



Creating A Single Global Electronic Market

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23

# ebXML Registry Services v0.90

## ebXML Registry Project Team

23 April 2001

### 1 Status of this Document

This document specifies an ebXML DRAFT STANDARD for the eBusiness community.

Distribution of this document is unlimited.

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

***This version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpecificationv0.89.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.89.pdf)

***Latest version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpecification.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecification.pdf)

***Previous version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpecificationv0.88.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.88.pdf)

## 24 **2 ebXML participants**

25 ebXML Registry Services, v1.0 was developed by the ebXML Registry Project Team.  
26 At the time this specification was approved, the membership of the ebXML Registry  
27 Project Team was as follows:

28  
29 Lisa Carnahan, NIST  
30 Joe Dalman, Tie  
31 Philippe DeSmedt, Viquity  
32 Sally Fuger, AIAG  
33 Len Gallagher, NIST  
34 Steve Hanna, Sun Microsystems  
35 Scott Hinkelman, IBM  
36 Michael Kass, NIST  
37 Jong.L Kim, Innodigital  
38 Sangwon Lim, Korea Institute for Electronic Commerce  
39 Bob Miller, GXS  
40 Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan  
41 Dale Moberg, Sterling Commerce  
42 Ron Monzillo, Sun Microsystems  
43 JP Morgenthal, eThink Systems, Inc.  
44 Joel Munter, Intel  
45 Farrukh Najmi, Sun Microsystems  
46 Scott Nieman, Norstan Consulting  
47 Frank Olken, Lawrence Berkeley National Laboratory  
48 Michael Park, eSum Technologies  
49 Bruce Peat, eProcess Solutions  
50 Mike Rowley, Excelon Corporation  
51 Waqar Sadiq, Vitria  
52 Krishna Sankar, Cisco Systems Inc.  
53 Kim Tae Soo, Government of Korea  
54 Nikola Stojanovic, Encoda Systems, Inc.  
55 David Webber, XML Global  
56 Yutaka Yoshida, Sun Microsystems  
57 Prasad Yendluri, webmethods  
58 Peter Z. Zhoo, Knowledge For the new Millennium  
59

59 **Table of Contents**

60 **1 Status of this Document**..... 1

61 **2 ebXML participants** ..... 2

62 **Table of Contents**..... 3

63 **Table of Tables** ..... 7

64 **3 Introduction** ..... 8

65 3.1 Summary of Contents of Document .....8

66 3.2 General Conventions .....8

67 3.3 Audience .....8

68 3.4 Related Documents.....8

69 **4 Design Objectives** ..... 9

70 4.1 Goals.....9

71 4.2 Caveats and Assumptions.....9

72 **5 System Overview** ..... 9

73 5.1 What The ebXML Registry Does .....9

74 5.2 How The ebXML Registry Works .....9

75 5.2.1 Schema Documents Are Submitted.....10

76 5.2.2 Business Process Documents Are Submitted .....10

77 5.2.3 Seller’s Collaboration Protocol Profile Is Submitted.....10

78 5.2.4 Buyer Discovers The Seller .....10

79 5.2.5 CPA Is Established.....10

80 5.3 Where the Registry Services May Be Implemented .....11

81 5.4 Implementation Conformance .....11

82 5.4.1 Conformance as an ebXML Registry .....11

83 5.4.2 Conformance as an ebXML Registry Client.....11

84 **6 Registry Architecture**..... 12

85 6.1 ebXML Registry Profiles and Agreements .....13

86 6.2 Client To Registry Communication Bootstrapping.....13

87 6.3 Interfaces .....14

88 6.4 Interfaces Exposed By The Registry .....14

89 6.4.1 Interface RegistryService .....15

90 6.4.2 Interface ObjectManager .....15

91 6.4.3 Interface ObjectQueryManager .....16

92 6.5 Interfaces Exposed By Registry Clients .....17

93 6.5.1 Interface RegistryClient .....17

94 6.5.2 Interface ObjectManagerClient.....17

95 6.5.3 Interface ObjectQueryManagerClient .....19

96 **7 Object Management Service** ..... 20

97 7.1 Life Cycle of a Repository Item .....20

98	7.2	RegistryObject Attributes.....	21
99	7.3	The Submit Objects Protocol.....	21
100	7.3.1	Universally Unique ID Generation .....	22
101	7.3.2	ID Attribute And Object References.....	22
102	7.3.3	Sample SubmitObjectsRequest.....	22
103	7.4	The Add Slots Protocol .....	25
104	7.5	The Remove Slots Protocol.....	25
105	7.6	The Approve Objects Protocol .....	26
106	7.7	The Deprecate Objects Protocol .....	26
107	7.8	The Remove Objects Protocol .....	27
108	7.8.1	Deletion Scope DeleteRepositoryItemOnly .....	27
109	7.8.2	Deletion Scope DeleteAll.....	27
110	<b>8</b>	<b>Object Query Management Service .....</b>	<b>28</b>
111	8.1	Browse and Drill Down Query Support.....	29
112	8.1.1	Get Root Classification Nodes Request .....	29
113	8.1.2	Get Classification Tree Request.....	30
114	8.1.3	Get Classified Objects Request.....	30
115	8.1.3.1	Get Classified Objects Request Example .....	31
116	8.2	Filter Query Support.....	32
117	8.2.1	FilterQuery.....	34
118	8.2.2	RegistryEntryQuery .....	36
119	8.2.3	AuditableEventQuery.....	42
120	8.2.4	ClassificationNodeQuery .....	45
121	8.2.5	RegistryPackageQuery.....	48
122	8.2.6	OrganizationQuery .....	50
123	8.2.7	ReturnRegistryEntry .....	53
124	8.2.8	ReturnRepositoryItem.....	57
125	8.2.9	Registry Filters .....	61
126	8.2.10	XML Clause Constraint Representation .....	64
127	8.3	SQL Query Support.....	68
128	8.3.1	SQL Query Syntax Binding To [ebRIM].....	68
129	8.3.1.1	Interface and Class Binding .....	68
130	8.3.1.2	Accessor Method To Attribute Binding.....	69
131	8.3.1.3	Primitive Attributes Binding .....	69
132	8.3.1.4	Reference Attribute Binding .....	69
133	8.3.1.5	Complex Attribute Binding.....	69
134	8.3.1.6	Collection Attribute Binding.....	70
135	8.3.2	Semantic Constraints On Query Syntax.....	70
136	8.3.3	SQL Query Results.....	70
137	8.3.4	Simple Metadata Based Queries .....	70
138	8.3.5	RegistryEntry Queries .....	71
139	8.3.6	Classification Queries.....	71
140	8.3.6.1	Identifying ClassificationNodes .....	71
141	8.3.6.2	Getting Root Classification Nodes .....	71
142	8.3.6.3	Getting Children of Specified ClassificationNode.....	71

143	8.3.6.4 Getting Objects Classified By a ClassificationNode .....	72
144	8.3.6.5 Getting ClassificationNodes That Classify an Object...	72
145	8.3.7 Association Queries.....	72
146	8.3.7.1 Getting All Association With Specified Object As Its Source	
147	72	
148	8.3.7.2 Getting All Association With Specified Object As Its Target	
149	72	
150	8.3.7.3 Getting Associated Objects Based On Association Attributes	
151	72	
152	8.3.7.4 Complex Association Queries .....	73
153	8.3.8 Package Queries.....	73
154	8.3.8.1 Complex Package Queries .....	73
155	8.3.9 ExternalLink Queries .....	73
156	8.3.9.1 Complex ExternalLink Queries.....	73
157	8.3.10 Audit Trail Queries.....	74
158	8.4 Ad Hoc Query Request/Response .....	74
159	8.5 Content Retrieval .....	75
160	8.5.1 Identification Of Content Payloads .....	75
161	8.5.2 GetContentResponse Message Structure .....	76
162	8.6 Query And Retrieval: Typical Sequence .....	76
163	<b>9 Registry Security.....</b>	<b>77</b>
164	9.1 Integrity of Registry Content.....	77
165	9.1.1 Message Payload Signature.....	78
166	9.2 Authentication .....	78
167	9.2.1 Message Header Signature.....	78
168	9.3 Confidentiality.....	78
169	9.3.1 On-the-wire Message Confidentiality.....	78
170	9.3.2 Confidentiality of Registry Content .....	79
171	9.4 Authorization .....	79
172	9.4.1 Pre-defined Roles For Registry Users .....	79
173	9.4.2 Default Access Control Policies.....	79
174	<b>Appendix A Schemas and DTD Definitions.....</b>	<b>80</b>
175	A.1 ebXML Error Message .....	80
176	A.2 ebXML Registry DTD .....	81
177	<b>Appendix B Interpretation of UML Diagrams.....</b>	<b>92</b>
178	B.1 UML Class Diagram .....	92
179	B.2 UML Sequence Diagram .....	92
180	<b>Appendix C SQL Query.....</b>	<b>93</b>
181	C.1 SQL Query Syntax Specification .....	93
182	C.2 Non-Normative BNF for Query Syntax Grammar .....	94
183	C.3 Relational Schema For SQL Queries .....	95
184	<b>Appendix D Non-normative Content Based Ad Hoc Queries .....</b>	<b>101</b>
185	D.1.1 Automatic Classification of XML Content.....	102

186	D.1.2 Index Definition.....	102
187	D.1.3 Example Of Index Definition.....	102
188	D.1.4 Proposed XML Definition.....	103
189	D.1.5 Example of Automatic Classification.....	103
190	<b>Appendix E Security Implementation Guideline.....</b>	<b>103</b>
191	E.1 Authentication.....	103
192	E.2 Authorization.....	104
193	E.3 Registry Bootstrap.....	104
194	E.4 Content Submission – Client Responsibility.....	104
195	E.5 Content Submission – Registry Responsibility.....	104
196	E.6 Content Delete/Deprecate – Client Responsibility.....	104
197	E.7 Content Delete/Deprecate – Registry Responsibility.....	105
198	<b>Appendix F Native Language Support (NLS).....</b>	<b>105</b>
199	F.1 Definitions.....	105
200	F.1.1 Coded Character Set (CCS):.....	105
201	F.1.2 Character Encoding Scheme (CES):.....	105
202	F.1.3 Character Set (charset):.....	105
203	F.2 NLS And Request / Response Messages.....	105
204	F.3 NLS And Storing of RegistryEntry.....	106
205	F.3.1 Character Set of <i>RegistryEntry</i> .....	106
206	F.3.2 Language Information of <i>RegistryEntry</i> .....	106
207	F.4 NLS And Storing of Repository Items.....	106
208	F.4.1 Character Set of Repository Items.....	106
209	F.4.2 Language information of repository item.....	107
210	<b>Appendix G Terminology Mapping.....</b>	<b>107</b>
211	<b>References.....</b>	<b>109</b>
212	<b>Disclaimer.....</b>	<b>110</b>
213	<b>Contact Information.....</b>	<b>111</b>
214	<b>Copyright Statement.....</b>	<b>112</b>
215	<b>Table of Figures</b>	
216	Figure 1: Registry Architecture Supports Flexible Topologies.....	12
217	Figure 2: ebXML Registry Interfaces.....	14
218	Figure 3: Life Cycle of a Registry Entry.....	20
219	Figure 4: Submit Objects Sequence Diagram.....	21
220	Figure 5: Add Slots Sequence Diagram.....	25
221	Figure 6: Remove Slots Sequence Diagram.....	25
222	Figure 7: Approve Objects Sequence Diagram.....	26

---

223	Figure 8: Deprecate Objects Sequence Diagram .....	27
224	Figure 9: Remove Objects Sequence Diagram .....	28
225	Figure 10: Get Root Classification Nodes Sequence Diagram.....	29
226	Figure 11: Get Root Classification Nodes Asynchronous Sequence Diagram ...	29
227	Figure 12: Get Classification Tree Sequence Diagram .....	30
228	Figure 13: Get Classification Tree Asynchronous Sequence Diagram.....	30
229	Figure 14: A Sample Geography Classification .....	31
230	Figure 15: Submit Ad Hoc Query Sequence Diagram .....	75
231	Figure 16: Submit Ad Hoc Query Asynchronous Sequence Diagram .....	75
232	Figure 17: Typical Query and Retrieval Sequence .....	77

233 **Table of Tables**

234	Table 1: Terminology Mapping Table .....	108
235		
236		

## 236 **3 Introduction**

### 237 **3.1 Summary of Contents of Document**

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

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

### 243 **3.2 General Conventions**

244 The following conventions are used throughout this document:

- 245 o UML diagrams are used as a way to concisely describe concepts. They are not  
246 intended to convey any specific *Implementation* or methodology requirements.
- 247 o The term “*repository item*” is used to refer to an object that has been submitted to a  
248 Registry for storage and safekeeping (e.g. an XML document or a DTD). Every  
249 repository item is described by a RegistryEntry instance.
- 250 o The term “*RegistryEntry*” is used to refer to an object that provides metadata about a  
251 *repository item*.
- 252 o *Capitalized Italic* words are defined in the ebXML Glossary.

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

### 256 **3.3 Audience**

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

- 259 o Implementers of ebXML Registry Services
- 260 o Implementers of ebXML Registry Clients

### 261 **3.4 Related Documents**

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

- 264 a) *ebXML Registry Information Model [ebRIM]*
- 265 b) *ebXML Message Service Specification [ebMS]*
- 266 c) *ebXML Business Process Specification Schema [ebBPM]*
- 267 d) *ebXML Collaboration-Protocol Profile and Agreement Specification [ebCPP]*



## 268 **4 Design Objectives**

### 269 **4.1 Goals**

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

- 271 o Communicate functionality of Registry services to software developers
- 272 o Specify the interface for Registry clients and the Registry
- 273 o Provide a basis for future support of more complete ebXML Registry requirements
- 274 o Be compatible with other ebXML specifications

### 275 **4.2 Caveats and Assumptions**

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

279 It is assumed that:

- 280 1. All interactions between the clients of the ebXML Registry and the ebXML  
281 Registry may optionally be implemented by means other than that specified in  
282 the ebXML Message Service Specification. However, these optional  
283 communication means are outside the scope of this specification.
- 284 2. All access to the Registry content is exposed via the interfaces defined for the  
285 Registry Services.
- 286 3. The Registry makes use of a Repository for storing and retrieving persistent  
287 information required by the Registry Services. This is an implementation detail  
288 that will not be discussed further in this specification.

