every object, the access controller will iterate through all the AccessControlPolicy objects
4172 with the object and see if there is a chain through the permission objects to verify that the
4173 requested method is permitted for the Principal. If any of the permission objects which the object
4174 is associated with has a common role, or identity, or group with the principal, the action is
4175 permitted.

4176 **Registry Bootstrap**

4177 When a Registry is newly created, a default Principal object should be created with the identity
4178 of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message
4179 signed by the Registry Admin will get all the privileges.

4180 When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the
4181 default AccessControlPolicy. This includes the creation of the necessary Permission instances as
4182 well as the Privileges and Privilege attributes.

4183 **Content Submission – Client Responsibility**

4184 The Registry client has to sign the contents before submission – otherwise the content will be
4185 rejected.

4186 **Content Submission – Registry Responsibility**

- 4187 1. As with any other request, the client will first be authenticated. In this case, the Principal
4188 object will get the DN from the certificate.
- 4189 2. As per the request in the message, the RegistryEntry will be created.
- 4190 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.

- 4191 4. If a principal with the identity of the SO is not available, an identity object with the SO's DN
4192 is created.
- 4193 5. A principal with this identity is created.

4194 **Content Delete/Deprecate – Client Responsibility**

4195 The Registry client has to sign the payload (not entire message) before submission, for
4196 authentication purposes; otherwise, the request will be rejected

4197 **Content Delete/Deprecate – Registry Responsibility**

- 4198 1. As with any other request, the client will first be authenticated. In this case, the Principal
4199 object will get the DN from the certificate. As there will be a principal with this identity in
4200 the Registry, the Principal object will get all the roles from that object
- 4201 2. As per the request in the message (delete or deprecate), the appropriate method in the
4202 RegistryObject class will be accessed.
- 4203 3. The access controller performs the authorization by iterating through the Permission objects
4204 associated with this object via the singleton default AccessControlPolicy.
- 4205 4. If authorization succeeds then the action will be permitted. Otherwise an error response is
4206 sent back with a suitable AuthorizationException error message.

4207 **Using ds:KeyInfo Field**

4208 Two typical usage scenarios for ds:KeyInfo are described below.

4209 **Scenario 1**

- 4210 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
- 4211 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
- 4212 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
- 4213 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature
- 4214 5. Registration Authority validates the header signature using the public key from the
4215 certificate.
- 4216 6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the
4217 certificate from the KeyInfo field of the payload signature. Note that this step is not an
4218 essential one if the onus of validation is that of the eventual user, another Registry Client, of
4219 the content.

4220 **Scenario 2**

- 4221 1. RC1 signs the payload and SOAP envelope using its private key and publishes to the
4222 Registry.
- 4223 2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
- 4224 3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature.
4225 This step is required in addition to step 2 because when RC2 retrieves content, it should see
4226 RC1's signature with the payload.
- 4227 4. RC2 retrieves content from the Registry.

-
- 4228 5. Registration Authority signs the SOAP envelope using its private key. Registration Authority
4229 sends RC1's content and the RC1's signature (signed by RC1).
- 4230 6. Registration Authority need not send its certificate in the KeyInfo field since RC2 is assumed
4231 to have obtained the Registration Authority's certificate out of band and installed it in its
4232 local key store.
- 4233 7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the
4234 Registration Authority's signature.
- 4235 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates
4236 the signature on the payload.

4237 **Appendix G Native Language Support (NLS)**

4238 **Definitions**

4239 Although this section discusses only character set and language, the following terms have to be
4240 defined clearly.

4241 **Coded Character Set (CCS):**

4242 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of
4243 CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

4244 **Character Encoding Scheme (CES):**

4245 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are
4246 ISO-2022, UTF-8.

4247 **Character Set (charset):**

4248 ?? charset is a set of rules for mapping from a sequence of octets to a sequence of
4249 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.

4250 ?? A list of registered character sets can be found at [IANA].

4251 **NLS And Request / Response Messages**

4252 For the accurate processing of data in both registry client and registry services, it is essential to
4253 know which character set is used. Although the body part of the transaction may contain the
4254 charset in xml encoding declaration, registry client and registry services shall specify charset
4255 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a
4256 text/xml entity is received with the charset parameter omitted, MIME processors and XML
4257 processors MUST use the default charset value of "us-ascii". For example:

```
4258  
4259 Content-Type: text/xml; charset=ISO-2022-JP  
4260
```

4261 Also, when an application/xml entity is used, the charset parameter is optional, and registry
4262 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which
4263 directly address this contingency.

4264 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

4265 **NLS And Storing of RegistryObject**

4266 This section provides NLS guidelines on how a registry should store *RegistryObject* instances.
4267 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple
4268 locales. Thus there is no language or character set associated with a specific RegistryObject
4269 instance.

4270 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows.
4271 Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in
4272 RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed
4273 as LocalizedString sub-elements within the XML element representing the I18N capable

4274 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a
4275 specific locale. Each LocalizedString element has a charset and lang attribute as well as a value
4276 attribute of type string.

4277 **Character Set of *LocalizedString***

4278 The character set used by a locale specific String (LocalizedString) is defined by the charset
4279 attribute. It is highly recommended to use UTF-8 or UTF-16 for maximum inter-operability.

4280 **Language Information of *LocalizedString***

4281 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

4282 **NLS And Storing of Repository Items**

4283 This section provides NLS guidelines on how a registry should store repository items.

4284 While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is
4285 always associated with a single repository item. The repository item may be in a single locale or
4286 may be in multiple locales. This specification does not specify the repository item.