## 289 **5 System Overview**

### 290 **5.1 What The ebXML Registry Does**

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

### 296 **5.2 How The ebXML Registry Works**

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

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

### 307 **5.2.1 Schema Documents Are Submitted**

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

### 312 **5.2.2 Business Process Documents Are Submitted**

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

### 317 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

### 322 **5.2.4 Buyer Discovers The Seller**

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

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

### 329 **5.2.5 CPA Is Established**

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

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

### 336 **5.3 Where the Registry Services May Be Implemented**

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

### 339 **5.4 Implementation Conformance**

340 An implementation is a *conforming* ebXML Registry if the implementation meets the  
341 conditions in Section 5.4.1. An implementation is a conforming ebXML Registry Client if  
342 the implementation meets the conditions in Section 5.4.2. An implementation is a  
343 conforming ebXML Registry and a conforming ebXML Registry Client if the  
344 implementation conforms to the conditions of Section 5.4.1 and Section 5.4.2. An  
345 implementation shall be a conforming ebXML Registry, a conforming ebXML Registry  
346 Client, or a conforming ebXML Registry and Registry Client.

#### 347 **5.4.1 Conformance as an ebXML Registry**

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

- 350 1. Conforms to *the ebXML Registry Information Model [ebRIM]*.
- 351 2. Supports the syntax and semantics of the Registry Interfaces and Security  
352 Model.
- 353 3. Supports the defined ebXML Error Message DTD (Appendix A.1)
- 354 4. Supports the defined ebXML Registry DTD (Appendix A.2)
- 355 5. Optionally supports the syntax and semantics of Section 8.3, SQL Query  
356 Support.

#### 357 **5.4.2 Conformance as an ebXML Registry Client**

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

- 360 1. Supports the ebXML CPA and bootstrapping process.
- 361 2. Supports the syntax and the semantics of the Registry Client Interfaces.
- 362 3. Supports the defined ebXML Error Message DTD.
- 363 4. Supports the defined ebXML Registry DTD.

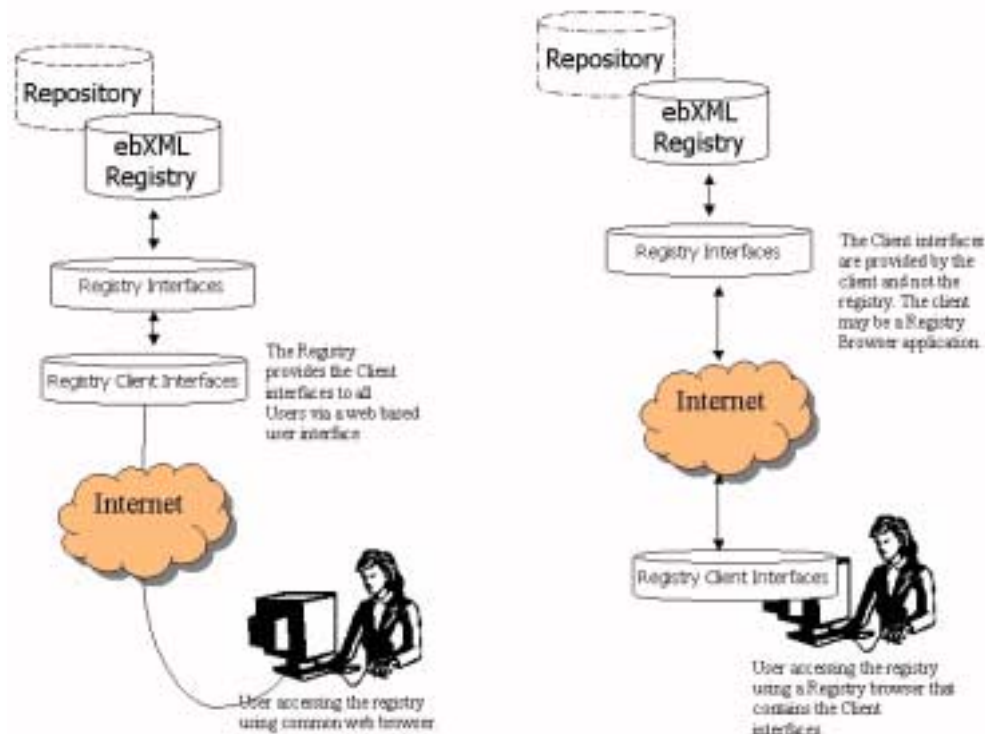
## 364 6 Registry Architecture

365 The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry  
 366 Clients. The Registry Client interfaces may be local to the registry or local to the user.  
 367 Figure 1 depicts the two possible topologies supported by the registry architecture with  
 368 respect to the Registry and Registry Clients.

369 The picture on the left side shows the scenario where the Registry provides a web  
 370 based “thin client” application for accessing the Registry that is available to the user  
 371 using a common web browser. In this scenario the Registry Client interfaces reside  
 372 across the internet and are local to the Registry from the user’s view.

373 The picture on the right side shows the scenario where the user is using a “fat client”  
 374 Registry Browser application to access the registry. In this scenario the Registry Client  
 375 interfaces reside within the Registry Browser tool and are local to the Registry from the  
 376 user’s view. The Registry Client interfaces communicate with the Registry over the  
 377 internet in this scenario.

378 A third topology made possible by the registry architecture is where the Registry Client  
 379 interfaces reside in a server side business component such as a Purchasing business  
 380 component. In this topology there may be no direct user interface or user intervention  
 381 involved. Instead the Purchasing business component may access the Registry in an  
 382 automated manner to select possible sellers or service providers based current  
 383 business needs.



384

385

**Figure 1: Registry Architecture Supports Flexible Topologies**

386 Clients communicate with the Registry using the ebXML Messaging Service in the same  
387 manner as any two ebXML applications communicating with each other.

388 Future versions of this specification may provide additional services to explicitly extend  
389 the Registry architecture to support distributed registries. However this current version  
390 of the specification does not preclude ebXML Registries from cooperating with each  
391 other to share information, nor does it preclude owners of ebXML Registries from  
392 registering their ebXML registries with other registry systems, catalogs, or directories.

## 393 **6.1 ebXML Registry Profiles and Agreements**

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

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

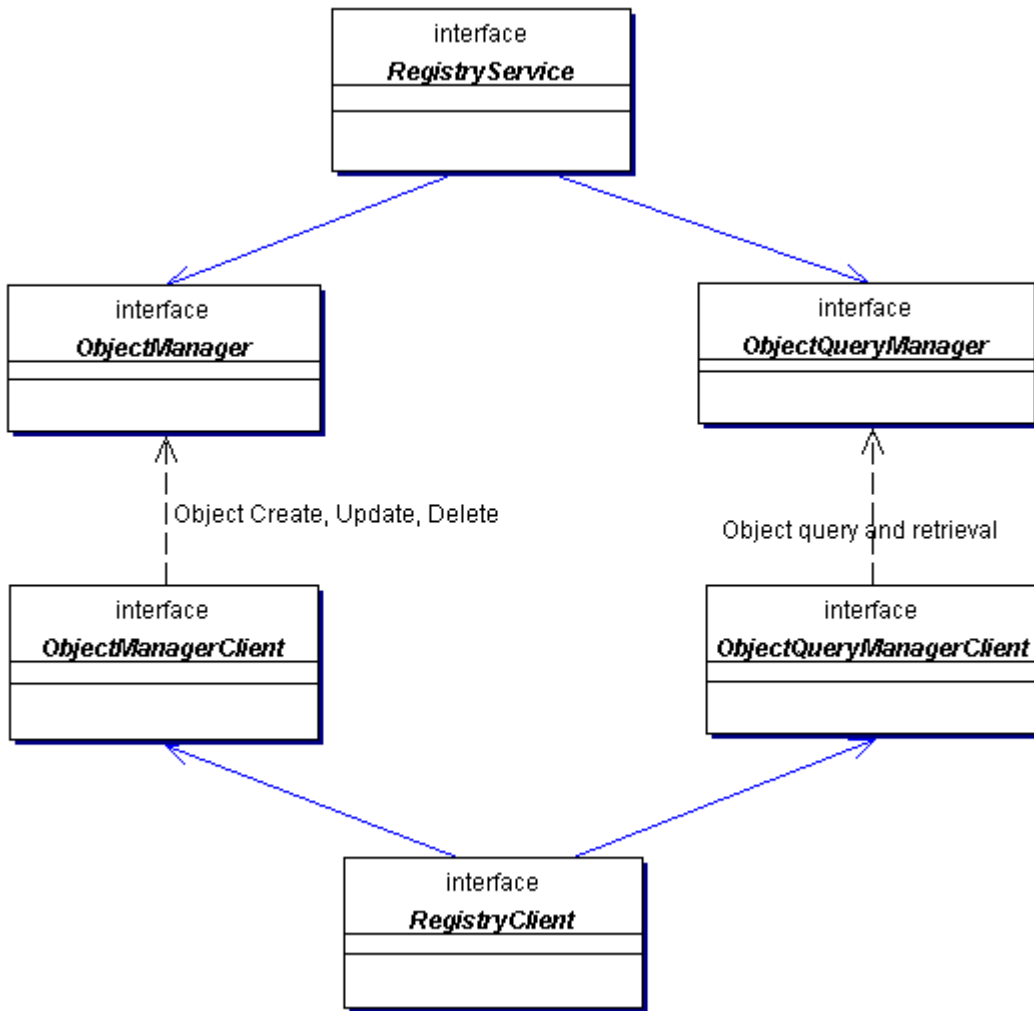
403 The CPA between clients and the Registry should describe the interfaces that the  
404 Registry and the client expose to each other for Registry-specific interactions. These  
405 interfaces are described in Figure 2 and subsequent sections. The definition of the  
406 Registry CPP template and a Registry Client CPP template are beyond the scope of this  
407 document.

## 408 **6.2 Client To Registry Communication Bootstrapping**

409 Since there is no previously established CPA between the Registry and the  
410 RegistryClient, the client must know at least one Transport-specific communication  
411 address for the Registry. This communication address is typically a URL to the Registry,  
412 although it could be some other type of address such as an email address.

413 For example, if the communication used by the Registry is HTTP, then the  
414 communication address is a URL. In this example, the client uses the Registry's public  
415 URL to create an implicit CPA with the Registry. When the client sends a request to the  
416 Registry, it provides a URL to itself. The Registry uses the client's URL to form its  
417 version of an implicit CPA with the client. At this point a session is established within the  
418 Registry.

419 For the duration of the client's session with the Registry, messages may be exchanged  
420 bidirectionally as required by the interaction protocols defined in this specification.



421  
422

Figure 2: ebXML Registry Interfaces

423 **6.3 Interfaces**

424 This specification defines the interfaces exposed by both the Registry (Section 6.4) and  
 425 the Registry Client (Section 6.5). Figure 2 shows the relationship between the  
 426 interfaces and the mapping of specific Registry interfaces with specific Registry Client  
 427 interfaces.

428 **6.4 Interfaces Exposed By The Registry**

429 The ebXML Registry implements the following interfaces as its services (Registry  
 430 Services).

431  
432

433 **6.4.1 Interface RegistryService**

434  
 435 This is the principal interface implemented by the Registry. It provides the methods that  
 436 are used by the client to discover service-specific interfaces implemented by the  
 437 Registry.

438

**Method Summary**

<a href="#">ObjectManager</a>	<a href="#">getObjectManager()</a> Returns the ObjectManager interface implemented by the Registry service.
<a href="#">ObjectQueryManager</a>	<a href="#">getObjectQueryManager()</a> Returns the ObjectQueryManager interface implemented by the Registry service.

439 **6.4.2 Interface ObjectManager**

440  
 441 This is the interface exposed by the Registry Service that implements the Object life  
 442 cycle management functionality of the Registry. Its methods are invoked by the Registry  
 443 Client. For example, the client may use this interface to submit objects, to classify and  
 444 associate objects and to deprecate and remove objects. For this specification the  
 445 semantic meaning of submit, classify, associate, deprecate and remove is found in  
 446 [ebRIM].

447

**Method Summary**

Void	<a href="#">approveObjects(ApproveObjectsRequest req)</a> Approves one or more previously submitted objects.
Void	<a href="#">deprecateObjects(DeprecateObjectsRequest req)</a> Deprecates one or more previously submitted objects.
Void	<a href="#">removeObjects(RemoveObjectsRequest req)</a> Removes one or more previously submitted objects from the Registry.
void	<a href="#">submitObjects(SubmitObjectsRequest req)</a> Submits one or more objects and possibly related metadata such as Associations and Classifications.
void	<a href="#">addSlots(AddSlotsRequest req)</a> Add slots to one or more registry entries.

void	<a href="#">removeSlots</a> ( <a href="#">RemoveSlotsRequest</a> req) Remove specified slots from one or more registry entries.
------	--

448 **6.4.3 Interface ObjectQueryManager**

449 \_\_\_\_\_

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

454 \_\_\_\_\_

<b>Method Summary</b>	
<a href="#">GetClassificationTreeResponse</a>	<a href="#">getClassificationTree</a> ( <a href="#">GetClassificationTreeRequest</a> req) Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.
void	<a href="#">getClassificationTreeAsync</a> ( <a href="#">GetClassificationTreeRequest</a> req) Asynchronous version of getClassificationTree.
<a href="#">GetClassifiedObjectsResponse</a>	<a href="#">getClassifiedObjects</a> ( <a href="#">GetClassifiedObjectsRequest</a> req) Returns a collection of references to RegistryEntries classified under specified ClassificationItem.
void	<a href="#">getClassifiedObjectsAsync</a> ( <a href="#">GetClassifiedObjectsRequest</a> req) Asynchronous version of getClassifiedObjects.
<a href="#">GetContentResponse</a>	<a href="#">getContent</a> ( ) Returns the content of the specified Repository Item. The response includes all the content specified in the request as additional payloads within the response message.
void	<a href="#">getContentAsync</a> ( ) Async version of getContent.
<a href="#">GetRootClassificationNodesResponse</a>	<a href="#">getRootClassificationNodes</a> ( <a href="#">GetRootClassificationNodesRequest</a> req) Returns all root ClassificationNodes that match



		the namePattern attribute in GetRootClassificationNodesRequest request.
	void	<a href="#">getRootClassificationNodesAsync</a> ( <a href="#">GetRootClassificationNodesRequest</a> req) Async version of getRootClassificationNodes.
	<a href="#">AdhocQueryResponse</a>	<a href="#">submitAdhocQuery</a> ( <a href="#">AdhocQueryRequest</a> req) Submit an ad hoc query request.
	void	<a href="#">submitAdhocQueryAsync</a> ( <a href="#">AdhocQueryRequest</a> req) Async version of submitAdhocQuery.

455 **6.5 Interfaces Exposed By Registry Clients**

456 An ebXML Registry client implements the following interfaces.

457 **6.5.1 Interface RegistryClient**

458 

---

459 This is the principal interface implemented by a Registry client. The client provides this  
 460 interface when creating a connection to the Registry. It provides the methods that are  
 461 used by the Registry to discover service-specific interfaces implemented by the client.

462 

---

<b>Method Summary</b>	
<a href="#">ObjectManagerClient</a>	<a href="#">getObjectManagerClient</a> () Returns the ObjectManagerClient interface implemented by the client.
<a href="#">ObjectQueryManagerClient</a>	<a href="#">getObjectQueryManagerClient</a> () Returns the ObjectQueryManagerClient interface implemented by the client.

463

464 **6.5.2 Interface ObjectManagerClient**

465 

---

466 This is the client callback interface for the ObjectManager service of the Registry. The  
 467 ObjectManager invokes its methods to notify the client about the results of a previously  
 468 submitted request from the client to the ObjectManager service.

469 

---

<b>Method Summary</b>	
void	<a href="#">addSlotsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted AddSlotsRequest was accepted by the Registry.
void	<a href="#">addSlotsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted AddSlotsRequest was not accepted by the Registry due to an error.
void	<a href="#">approveObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.
void	<a href="#">approveObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted ApproveObjectsRequest was not accepted by the Registry due to an error.
void	<a href="#">deprecateObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted DeprecateObjectsRequest was accepted by the Registry.
void	<a href="#">deprecateObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted DeprecateObjectsRequest was not accepted by the Registry due to an error.
void	<a href="#">removeObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted RemoveObjectsRequest was accepted by the Registry.
void	<a href="#">removeSlotsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted RemoveSlotsRequest was accepted by the Registry.
void	<a href="#">removeObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted RemoveObjectsRequest was not accepted by the Registry due to an error.
void	<a href="#">removeSlotsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted RemoveSlotsRequest was not accepted by the Registry due to an error.
void	<a href="#">submitObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted SubmitObjectsRequest was accepted by the Registry.
void	<a href="#">submitObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted SubmitObjectsRequest was not accepted by the Registry due to an error.

470

471 **6.5.3 Interface ObjectQueryManagerClient**

472

473 This is the client callback interface for the ObjectQueryManager service of the Registry.  
 474 The ObjectQueryManager invokes its methods to notify the client about the results of a  
 475 previously submitted query request from the client to the ObjectQueryManager service.

476

<b>Method Summary</b>	
void	<a href="#"><u>getClassificationTreeAsyncResponse</u></a> ( <a href="#"><u>GetClassificationTreeResponse</u></a> resp) Async response for getClassificationTreeAsync request.
void	<a href="#"><u>getClassifiedObjectsAsyncResponse</u></a> ( <a href="#"><u>GetClassifiedObjectsResponse</u></a> resp) Async response for getClassifiedObjectsAsync request.
void	<a href="#"><u>getContentAsyncResponse</u></a> ( <a href="#"><u>GetContentResponse</u></a> resp) Async response for getContent request.
void	<a href="#"><u>getRootClassificationNodesAsyncResponse</u></a> ( <a href="#"><u>GetRootClassificationNodesResponse</u></a> resp) Async response for getRootClassificationNodesAsync request.
void	<a href="#"><u>submitAdhocQueryAsyncResponse</u></a> ( <a href="#"><u>AdhocQueryResponse</u></a> resp) Async response for submitAdhocQueryAsync request.

477

478

478 **7 Object Management Service**

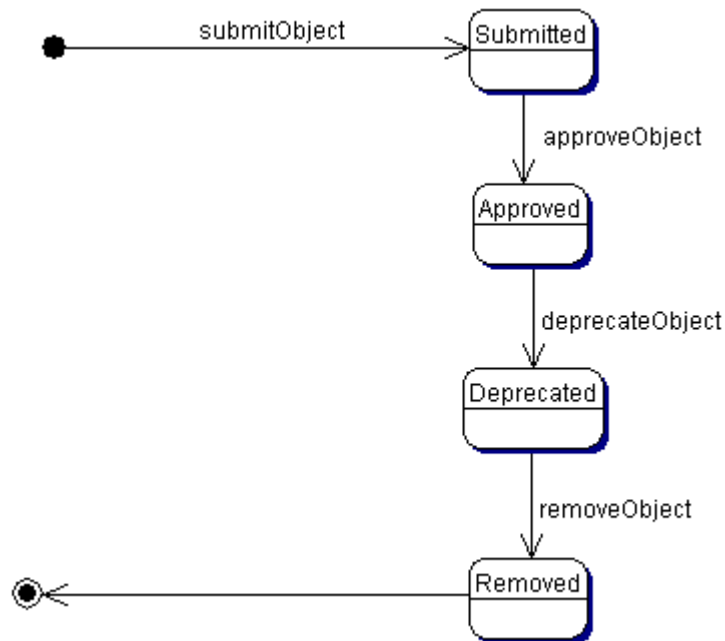
479 This section defines the ObjectManagement service of the Registry. The Object  
 480 Management Service is a sub-service of the Registry service. It provides the  
 481 functionality required by RegistryClients to manage the life cycle of repository items  
 482 (e.g. XML documents required for ebXML business processes). The Object  
 483 Management Service can be used with all types of repository items as well as the  
 484 metadata objects specified in [ebRIM] such as Classification and Association.

485 The minimum *security policy* for an ebXML registry is to accept content from any client if  
 486 the content is digitally signed by a certificate issued by a Certificate Authority  
 487 recognized by the ebXML registry. Submitting Organizations do not have to register  
 488 prior to submitting content.

489 **7.1 Life Cycle of a Repository Item**

490 The main purpose of the ObjectManagement service is to manage the life cycle of  
 491 repository items.

492 Figure 3 shows the typical life cycle of a repository item. Note that the current version of  
 493 this specification does not support Object versioning. Object versioning will be added in  
 494 a future version of this specification.



495

496

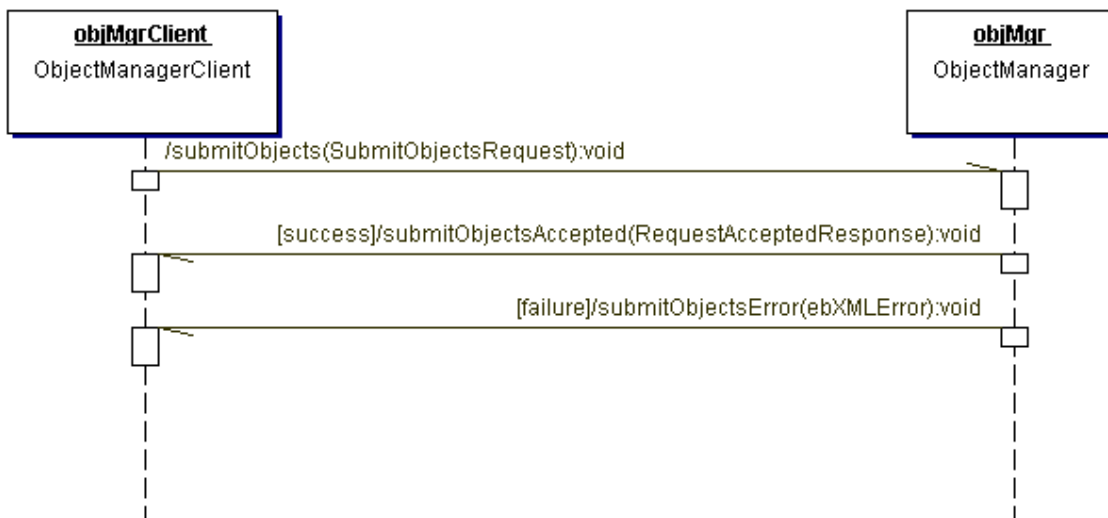
**Figure 3: Life Cycle of a Repository Item**

497 **7.2 RegistryObject Attributes**

498 A repository item is associated with a set of standard metadata defined as attributes of  
 499 the RegistryObject class and its sub-classes as described in [ebRIM]. These attributes  
 500 reside outside of the actual repository item and catalog descriptive information about the  
 501 repository item. XML elements called ExtrinsicObject and IntrinsicObject (See Appendix  
 502 A.2 for details) encapsulate all object metadata attributes defined in [ebRIM] as XML  
 503 attributes.

504 **7.3 The Submit Objects Protocol**

505 This section describes the protocol of the Registry Service that allows a RegistryClient  
 506 to submit one or more repository items to the repository using the *ObjectManager* on  
 507 behalf of a Submitting Organization. It is expressed in UML notation as described in  
 508 Appendix B.



509  
 510 **Figure 4: Submit Objects Sequence Diagram**

511 For details on the schema for the *Business documents* shown in this process refer to  
 512 Appendix A.2.

513 The SubmitObjectRequest message includes a RegistrEntryList element.

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

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

### 520 **7.3.1 Universally Unique ID Generation**

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

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

525 This id is usually generated by the registry. The `id` attribute for submitted objects may  
526 optionally be supplied by the client. If the client supplies the `id` and it conforms to the  
527 format of a URN that specifies a DCE 128 bit UUID

528 then the registry assumes that the client wishes to specify the `id` for the object. In this  
529 case, the registry must honor a client-supplied `id` and use it as the `id` attribute of the  
530 object in the registry. If the `id` is found by the registry to not be globally unique, the  
531 registry must send an `ebXML_Error` in response with an `InvalidIdError` message.

532 If the client does not supply an `id` for a submitted object then the registry must generate  
533 a universally unique `id`. Whether the `id` is generated by the client or whether it is  
534 generated by the registry, it must be generated using the DCE 128 bit UUID generation  
535 algorithm as specified in [UUID].

### 536 **7.3.2 ID Attribute And Object References**

537 The `id` attribute of an object may be used by other objects to reference the first object.  
538 Such references are common both within the `SubmitObjectsRequest` as well as within  
539 the registry. Within a `SubmitObjectsRequest`, the `id` attribute may be used to refer to an  
540 object within the `SubmitObjectsRequest` as well as to refer to an object within the  
541 registry. An object in the `SubmitObjectsRequest` that needs to be referred to within the  
542 request document may be assigned an `id` by the submitter so that it can be referenced  
543 within the request. The submitter may give the object a proper `uuid` URN, in which case  
544 the `id` is permanently assigned to the object within the registry. Alternatively, the  
545 submitter may assign an arbitrary `id` (not a proper `uuid` URN) as long as the `id` is unique  
546 within the request document. In this case the `id` serves as a linkage mechanism within  
547 the request document but must be ignored by the registry and replaced with a registry  
548 generated `id` upon submission.

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

### 553 **7.3.3 Sample SubmitObjectsRequest**

554 The following example shows several different use cases in a single  
555 `SubmitObjectsRequest`. It does not show the complete ebXML Message with the  
556 message header and additional payloads in the message for the repository items.

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

```

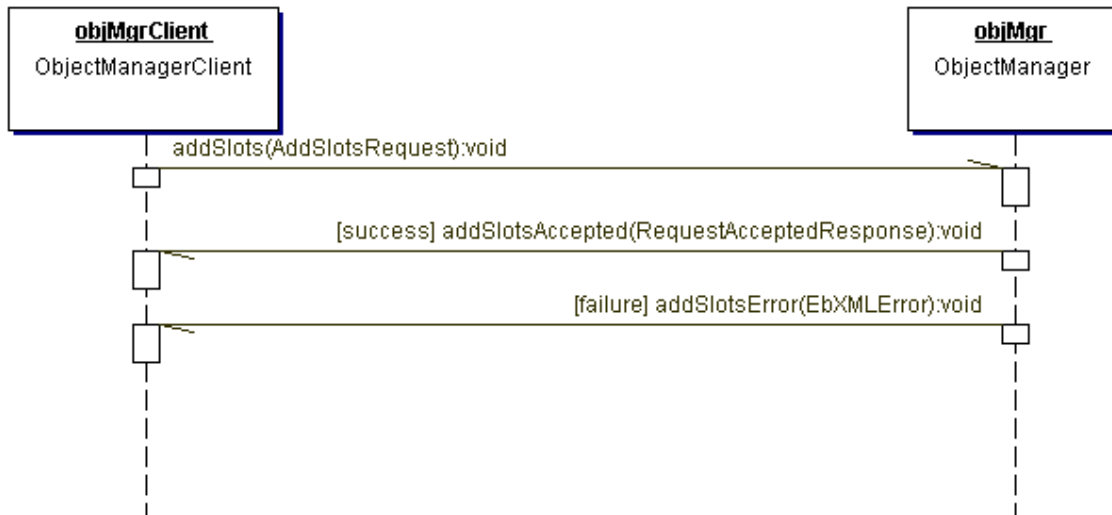
560
561 <?xml version = "1.0" encoding = "UTF-8"?>
562 <!DOCTYPE SubmitObjectsRequest SYSTEM "file:///home/najmi/Registry.dtd">
563
564 <SubmitObjectsRequest>
565   <RegistryEntryList>
566
567     <!--
568     The following 3 objects package specified ExtrinsicObject in specified
569     Package, where both the Package and the ExtrinsicObject are
570     being submitted
571     -->
572     <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
573     <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
574       objectType = "CPP" name = "Widget Profile"
575       description = "ACME's profile for selling widgets"/>
576     <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
577       sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>
578
579     <!--
580     The following 3 objects package specified ExtrinsicObject in specified Package,
581     Where the Package is being submitted and the ExtrinsicObject is
582     already in registry
583     -->
584     <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
585     <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
586     <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
587       associationType = "Packages" sourceObject = "acmePackage2"
588       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
589
590     <!--
591     The following 3 objects package specified ExtrinsicObject in specified Package,
592     where the Package and the ExtrinsicObject are already in registry
593     -->
594     <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
595     <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
596     <!-- id is unspecified implying that registry must create a uuid for this object -->
597     <Association associationType = "Packages"
598       sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
599       targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
600
601     <!--
602     The following 3 objects externally link specified ExtrinsicObject using
603     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
604     are being submitted
605     -->
606     <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
607     <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
608       name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
609     <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
610       sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
611
612     <!--
613     The following 2 objects externally link specified ExtrinsicObject using specified
614     ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
615     is already in registry. Note that the targetObject points to an ObjectRef in a
616     previous line
617     -->
618     <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
619     <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
620       associationType = "ExternallyLinks" sourceObject = "acmeLink2"
621       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
622
623     <!--
  