4287 **Character Set of Repository Items**

4288 The MIME **Content-Type** mime header for the mime multi-part containing the repository
4289 item MAY contain a "charset" attribute that specifies the character set used by the repository
4290 item. For example:

4291

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

4292

4293

4294 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset
4295 of a repository item must be preserved as it is originally specified in the transaction.

4296 **Language information of repository item**

4297 The Content-language mime header for the mime bodypart containing the repository item may
4298 specify the language for a locale specific repository item. The value of the Content-language
4299 mime header property must conform to [RFC 1766].

4300 This document currently specifies only the method of sending the information of character set
4301 and language, and how it is stored in a registry. However, the language information may be used
4302 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a
4303 language negotiation procedure, like registry client is asking a favorite language for messages
4304 from registry services, could be another functionality for the future revision of this document.

4305 **Appendix H Terminology Mapping**

4306 While every attempt has been made to use the same terminology used in other works there are
 4307 some terminology differences. The following table shows the terminology mapping between this
 4308 specification and that used in other specifications and working groups.

4309 **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	

4310 References

- 4311 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 4312 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 4313 [TA] ebXML Technical Architecture
- 4314 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf
- 4315 [OAS] OASIS Information Model
- 4316 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 4317 [ISO] ISO 11179 Information Model
- 4318 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 4319
- 4320 [ebRIM] ebXML Registry Information Model
- 4321 http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
- 4322 [ebBPM] ebXML Business Process Specification Schema
- 4323 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 4324 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 4325 http://www.ebxml.org/project_teams/trade_partner/private/
- 4326 [ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
- 4327 <http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html>
- 4328 [CTB] Context table informal document from Core Components
- 4329 [ebMS] ebXML Messaging Service Specification, Version 0.21
- 4330 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
- 4331 [SEC] ebXML Risk Assessment Technical Report, Version 3.6
- 4332 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 4333 [XPT] XML Path Language (XPath) Version 1.0
- 4334 <http://www.w3.org/TR/xpath>
- 4335 [SQL] Structured Query Language (FIPS PUB 127-2)
- 4336 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 4337 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 4338 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 4339 [IANA] IANA (Internet Assigned Numbers Authority).
- 4340 Official Names for Character Sets, ed. Keld Simonsen et al.
- 4341 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 4342 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
- 4343 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
- 4344 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 4345 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
- 4346 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
- 4347 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 4348 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
- 4349 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
- 4350 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 4351 [RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
- 4352 Internet Security Glossary, ed. R. Shirey. May 2000.
- 4353 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html>

- 4354 [RFC 3023] ETF (Internet Engineering Task Force). RFC 3023:
4355 XML Media Types, ed. M. Murata. 2001.
4356 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
4357 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
4358 <http://www.w3.org/TR/REC-xml>
4359 [UUID] DCE 128 bit Universal Unique Identifier
4360 http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
4361 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
4362 [WSDL]W3C Note. Web Services Description Language (WSDL) 1.1
4363 <http://www.w3.org/TR/wsdl>
4364 [SOAP11]W3C Note. Simple Object Access Protocol, May 2000,
4365 <http://www.w3.org/TR/SOAP>
4366 [SOAPAt]W3C Note: SOAP with Attachments, Dec 2000,
4367 <http://www.w3.org/TR/SOAP-attachments>
4368 [XMLDSIG] XML-Signature Syntax and Processing,
4369 <http://www.w3.org/TR/2001/PR-xmlsig-core-20010820/>

4370 **Disclaimer**

- 4371 The views and specification expressed in this document are those of the authors and are not
4372 necessarily those of their employers. The authors and their employers specifically disclaim
4373 responsibility for any problems arising from correct or incorrect implementation or use of this
4374 design.

4375 Contact Information**4376 Team Leader**

4377 Name: Lisa Carnahan
4378 Company:
4379 Street:
4380 City, State, Postal Code:
4381 Country: USA
4382 Phone:
4383 Email: lisa.carnahan@nist.gov

4384

4385 Vice Team Lead

4386 Name: Yutaka Yoshida
4387 Company: Sun Microsystems
4388 Street: 901 San Antonio Road, MS UMPK17-102
4389 City, State, Postal Code: Palo Alto, CA 94303
4390 Country: USA
4391 Phone: 650.786.5488
4392 Email: Yutaka.Yoshida@eng.sun.com

4393

4394 Editor

4395 Name: Anne A. Fischer
4396 Company: Drummond Group, Inc.
4397 Street: 4700 Bryant Irvin Ct., Suite 303
4398 City, State, Postal Code: Fort Worth, Texas 76107-7645
4399 Country: USA
4400 Phone: 817-371-2367
4401 Email: anne@drummondgroup.com

4402

4403 **Copyright Statement**

4404 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

4405 This document and translations of it may be copied and furnished to others, and derivative works
4406 that comment on or otherwise explain it or assist in its implementation MAY be prepared,
4407 copied, published and distributed, in whole or in part, without restriction of any kind, provided
4408 that the above copyright notice and this paragraph are included on all such copies and derivative
4409 works. However, this document itself MAY not be modified in any way, such as by removing
4410 the copyright notice or references to ebXML, UN/CEFACT, or OASIS, except as required to
4411 translate it into languages other than English.

4412 The limited permissions granted above are perpetual and will not be revoked by ebXML or its
4413 successors or assigns.

4414 This document and the information contained herein is provided on an "AS IS" basis and
4415 ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT
4416 NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
4417 WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
4418 MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.