```

```
624 The following 2 objects externally identify specified ExtrinsicObject using specified
625 ExternalIdentifier, where the ExternalIdentifier is being submitted and the
626 ExtrinsicObject is already in registry. Note that the targetObject points to an
627 ObjectRef in a previous line
628 -->
629 <ExternalIdentifier id = "acmeDUNSID" name = "DUNS" description = "DUNS ID for ACME"
630 value = "13456789012"/>
631 <Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc"
632 associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSID"
633 targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
634
635 <!--
636 The following show submission of a brand new classification scheme in its entirety
637 -->
638 <ClassificationNode id = "geographyNode" name = "Geography"
639 description = "The Geography scheme example from Registry Services Spec" />
640 <ClassificationNode id = "asiaNode" name = "Asia"
641 description = "The Asia node under the Geography node" parent="geographyNode" />
642 <ClassificationNode id = "japanNode" name = "Japan"
643 description = "The Japan node under the Asia node" parent="asiaNode" />
644 <ClassificationNode id = "koreaNode" name = "Korea"
645 description = "The Korea node under the Asia node" parent="asiaNode" />
646 <ClassificationNode id = "europeNode" name = "Europe"
647 description = "The Europe node under the Geography node" parent="geographyNode" />
648 <ClassificationNode id = "germanyNode" name = "Germany"
649 description = "The Germany node under the Asia node" parent="europeNode" />
650 <ClassificationNode id = "northAmericaNode" name = "North America"
651 description = "The North America node under the Geography node"
652 parent="geographyNode" />
653 <ClassificationNode id = "usNode" name = "US"
654 description = "The US node under the Asia node" parent="northAmericaNode" />
655
656 <!--
657 The following show submission of a Automotive sub-tree of ClassificationNodes that
658 gets added to an existing classification scheme named 'Industry'
659 that is already in the registry
660 -->
661 <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
662 <ClassificationNode id = "automotiveNode" name = "Automotive"
663 description = "The Automotive sub-tree under Industry scheme"
664 parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
665 <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
666 description = "The Parts Supplier node under the Automotive node"
667 parent="automotiveNode" />
668 <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
669 description = "The Engine Supplier node under the Automotive node"
670 parent="automotiveNode" />
671
672 <!--
673 The following show submission of 2 Classifications of an object that is already in
674 the registry using 2 ClassificationNodes. One ClassificationNode
675 is being submitted in this request (Japan) while the other is already in the registry.
676 -->
677 <Classification id = "japanClassification"
678 description = "Classifies object by /Geography/Asia/Japan node"
679 classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
680 classificationNode="japanNode" />
681 <Classification id = "classificationUsingExistingNode"
682 description = "Classifies object using a node in the registry"
683 classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
684 classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
685 <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />
686
687
688 </RegistryEntryList>
689 </SubmitObjectsRequest>
```



690 **7.4 The Add Slots Protocol**

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



694

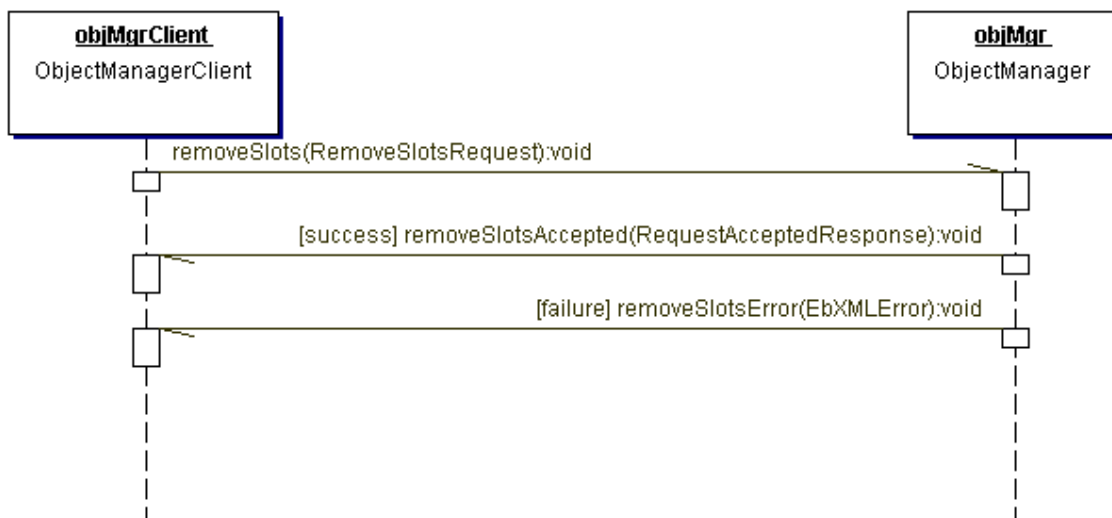
695

**Figure 5: Add Slots Sequence Diagram**

696

697 **7.5 The Remove Slots Protocol**

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



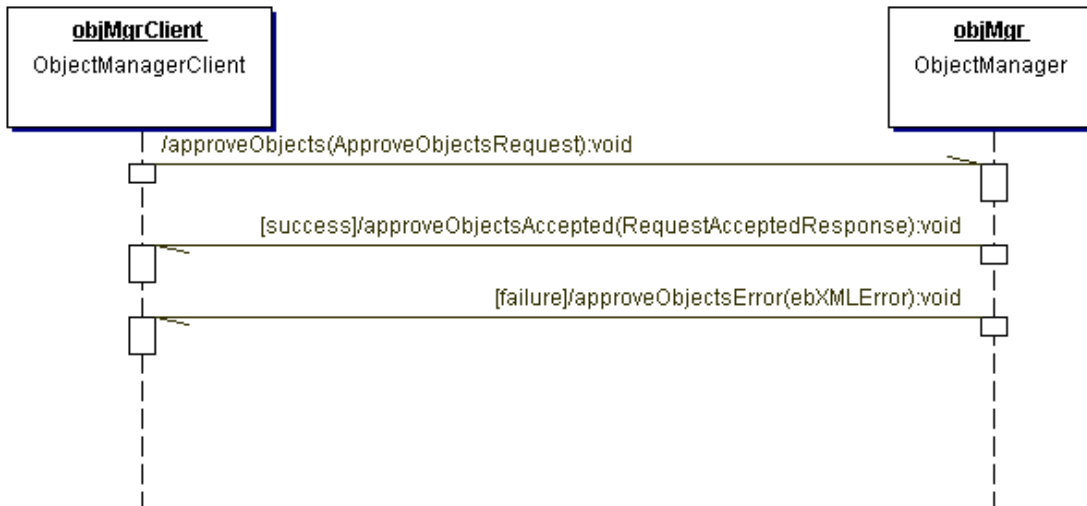
700

701

**Figure 6: Remove Slots Sequence Diagram**

702 **7.6 The Approve Objects Protocol**

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



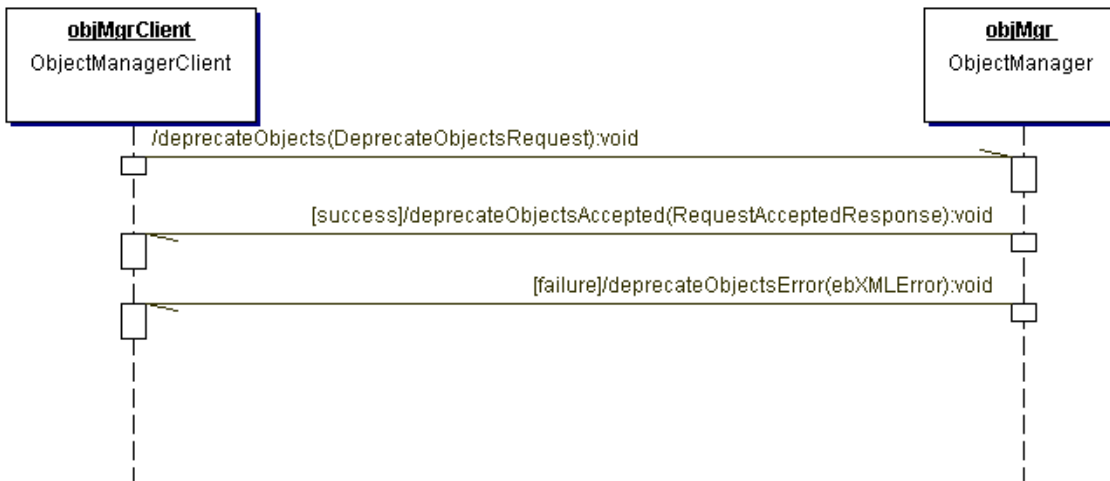
707

708 **Figure 7: Approve Objects Sequence Diagram**

709 For details on the schema for the business documents shown in this process refer to  
 710 Appendix A.2.

711 **7.7 The Deprecate Objects Protocol**

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



717  
718

**Figure 8: Deprecate Objects Sequence Diagram**

719 For details on the schema for the business documents shown in this process refer to  
720 Appendix A.2.

721 **7.8 The Remove Objects Protocol**

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

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

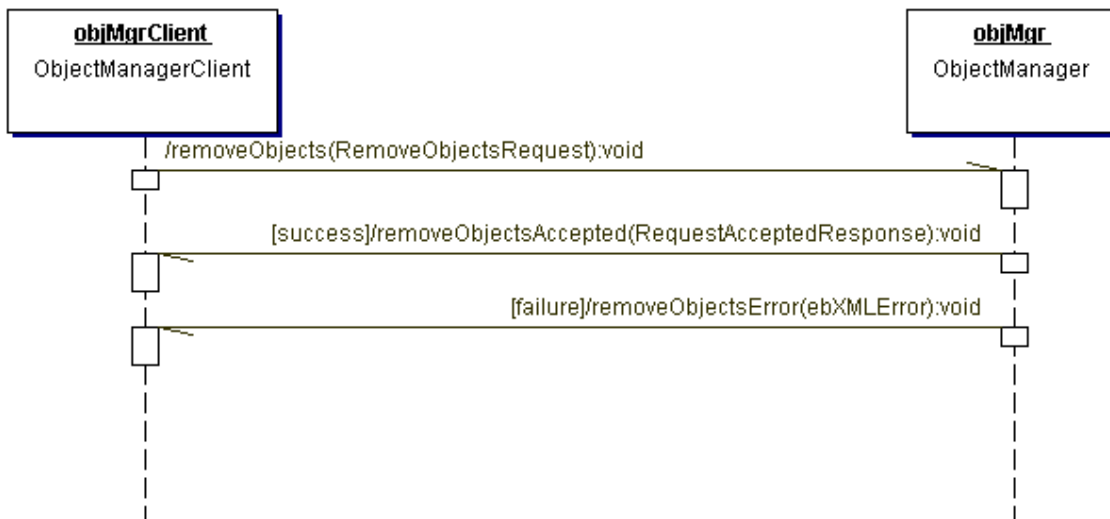
728 **7.8.1 Deletion Scope DeleteRepositoryItemOnly**

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

732 **7.8.2 Deletion Scope DeleteAll**

733 This deletionScope specifies that the request should delete both the RegistryEntry and  
734 the repository item for the specified registry entries. Only if all references (e.g.  
735 Associations, Classifications, ExternalLinks) to a RegistryEntry have been removed, can  
736 that RegistryEntry then be removed using a RemoveObjectsRequest with  
737 deletionScope DeleteAll. Attempts to remove a RegistryEntry while it still has references  
738 results in an InvalidRequestError that is returned within an ebXMLError message sent to  
739 the ObjectManagerClient by the ObjectManager.

740 The remove object protocol is expressed in UML notation as described in Appendix B.



741  
742

**Figure 9: Remove Objects Sequence Diagram**

743 For details on the schema for the business documents shown in this process refer to  
744 Appendix A.2.

745 **8 Object Query Management Service**

746 This section describes the capabilities of the Registry Service that allow a client  
747 (ObjectQueryManagerClient) to search for or query RegistryEntries in the ebXML  
748 Registry using the ObjectQueryManager interface of the Registry.

749 The Registry supports multiple query capabilities. These include the following:

- 750 1. Browse and Drill Down Query
- 751 2. Filtered Query
- 752 3. SQL Query

753 The browse and drill down query in Section 8.1 and the filtered query mechanism in  
754 Section 8.2 SHALL be supported by every Registry implementation. The SQL query  
755 mechanism is an optional feature and MAY be provided by a registry implementation.  
756 However, if a vendor provides an SQL query capability to an ebXML Registry it SHALL  
757 conform to this document. As such this capability is a normative yet optional capability.

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

760 Any errors in the query request messages are indicated in the corresponding query  
761 response message. Note that for each query request/response there is both a  
762 synchronous and asynchronous version of the interaction.

763 **8.1 Browse and Drill Down Query Support**

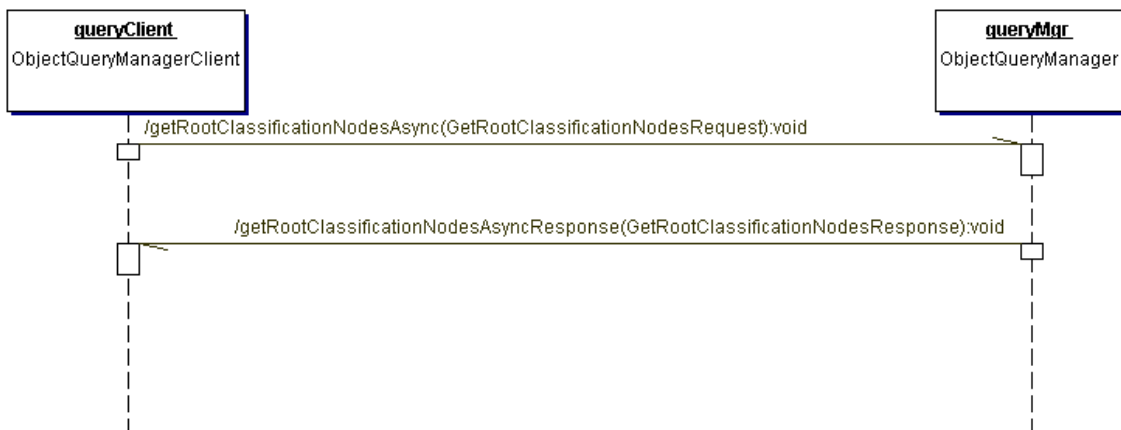
764 The browse and drill down query style is supported by a set of interaction protocols  
 765 between the ObjectQueryManagerClient and the ObjectQueryManager. Sections 8.1.1,  
 766 8.1.2 and 8.1.3 describe these protocols.

767 **8.1.1 Get Root Classification Nodes Request**

768 An ObjectQueryManagerClient sends this request to get a list of root  
 769 ClassificationNodes defined in the repository. Root classification nodes are defined as  
 770 nodes that have no parent. Note that it is possible to specify a namePattern attribute  
 771 that can filter on the name attribute of the root ClassificationNodes. The namePattern  
 772 must be specified using a wildcard pattern defined by SQL-92 LIKE clause as defined  
 773 by [SQL].



774  
 775 **Figure 10: Get Root Classification Nodes Sequence Diagram**



776  
 777 **Figure 11: Get Root Classification Nodes Asynchronous Sequence Diagram**

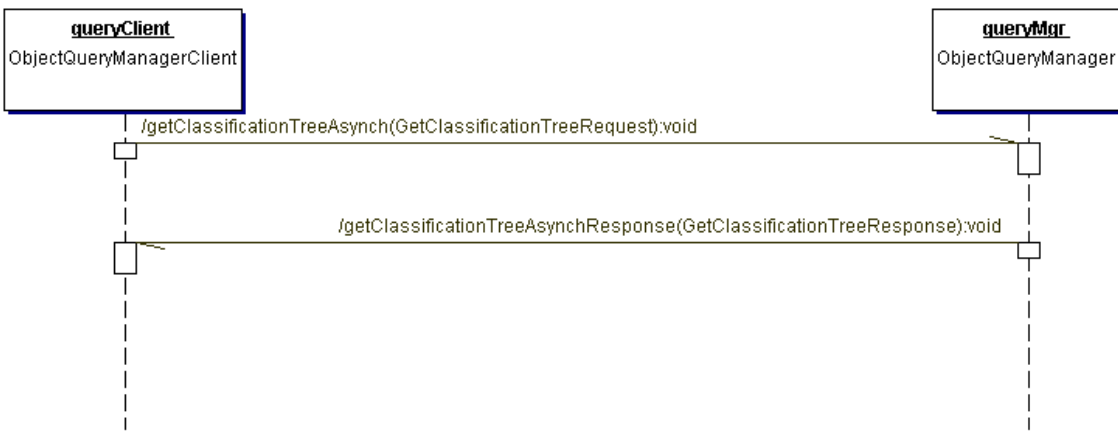
778 For details on the schema for the business documents shown in this process refer to  
 779 Appendix A.2.

780 **8.1.2 Get Classification Tree Request**

781 An ObjectQueryManagerClient sends this request to get the ClassificationNode sub-tree  
 782 defined in the repository under the ClassificationNodes specified in the request. Note  
 783 that a GetClassificationTreeRequest can specify an integer attribute called *depth* to get  
 784 the sub-tree up to the specified depth. If *depth* is the default value of 1, then only the  
 785 immediate children of the specified ClassificationNodeList are returned. If *depth* is 0 or a  
 786 negative number then the entire sub-tree is retrieved.



787  
 788 **Figure 12: Get Classification Tree Sequence Diagram**



789  
 790 **Figure 13: Get Classification Tree Asynchronous Sequence Diagram**

791 For details on the schema for the business documents shown in this process refer to  
 792 Appendix A.2.

793 **8.1.3 Get Classified Objects Request**

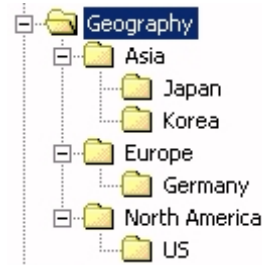
794 An ObjectQueryManagerClient sends this request to get a list of RegistryEntries that are  
 795 classified by all of the specified ClassificationNodes (or any of their descendants), as  
 796 specified by the ObjectRefList in the request.

797 It is possible to get RegistryEntries based on matches with multiple classifications. Note  
798 that specifying a ClassificationNode is implicitly specifying a logical OR with all  
799 descendants of the specified ClassificationNode.

800 When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it should  
801 return Objects that are:

- 802 1. Either directly classified by the specified ClassificationNode
- 803 2. Or are directly classified by a descendant of the specified ClassificationNode

#### 804 8.1.3.1 Get Classified Objects Request Example



805

806

Figure 14: A Sample Geography Classification

807 Let us say a classification tree has the structure shown in Figure 14:

- 808 • If the Geography node is specified in the GetClassifiedObjectsRequest then the  
809 GetClassifiedObjectsResponse should include all RegistryEntries that are directly  
810 classified by Geography *or* North America *or* US *or* Asia *or* Japan *or* Korea *or*  
811 Europe *or* Germany.
- 812 • If the Asia node is specified in the GetClassifiedObjectsRequest then the  
813 GetClassifiedObjectsResponse should include all RegistryEntries that are directly  
814 classified by Asia *or* Japan *or* Korea.
- 815 • If the Japan *and* Korea nodes are specified in the GetClassifiedObjectsRequest  
816 then the GetClassifiedObjectsResponse should include all RegistryEntries that  
817 are directly classified by both Japan *and* Korea.
- 818 • If the North America *and* Asia node is specified in the  
819 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should  
820 include all RegistryEntries that are directly classified by (North America *or* US)  
821 *and* (Asia *or* Japan *or* Korea).

822

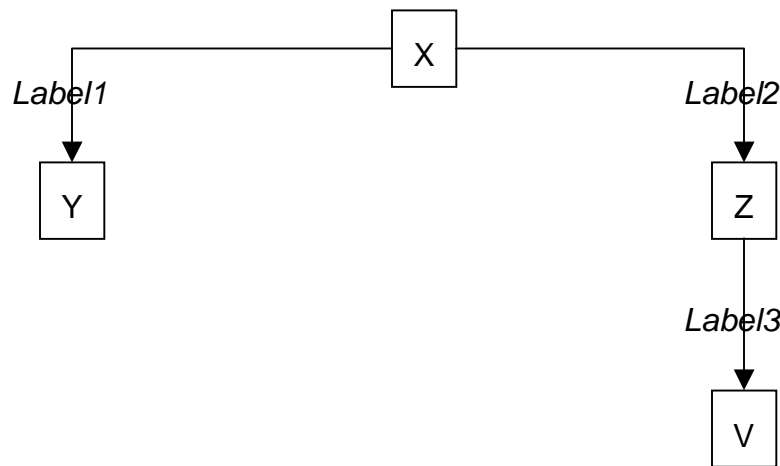
823

823 **8.2 Filter Query Support**

824 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML  
 825 conforming Registry implementation. Each query alternative is directed against a single  
 826 class defined by the ebXML Registry Information Model (ebRIM). The result of such a  
 827 query is a set of identifiers for instances of that class. A FilterQuery may be a stand-  
 828 alone query or it may be the initial action of a ReturnRegistryEntry query or a  
 829 ReturnRepositoryItem query.

830 A client submits a FilterQuery, a ReturnRegistryEntry query, or a ReturnRepositoryItem  
 831 query to the ObjectQueryManager as part of an AdhocQueryRequest. The  
 832 ObjectQueryManager sends an AdhocQueryResponse back to the client, enclosing the  
 833 appropriate FilterQueryResponse, ReturnRegistryEntryResponse, or  
 834 ReturnRepositoryItemResponse specified herein. The sequence diagrams for  
 835 AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.4.

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



842  
843  
844  
845  
846  
847  
848  
849  
850  
851 **Figure 15: Example ebRIM Binding**

852 Label1 identifies an association from X to Y, Label2 identifies an association from X to  
 853 Z, and Label3 identifies an association from Z to V. Labels can be omitted if there is no  
 854 ambiguity as to which ebRIM association is intended. The name of the query is  
 855 determined by the root class, i.e. this is an ebRIM Binding for an XQuery. The Y node in  
 856 the tree is limited to the set of Y instances that are linked to X by the association  
 857 identified by Label1. Similarly, the Z and V nodes are limited to instances that are linked  
 858 to their parent node by the identified association.



859 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter  
860 is a restricted *predicate clause* over the attributes of a single class. The supported class  
861 filters are specified in Section 8.2.9 and the supported predicate clauses are defined in  
862 Section 8.2.10. A FilterQuery will be composed of elements that traverse the tree to  
863 determine which branches satisfy the designated class filters, and the query result will  
864 be the set of root node instances that support such a branch.

865 In the above example, the XQuery element will have three subelements, one an XFilter  
866 on the X class to eliminate X instances that do not satisfy the predicate of the XFilter,  
867 another a YFilter on the Y class to eliminate branches from X to Y where the target of  
868 the association does not satisfy the YFilter, and a third to eliminate branches along a  
869 path from X through Z to V. The third element is called a *branch* element because it  
870 allows class filters on each class along the path from X to V. In general, a branch  
871 element will have subelements that are themselves class filters, other branch elements,  
872 or a full blown query on the terminal class in the path.

873 If an association from a class X to a class Y is one-to-zero or one-to-one, then at most  
874 one branch or filter element on Y is allowed. However, if the association is one-to-many,  
875 then multiple filter or branch elements are allowed. This allows one to specify that an  
876 instance of X must have associations with multiple instances of Y before the instance of  
877 X is said to satisfy the branch element.

878 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is  
879 intended to be stable, the FilterQuery syntax is stable. However, if new structures are  
880 added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the  
881 same time.

882 Support for FilterQuery is required of every conforming ebXML Registry implementation,  
883 but other query options are possible. The Registry will hold a self-describing CPP that  
884 identifies all supported AdhocQuery options. This profile is described in Section 6.1.

885 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.6 below identify the virtual  
886 hierarchy for each FilterQuery alternative. The Semantic Rules for each query  
887 alternative specify the effect of that binding on query semantics.

888 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a  
889 way to structure an XML document as an expansion of the result of a  
890 RegistryEntryQuery. The ReturnRegistryEntry element specified in Section 8.2.7 allows  
891 one to specify what metadata one wants returned with each registry entry identified in  
892 the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section  
893 8.2.8 allows one to specify what repository items one wants returned based on their  
894 relationships to the registry entries identified by the result of a RegistryEntryQuery.  
895

## 895 8.2.1 FilterQuery

### 896 Purpose

897 To identify a set of registry instances from a specific registry class. Each alternative  
898 assumes a specific binding to ebRIM. The query result for each query alternative is a  
899 set of references to instances of the root class specified by the binding. The  
900 StatusResult is a success indication or a collection of warnings and/or exceptions.

### 901 Definition

```
902
903 <!ELEMENT FilterQuery
904   (
905     | RegistryEntryQuery
906     | AuditableEventQuery
907     | ClassificationNodeQuery
908     | RegistryPackageQuery
909     | OrganizationQuery          )>
910
911 <!ELEMENT FilterQueryResult
912   (
913     | RegistryEntryQueryResult
914     | AuditableEventQueryResult
915     | ClassificationNodeQueryResult
916     | RegistryPackageQueryResult
917     | OrganizationQueryResult  )>
918
919 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
920
921 <!ELEMENT RegistryEntryView EMPTY >
922 <!ATTLIST RegistryEntryView
923   objectURN      CDATA      #REQUIRED
924   contentURI     CDATA      #IMPLIED
925   objectID       CDATA      #IMPLIED >
926
927 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
928
929 <!ELEMENT AuditableEventView EMPTY >
930 <!ATTLIST AuditableEventView
931   objectID       CDATA      #REQUIRED
932   timestamp      CDATA      #REQUIRED >
933
934 <!ELEMENT ClassificationNodeQueryResult
935   (ClassificationNodeView*)>
936
937 <!ELEMENT ClassificationNodeView EMPTY >
938 <!ATTLIST ClassificationNodeView
939   objectURN      CDATA      #REQUIRED
940   contentURI     CDATA      #IMPLIED
941   objectID       CDATA      #IMPLIED >
942
943 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
944
945 <!ELEMENT RegistryPackageView EMPTY >
946 <!ATTLIST RegistryPackageView
```

```

945     objectURN      CDATA      #REQUIRED
946     contentURI     CDATA      #IMPLIED
947     objectID       CDATA      #IMPLIED >
948
949     <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
950
951     <!ELEMENT OrganizationView EMPTY >
952     <!ATTLIST OrganizationView
953         orgURN      CDATA      #REQUIRED
954         objectID     CDATA      #IMPLIED >
955
956     <!ELEMENT StatusResult ( Success | ( Exception | Warning )+ )>
957
958     <!ELEMENT Success EMPTY >
959
960     <!ELEMENT Exception ( #PCDATA )>
961     <!ATTLIST Exception
962         code      CDATA      #REQUIRED >
963
964     <!ELEMENT Warning ( #PCDATA )>
965     <!ATTLIST Warning
966         code      CDATA      #REQUIRED >

```

967 **Semantic Rules**

- 968 1. The semantic rules for each FilterQuery alternative are specified in subsequent  
969 subsections.
- 970 2. Each FilterQueryResult is a set of XML reference elements to identify each instance  
971 of the result set. Each XML attribute carries a value derived from the value of an  
972 attribute specified in the Registry Information Model as follows:
  - 973 a) objectID is the value of the ID attribute of the RegistryObject class,
  - 974 b) objectURN and orgURN are URN values derived from the object ID,
  - 975 c) contentURI is a URL value derived from the contentURI attribute of the  
976 RegistryEntry class,
  - 977 d) timestamp is a literal value to represent the value of the timestamp attribute of  
978 the AuditableEvent class.
- 979 3. An Exception indicates that The FilterQuery was not successful, so the  
980 FilterQueryResult is empty. A warning indicates that the FilterQuery was successful,  
981 so the FilterQueryResult is accurate, but the warning may give additional information  
982 back to the user.
- 983 4. If any exception or warning results, then it is returned as the appropriate alternative  
984 of the StatusResult element. In an ebXML Message Services environment,  
985 Exceptions and Warnings will map to an "ErrorList" element as specified by [ebTRP].  
986 See Appendix A.1.
- 987

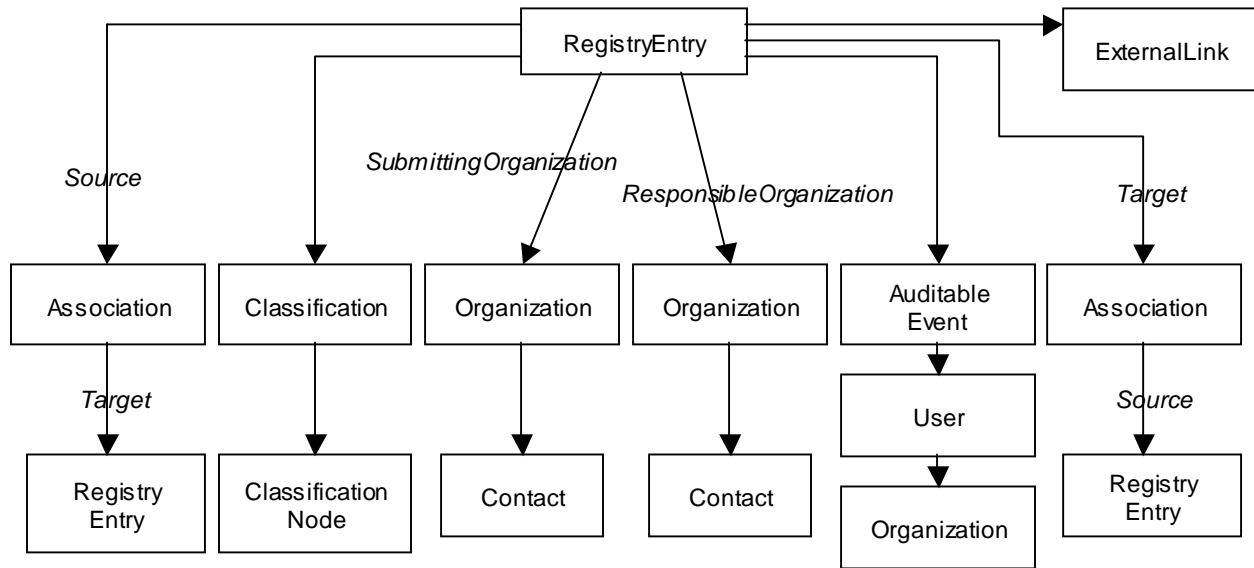
987 **8.2.2 RegistryEntryQuery**

988 **Purpose**

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

991 **ebRIM Binding**

992



993

994 **Definition**

```

995
996 <!ELEMENT RegistryEntryQuery
997   ( RegistryEntryFilter?,
998     SourceAssociationBranch*,
999     TargetAssociationBranch*,
1000     HasClassificationBranch*,
1001     SubmittingOrganizationBranch?,
1002     ResponsibleOrganizationBranch?,
1003     ExternalLinkFilter*,
1004     HasAuditableEventBranch* )>
1005
1006 <!ELEMENT SourceAssociationBranch
1007   ( AssociationFilter?,
1008     RegistryEntryFilter? )>
1009
1010 <!ELEMENT TargetAssociationBranch
1011   ( AssociationFilter?,
1012     RegistryEntryFilter? )>
1013
1014 <!ELEMENT HasClassificationBranch
1015   ( ClassificationFilter?,

```

```
1016         ClassificationNodeFilter?           )>
1017
1018     <!ELEMENT SubmittingOrganizationBranch
1019     (   OrganizationFilter?,
1020         ContactFilter?                       )>
1021
1022     <!ELEMENT ResponsibleOrganizationBranch
1023     (   OrganizationFilter?,
1024         ContactFilter?                       )>
1025
1026     <!ELEMENT HasAuditableEventBranch
1027     (   AuditableEventFilter?,
1028         UserFilter?,
1029         OrganizationFilter?                 )>
```

### 1030 **Semantic Rules**

- 1031 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The  
1032 following steps will eliminate instances in RE that do not satisfy the conditions of the  
1033 specified filters.
- 1034 a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below;  
1035 otherwise, let x be a registry entry in RE. If x does not satisfy the  
1036 RegistryEntryFilter as defined in Section 8.2.9, then remove x from RE.
- 1037 b) If a SourceAssociationBranch element is not specified, or if RE is empty, then  
1038 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1039 source object of some Association instance, then remove x from RE; otherwise,  
1040 treat each SourceAssociationBranch element separately as follows:
- 1041 If no AssociationFilter is specified within SourceAssociationBranch, then let AF  
1042 be the set of all Association instances that have x as a source object; otherwise,  
1043 let AF be the set of Association instances that satisfy the AssociationFilter and  
1044 have x as the source object. If AF is empty, then remove x from RE. If no  
1045 RegistryEntryFilter is specified within SourceAssociationBranch, then let RET be  
1046 the set of all RegistryEntry instances that are the target object of some element  
1047 of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the  
1048 RegistryEntryFilter and are the target object of some element of AF. If RET is  
1049 empty, then remove x from RE.
- 1050 c) If a TargetAssociationBranch element is not specified, or if RE is empty, then  
1051 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1052 target object of some Association instance, then remove x from RE; otherwise,  
1053 treat each TargetAssociationBranch element separately as follows:

- 1054 If no AssociationFilter is specified within TargetAssociationBranch, then let AF be  
1055 the set of all Association instances that have x as a target object; otherwise, let  
1056 AF be the set of Association instances that satisfy the AssociationFilter and have  
1057 x as the target object. If AF is empty, then remove x from RE. If no  
1058 RegistryEntryFilter is specified within TargetAssociationBranch, then let RES be  
1059 the set of all RegistryEntry instances that are the source object of some element  
1060 of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the  
1061 RegistryEntryFilter and are the source object of some element of AF. If RES is  
1062 empty, then remove x from RE.
- 1063 d) If a HasClassificationBranch element is not specified, or if RE is empty, then  
1064 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1065 source object of some Classification instance, then remove x from RE; otherwise,  
1066 treat each HasClassificationBranch element separately as follows:
- 1067 If no ClassificationFilter is specified within the HasClassificationBranch, then let  
1068 CL be the set of all Classification instances that have x as a source object;  
1069 otherwise, let CL be the set of Classification instances that satisfy the  
1070 ClassificationFilter and have x as the source object. If CL is empty, then remove  
1071 x from RE. If no ClassificationNodeFilter is specified within  
1072 HasClassificationBranch, then let CN be the set of all ClassificationNode  
1073 instances that are the target object of some element of CL; otherwise, let CN be  
1074 the set of RegistryEntry instances that satisfy the ClassificationNodeFilter and  
1075 are the target object of some element of CL. If CN is empty, then remove x from  
1076 RE.
- 1077 e) If a SubmittingOrganizationBranch element is not specified, or if RE is empty,  
1078 then continue below; otherwise, let x be a remaining registry entry in RE. If x  
1079 does not have a submitting organization, then remove x from RE. If no  
1080 OrganizationFilter is specified within SubmittingOrganizationBranch, then let SO  
1081 be the set of all Organization instances that are the submitting organization for x;  
1082 otherwise, let SO be the set of Organization instances that satisfy the  
1083 OrganizationFilter and are the submitting organization for x. If SO is empty, then  
1084 remove x from RE. If no ContactFilter is specified within  
1085 SubmittingOrganizationBranch, then let CT be the set of all Contact instances  
1086 that are the contacts for some element of SO; otherwise, let CT be the set of  
1087 Contact instances that satisfy the ContactFilter and are the contacts for some  
1088 element of SO. If CT is empty, then remove x from RE.

- 1089 f) If a ResponsibleOrganizationBranch element is not specified, or if RE is empty,  
1090 then continue below; otherwise, let x be a remaining registry entry in RE. If x  
1091 does not have a responsible organization, then remove x from RE. If no  
1092 OrganizationFilter is specified within ResponsibleOrganizationBranch, then let  
1093 RO be the set of all Organization instances that are the responsible organization  
1094 for x; otherwise, let RO be the set of Organization instances that satisfy the  
1095 OrganizationFilter and are the responsible organization for x. If RO is empty, then  
1096 remove x from RE. If no ContactFilter is specified within  
1097 SubmittingOrganizationBranch, then let CT be the set of all Contact instances  
1098 that are the contacts for some element of RO; otherwise, let CT be the set of  
1099 Contact instances that satisfy the ContactFilter and are the contacts for some  
1100 element of RO. If CT is empty, then remove x from RE.
- 1101 g) If an ExternalLinkFilter element is not specified, or if RE is empty, then continue  
1102 below; otherwise, let x be a remaining registry entry in RE. If x is not linked to  
1103 some ExternalLink instance, then remove x from RE; otherwise, treat each  
1104 ExternalLinkFilter element separately as follows:
- 1105 Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter and  
1106 are linked to x. If EL is empty, then remove x from RE.
- 1107 h) If a HasAuditableEventBranch element is not specified, or if RE is empty, then  
1108 continue below; otherwise, let x be a remaining registry entry in RE. If x is not  
1109 linked to some AuditableEvent instance, then remove x from RE; otherwise, treat  
1110 each HasAuditableEventBranch element separately as follows:
- 1111 If an AuditableEventFilter is not specified within HasAuditableEventBranch, then  
1112 let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the  
1113 set of AuditableEvent instances that satisfy the AuditableEventFilter and are  
1114 auditable events for x. If AE is empty, then remove x from RE. If a UserFilter is  
1115 not specified within HasAuditableEventBranch, then let AI be the set of all User  
1116 instances linked to an element of AE; otherwise, let AI be the set of User  
1117 instances that satisfy the UserFilter and are linked to an element of AE. If AI is  
1118 empty, then remove x from RE. If an OrganizationFilter is not specified within  
1119 HasAuditableEventBranch, then let OG be the set of all Organization instances  
1120 that are linked to an element of AI; otherwise, let OG be the set of Organization  
1121 instances that satisfy the OrganizationFilter and are linked to an element of AI. If  
1122 OG is empty, then remove x from RE.
- 1123 2. If RE is empty, then raise the warning: *registry entry query result is empty*;  
1124 otherwise, return RE as the result of the RegistryEntryQuery.
- 1125 3. Return any accumulated warnings or exceptions as the StatusResult associated with  
1126 the RegistryEntryQuery.
- 1127
- 1128
- 1129

1130 **Examples**

1131 A client wants to establish a trading relationship with XYZ Corporation and wants to  
 1132 know if they have registered any of their business documents in the Registry. The  
 1133 following query returns a set of registry entry identifiers for currently registered items  
 1134 submitted by any organization whose name includes the string "XYZ". It does not return  
 1135 any registry entry identifiers for superceded, replaced, deprecated, or withdrawn items.  
 1136

```

1137 <RegistryEntryQuery>
1138   <RegistryEntryFilter>
1139     status EQUAL "Approved"           -- code by Clause, Section 8.2.10
1140   </RegistryEntryFilter>
1141   <SubmittingOrganizationBranch>
1142     <OrganizationFilter>
1143       name CONTAINS "XYZ"           -- code by Clause, Section 8.2.10
1144     </OrganizationFilter>
1145   </SubmittingOrganizationBranch>
1146 </RegistryEntryQuery>
1147
```

1148 A client is using the United Nations Standard Product and Services Classification  
 1149 (UNSPSC) scheme and wants to identify all companies that deal with products  
 1150 classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client  
 1151 knows that companies have registered their party profile documents in the Registry, and  
 1152 that each profile has been classified by the products the company deals with. The  
 1153 following query returns a set of registry entry identifiers for profiles of companies that  
 1154 deal with integrated circuit components.  
 1155

```

1156 <RegistryEntryQuery>
1157   <RegistryEntryFilter>
1158     objectType EQUAL "CPP" AND       -- code by Clause, Section 8.2.10
1159     status EQUAL "Approved"
1160   </RegistryEntryFilter>
1161   <HasClassificationBranch>
1162     <ClassificationNodeFilter>
1163       id STARTSWITH "urn:un:spsc:321118" -- code by Clause, Section 8.2.10
1164     </ClassificationNodeFilter>
1165   </HasClassificationBranch>
1166 </RegistryEntryQuery>

```

1167

1168 A client application needs all items that are classified by two different classification  
 1169 schemes, one based on "Industry" and another based on "Geography". Both schemes  
 1170 have been defined by ebXML and are registered. The root nodes of each scheme are  
 1171 identified by "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The  
 1172 following query identifies registry entries for all registered items that are classified by  
 1173 "Industry/Automotive" and by "Geography/Asia/Japan".

```

1174 <RegistryEntryQuery>
1175   <HasClassificationBranch>
1176     <ClassificationNodeFilter>

```



```
1178         id STARTSWITH "urn:ebxml:cs:industry" AND
1179         path EQUAL "Industry/Automotive"      -- code by Clause, Section 8.2.10
1180     </ClassificationNodeFilter>
1181     <ClassificationNodeFilter>
1182         id STARTSWITH "urn:ebxml:cs:geography" AND
1183         path EQUAL "Geography/Asia/Japan"    -- code by Clause, Section 8.2.10
1184     </ClassificationNodeFilter>
1185     </HasClassificationBranch>
1186 </RegistryEntryQuery>
```

1187

1188 A client application wishes to identify all registry Package instances that have a given  
1189 registry entry as a member of the package. The following query identifies all registry  
1190 packages that contain the registry entry identified by URN "urn:path:myitem" as a  
1191 member:

1192

```
1193     <RegistryEntryQuery>
1194         <RegistryEntryFilter>
1195             objectType EQUAL "RegistryPackage"    -- code by Clause, Section 8.2.10
1196         </RegistryEntryFilter>
1197         <SourceAssociationBranch>
1198             <AssociationFilter>                  -- code by Clause, Section 8.2.10
1199                 associationType EQUAL "HasMember" AND
1200                 targetObject EQUAL "urn:path:myitem"
1201             </AssociationFilter>
1202         </SourceAssociationBranch>
1203     </RegistryEntryQuery>
```

1204

1205 A client application wishes to identify all ClassificationNode instances that have some  
1206 given keyword as part of their name or description. The following query identifies all  
1207 registry classification nodes that contain the keyword "transistor" as part of their name  
1208 or as part of their description.

1209

```
1210     <RegistryEntryQuery>
1211         <RegistryEntryFilter>
1212             ObjectType="ClassificationNode" AND
1213             (name CONTAINS "transistor" OR      -- code by Clause, Section 8.2.10
1214              description CONTAINS "transistor")
1215         </RegistryEntryFilter>
1216     </RegistryEntryQuery>
```

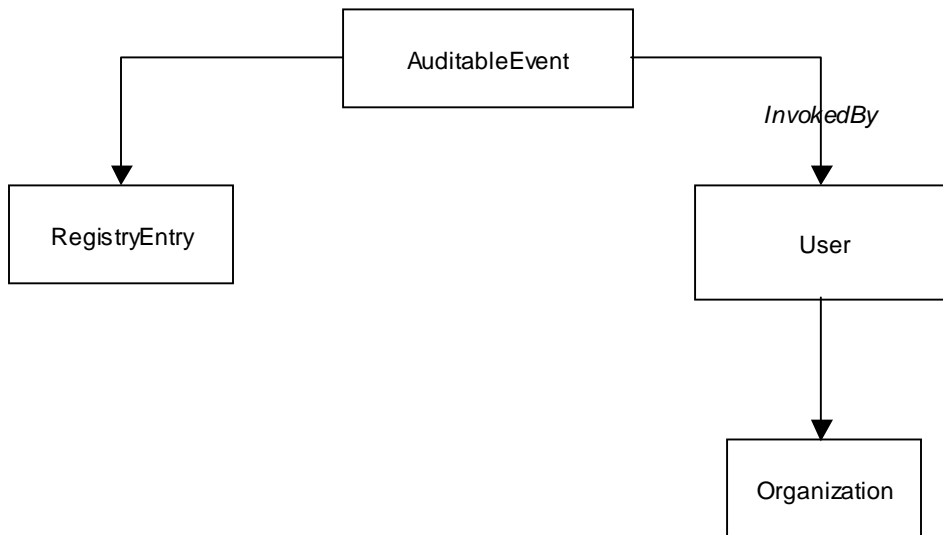
1217

1217 **8.2.3 AuditableEventQuery**

1218 **Purpose**

1219 To identify a set of auditable event instances as the result of a query over selected  
 1220 registry metadata.

1221 **ebRIM Binding**



1222 **Definition**

```

1223
1224 <!ELEMENT AuditableEventQuery
1225 ( AuditableEventFilter?,
1226 RegistryEntryQuery*,
1227 InvokedByBranch? )>
1228
1229 <!ELEMENT InvokedByBranch
1230 ( UserFilter?,
1231 OrganizationQuery? )>
    
```

1232

1233 **Semantic Rules**

- 1234 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The  
 1235 following steps will eliminate instances in AE that do not satisfy the conditions of the  
 1236 specified filters.

1237

- 1238 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below;  
1239 otherwise, let x be an auditable event in AE. If x does not satisfy the  
1240 AuditableEventFilter as defined in Section 8.2.9, then remove x from AE.
- 1241 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue  
1242 below; otherwise, let x be a remaining auditable event in AE. Treat each  
1243 RegistryEntryQuery element separately as follows:
- 1244 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If  
1245 x is not an auditable event for some registry entry in RE, then remove x from AE.
- 1246 c) If an InvokedByBranch element is not specified, or if AE is empty, then continue  
1247 below; otherwise, let x be a remaining auditable event in AE.
- 1248 Let u be the user instance that invokes x. If a UserFilter element is specified  
1249 within the InvokedByBranch, and if u does not satisfy that filter, then remove x  
1250 from AE; otherwise, continue below.
- 1251 If an OrganizationQuery element is not specified within the InvokedByBranch,  
1252 then continue below; otherwise, let OG be the set of Organization instances that  
1253 are identified by the organization attribute of u and are in the result set of the  
1254 OrganizationQuery. If OG is empty, then remove x from AE.
- 1255 2. If AE is empty, then raise the warning: *auditable event query result is empty*.
- 1256 3. Return AE as the result of the AuditableEventQuery.
- 1257 4. Return any accumulated warnings or exceptions as the StatusResult associated with  
1258 the AuditableEventQuery.

### 1259 Examples

1260 A Registry client has registered an item and it has been assigned a URN identifier  
1261 "urn:path:myitem". The client is now interested in all events since the beginning of the  
1262 year that have impacted that item. The following query will return a set of  
1263 AuditableEvent identifiers for all such events.

```
1264 <AuditableEventquery>  
1265   <AuditableEventFilter>  
1266     timestamp GE "2001-01-01" AND -- code by Clause, Section 8.2.10  
1267     registryEntry EQUAL "urn:path:myitem"  
1268   </AuditableEventFilter>  
1269 </AuditableEventQuery>
```

1271

1272 A client company has many registered objects in the Registry. The Registry allows  
1273 events submitted by other organizations to have an impact on your registered items,  
1274 e.g. new classifications and new associations. The following query will return a set of  
1275 identifiers for all auditable events, invoked by some other party, that had an impact on  
1276 an item submitted by "myorg" and for which "myorg" is the responsible organization.

1277

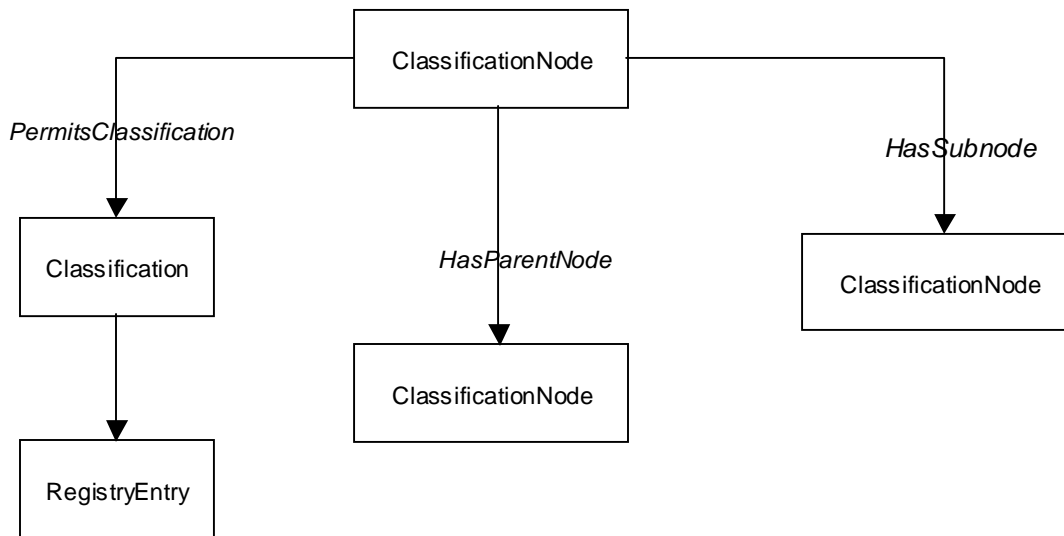
```
1278     <AuditableEventQuery>
1279         <RegistryEntryQuery>
1280             <SubmittingOrganizationBranch>
1281                 <OrganizationFilter>
1282                     id EQUAL "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1283                 </OrganizationFilter>
1284             </SubmittingOrganizationBranch>
1285             <ResponsibleOrganizationBranch>
1286                 <OrganizationFilter>
1287                     id EQUAL "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1288                 </OrganizationFilter>
1289             </ResponsibleOrganizationBranch>
1290         </RegistryEntryQuery>
1291         <InvokedByBranch>
1292             <OrganizationQuery>
1293                 <OrganizationFilter>
1294                     id -EQUAL "urn:somepath:myorg"         -- code by Clause, Section 8.2.10
1295                 </OrganizationFilter>
1296             </OrganizationQuery>
1297         </InvokedByBranch>
1298     </AuditableEventQuery>
1299
```

1299 **8.2.4 ClassificationNodeQuery**

1300 **Purpose**

1301 To identify a set of classification node instances as the result of a query over selected  
 1302 registry metadata.

1303 **ebRIM Binding**



1304 **Definition**

```

    1305
    1306 <!--ELEMENT ClassificationNodeQuery
    1307   ( ClassificationNodeFilter?,
    1308     PermitsClassificationBranch*,
    1309     HasParentNode?,
    1310     HasSubnode*           )>
    1311
    1312 <!--ELEMENT PermitsClassificationBranch
    1313   ( ClassificationFilter?,
    1314     RegistryEntryQuery?   )>
    1315
    1316 <!--ELEMENT HasParentNode
    1317   ( ClassificationNodeFilter?,
    1318     HasParentNode?        )>
    1319
    1320 <!--ELEMENT HasSubnode
    1321   ( ClassificationNodeFilter?,
    1322     HasSubnode*           )>
    
```

1323

1324

1325 **Semantic Rules**

- 1326 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry.  
1327 The following steps will eliminate instances in CN that do not satisfy the conditions of  
1328 the specified filters.
- 1329 a) If a ClassificationNodeFilter is not specified, or if CN is empty, then continue  
1330 below; otherwise, let x be a classification node in CN. If x does not satisfy the  
1331 ClassificationNodeFilter as defined in Section 8.2.9, then remove x from AE.
- 1332 b) If a PermitsClassificationBranch element is not specified, or if CN is empty, then  
1333 continue below; otherwise, let x be a remaining classification node in CN. If x is  
1334 not the target object of some Classification instance, then remove x from CN;  
1335 otherwise, treat each PermitsClassificationBranch element separately as follows:
- 1336 If no ClassificationFilter is specified within the PermitsClassificationBranch  
1337 element, then let CL be the set of all Classification instances that have x as the  
1338 target object; otherwise, let CL be the set of Classification instances that satisfy  
1339 the ClassificationFilter and have x as the target object. If CL is empty, then  
1340 remove x from CN. If no RegistryEntryQuery is specified within the  
1341 PermitsClassificationBranch element, then let RES be the set of all RegistryEntry  
1342 instances that are the source object of some classification instance in CL;  
1343 otherwise, let RE be the result set of the RegistryEntryQuery as defined in  
1344 Section 8.2.2 and let RES be the set of all instances in RE that are the source  
1345 object of some classification in CL. If RES is empty, then remove x from CN.
- 1346 c) If a HasParentNode element is not specified, or if CN is empty, then continue  
1347 below; otherwise, let x be a remaining classification node in CN and execute the  
1348 following paragraph with  $n=x$ .
- 1349 Let n be a classification node instance. If n does not have a parent node (i.e. if n  
1350 is a root node), then remove x from CN. Let p be the parent node of n. If a  
1351 ClassificationNodeFilter element is directly contained in HasParentNode and if p  
1352 does not satisfy the ClassificationNodeFilter, then remove x from CN.
- 1353 If another HasParentNode element is directly contained within this  
1354 HasParentNode element, then repeat the previous paragraph with  $n=p$ .
- 1355 d) If a HasSubnode element is not specified, or if CN is empty, then continue below;  
1356 otherwise, let x be a remaining classification node in CN. If x is not the parent  
1357 node of some ClassificationNode instance, then remove x from CN; otherwise,  
1358 treat each HasSubnode element separately and execute the following paragraph  
1359 with  $n = x$ .
- 1360 Let n be a classification node instance. If a ClassificationNodeFilter is not  
1361 specified within the HasSubnode element then let CNC be the set of all  
1362 classification nodes that have n as their parent node; otherwise, let CNC be the  
1363 set of all classification nodes that satisfy the ClassificationNodeFilter and have n  
1364 as their parent node. If CNC is empty then remove x from CN; otherwise, let y be  
1365 an element of CNC and continue with the next paragraph.

- 1366 If the HasSubnode element is terminal, i.e. if it does not directly contain another  
1367 HasSubnode element, then continue below; otherwise, repeat the previous  
1368 paragraph with the new HasSubnode element and with  $n = y$ .
- 1369 2. If CN is empty, then raise the warning: *classification node query result is empty*.
  - 1370 3. Return CN as the result of the ClassificationNodeQuery.
  - 1371 4. Return any accumulated warnings or exceptions as the StatusResult associated with  
1372 the ClassificationNodeQuery.

### 1373 Examples

1374 A client application wishes to identify all classification nodes defined in the Registry that  
1375 are root nodes and have a name that contains the phrase “product code” or the phrase  
1376 “product type”. Note: By convention, if a classification node has no parent (i.e. is a root  
1377 node), then the parent attribute of that instance is set to null and is represented as a  
1378 literal by a zero length string.

```
1380 <ClassificationNodeQuery>
1381   <ClassificationNodeFilter>
1382     (name CONTAINS "product code" OR      -- code by Clause, Section 8.2.10
1383      name CONTAINS "product type") AND
1384     parent EQUAL ""
1385   </ClassificationNodeFilter>
1386 </ClassificationNodeQuery>
```

1387

1388 A client application wishes to identify all of the classification nodes at the third level of a  
1389 classification scheme hierarchy. The client knows that the URN identifier for the root  
1390 node is “urn:ebxml:cs:myroot”. The following query identifies all nodes at the second  
1391 level under “myroot” (i.e. third level overall).

```
1393 <ClassificationNodeQuery>
1394   <HasParentNode>
1395     <HasParentNode>
1396       <ClassificationNodeFilter>
1397         id EQ "urn:ebxml:cs:myroot"  -- code by Clause, Section 8.2.10
1398       </ClassificationNodeFilter>
1399     </HasParentNode>
1400   </HasParentNode>
1401 </ClassificationNodeQuery>
```

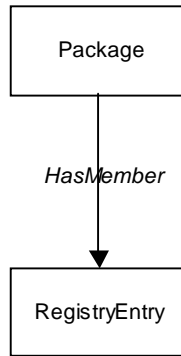
1402

1402 **8.2.5 RegistryPackageQuery**

1403 **Purpose**

1404 To identify a set of registry package instances as the result of a query over selected  
 1405 registry metadata.

1406 **ebRIM Binding**



1407 **Definition**

```

1408
1409 <!ELEMENT RegistryPackageQuery
1410 ( PackageFilter?,
1411 HasMemberBranch* )>
1412
1413 <!ELEMENT HasMemberBranch
1414 ( RegistryEntryQuery? )>
    
```

1415

1416 **Semantic Rules**

1417 1. Let RP denote the set of all persistent Package instances in the Registry. The  
 1418 following steps will eliminate instances in RP that do not satisfy the conditions of the  
 1419 specified filters.

- 1420 a) If a PackageFilter is not specified, or if RP is empty, then continue below;  
 1421 otherwise, let x be a package instance in RP. If x does not satisfy the  
 1422 PackageFilter as defined in Section 8.2.9, then remove x from RP.
- 1423 b) If a HasMemberBranch element is not directly contained in the  
 1424 RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x  
 1425 be a remaining package instance in RP. If x is an empty package, then remove x  
 1426 from RP; otherwise, treat each HasMemberBranch element separately as  
 1427 follows:

1428



1429 If a RegistryEntryQuery element is not directly contained in the  
1430 HasMemberBranch element, then let PM be the set of all RegistryEntry instances  
1431 that are members of the package x; otherwise, let RE be the set of RegistryEntry  
1432 instances returned by the RegistryEntryQuery as defined in Section 8.2.2 and let  
1433 PM be the subset of RE that are members of the package x. If PM is empty, then  
1434 remove x from RP.

- 1435 2. If RP is empty, then raise the warning: *registry package query result is empty*.
- 1436 3. Return RP as the result of the RegistryPackageQuery.
- 1437 4. Return any accumulated warnings or exceptions as the StatusResult associated with  
1438 the RegistryPackageQuery.

### 1439 Examples

1440 A client application wishes to identify all package instances in the Registry that contain  
1441 an Invoice extrinsic object as a member of the package.  
1442

```
1443 <RegistryPackageQuery>  
1444   <HasMemberBranch>  
1445     <RegistryEntryQuery>  
1446       <RegistryEntryFilter>  
1447         objectType EQ "Invoice"      -- code by Clause, Section 8.2.10  
1448       </RegistryEntryFilter>  
1449     </RegistryEntryQuery>  
1450   </HasMemberBranch>  
1451 </RegistryPackageQuery>  
1452
```

1453 A client application wishes to identify all package instances in the Registry that are not  
1454 empty.

```
1455 <RegistryEntryQuery>  
1456   <HasMemberBranch/>  
1457 </RegistryEntryQuery>  
1458  
1459
```

1460 A client application wishes to identify all package instances in the Registry that are  
1461 empty. Since the RegistryPackageQuery is not set up to do negations, clients will have  
1462 to do two separate RegistryPackageQuery requests, one to find all packages and  
1463 another to find all non-empty packages, and then do the set difference themselves.  
1464 Alternatively, they could do a more complex RegistryEntryQuery and check that the  
1465 packaging association between the package and its members is non-existent.

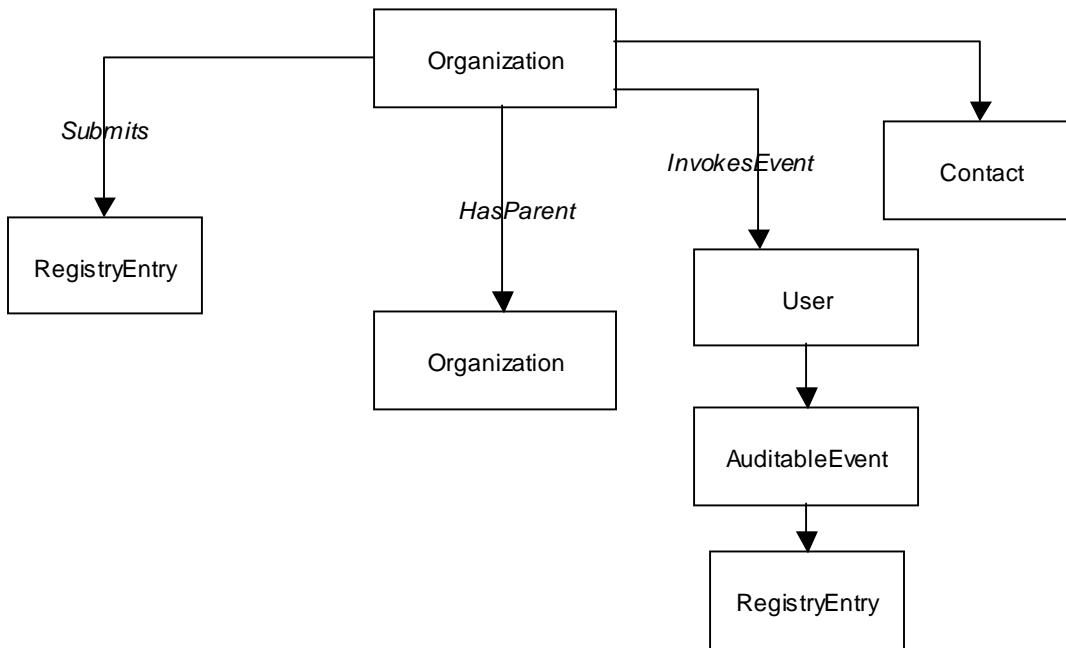
1466 Note: A registry package is an intrinsic RegistryEntry instance that is completely  
1467 determined by its associations with its members. Thus a RegistryPackageQuery can  
1468 always be re-specified as an equivalent RegistryEntryQuery using appropriate "Source"  
1469 and "Target" associations. However, the equivalent RegistryEntryQuery is often more  
1470 complicated to write.  
1471

1471 **8.2.6 OrganizationQuery**

1472 **Purpose**

1473 To identify a set of organization instances as the result of a query over selected registry  
 1474 metadata.

1475 **ebRIM Binding**



1476

1477 **Definition**

```

1478
1479 <!ELEMENT OrganizationQuery
1480 ( OrganizationFilter?,
1481 SubmitsRegistryEntry*,
1482 HasParentOrganization?,
1483 InvokesEventBranch*,
1484 ContactFilter )>
1485
1486 <!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>
1487
1488 <!ELEMENT HasParentOrganization
1489 ( OrganizationFilter?,
1490 HasParentOrganization? )>
1491
1492 <!ELEMENT InvokesEventBranch
1493 ( UserFilter?,
1494 AuditableEventFilter?,
1495 RegistryEntryQuery? )>
    
```

1496 **Semantic Rules**

- 1497 1. Let ORG denote the set of all persistent Organization instances in the Registry. The  
1498 following steps will eliminate instances in ORG that do not satisfy the conditions of  
1499 the specified filters.
- 1500 a) If an OrganizationFilter element is not directly contained in the  
1501 OrganizationQuery element, or if ORG is empty, then continue below; otherwise,  
1502 let x be an organization instance in ORG. If x does not satisfy the  
1503 OrganizationFilter as defined in Section 8.2.9, then remove x from RP.
- 1504 b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery,  
1505 or if ORG is empty, then continue below; otherwise, consider each  
1506 SubmitsRegistryEntry element separately as follows:
- 1507 If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element,  
1508 then let RES be the set of all RegistryEntry instances that have been submitted  
1509 to the Registry by organization x; otherwise, let RE be the result of the  
1510 RegistryEntryQuery as defined in Section 8.2.2 and let RES be the set of all  
1511 instances in RE that have been submitted to the Registry by organization x. If  
1512 RES is empty, then remove x from ORG.
- 1513 c) If a HasParentOrganization element is not specified within the  
1514 OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute  
1515 the following paragraph with o = x:
- 1516 Let o be an organization instance. If an OrganizationFilter is not specified within  
1517 the HasParentOrganization and if o has no parent (i.e. if o is a root organization  
1518 in the Organization hierarchy), then remove x from ORG; otherwise, let p be the  
1519 parent organization of o. If p does not satisfy the OrganizationFilter, then remove  
1520 x from ORG.
- 1521 If another HasParentOrganization element is directly contained within this  
1522 HasParentOrganization element, then repeat the previous paragraph with o = p.
- 1523 d) If an InvokesEventBranch element is not specified within the OrganizationQuery,  
1524 or if ORG is empty, then continue below; otherwise, consider each  
1525 InvokesEventBranch element separately as follows:
- 1526 If an UserFilter is not specified, and if x is not the submitting organization of some  
1527 AuditableEvent instance, then remove x from ORG. If an AuditableEventFilter is  
1528 not specified, then let AE be the set of all AuditableEvent instances that have x  
1529 as the submitting organization; otherwise, let AE be the set of AuditableEvent  
1530 instances that satisfy the AuditableEventFilter and have x as the submitting  
1531 organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is  
1532 not specified in the InvokesEventBranch element, then let RES be the set of all  
1533 RegistryEntry instances associated with an event in AE; otherwise, let RE be the  
1534 result set of the RegistryEntryQuery, as specified in Section 8.2.2, and let RES  
1535 be the subset of RE of entries submitted by x. If RES is empty, then remove x  
1536 from ORG.

1537 e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG is  
1538 empty, then continue below; otherwise, consider each ContactFilter separately as  
1539 follows:

1540 Let CT be the set of Contact instances that satisfy the ContactFilter and are the  
1541 contacts for organization x. If CT is empty, then remove x from ORG.

1542 2. If ORG is empty, then raise the warning: *organization query result is empty*.

1543 3. Return ORG as the result of the OrganizationQuery.

1544 4. Return any accumulated warnings or exceptions as the StatusResult associated with  
1545 the OrganizationQuery.

#### 1546 Examples

1547 A client application wishes to identify a set of organizations, based in France, that have  
1548 submitted a PartyProfile extrinsic object this year.

```
1550 <OrganizationQuery>  
1551   <OrganizationFilter>  
1552     country EQUAL "France"           -- code by Clause, Section 8.2.10  
1553   </OrganizationFilter>  
1554   <SubmitsRegistryEntry>  
1555     <RegistryEntryQuery>  
1556       <RegistryEntryFilter>  
1557         objectType EQUAL "CPP"      -- code by Clause, Section 8.2.10  
1558       </RegistryEntryFilter>  
1559       <HasAuditableEventBranch>  
1560         <AuditableEventFilter>  
1561           timestamp GE "2001-01-01" -- code by Clause, Section 8.2.10  
1562         </AuditableEventFilter>  
1563       </HasAuditableEventBranch>  
1564     </RegistryEntryQuery>  
1565   </SubmitsRegistryEntry>  
1566 </OrganizationQuery>
```

1567

1568 A client application wishes to identify all organizations that have XYZ, Corporation as a  
1569 parent. The client knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is  
1570 no guarantee that subsidiaries of XYZ have a URN that uses the same format, so a full  
1571 query is required.

```
1572  
1573 <OrganizationQuery>  
1574   <HasParentOrganization>  
1575     <OrganizationFilter>  
1576       id EQUAL "urn:ebxml:org:xyz"  -- code by Clause, Section 8.2.10  
1577     </OrganizationFilter>  
1578   </HasParentOrganization>  
1579 </OrganizationQuery>  
1580
```

## 1580 8.2.7 ReturnRegistryEntry

### 1581 Purpose

1582 To construct an XML document that contains selected registry metadata associated with  
 1583 the registry entries identified by a RegistryEntryQuery. NOTE: Initially, the  
 1584 RegistryEntryQuery could be the URN identifier for a single registry entry.

### 1585 Definition

```

1586
1587 <!ELEMENT ReturnRegistryEntry
1588   ( RegistryEntryQuery,
1589     WithClassifications?,
1590     WithSourceAssociations?,
1591     WithTargetAssociations?,
1592     WithAuditableEvents?,
1593     WithExternalLinks? )>
1594
1595 <!ELEMENT WithClassifications ( ClassificationFilter? )>
1596 <!ELEMENT WithSourceAssociations ( AssociationFilter? )>
1597 <!ELEMENT WithTargetAssociations ( AssociationFilter? )>
1598 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>
1599 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
1600
1601 <!ELEMENT ReturnRegistryEntryResult
1602   ( RegistryEntryMetadata*, StatusResult )>
1603
1604 <!ELEMENT RegistryEntryMetadata
1605   ( RegistryEntry,
1606     Classification*,
1607     SourceAssociations?,
1608     TargetAssociations?,
1609     AuditableEvent*,
1610     ExternalLink* )>
1611
1612 <!ELEMENT SourceAssociations ( Association* )>
1613 <!ELEMENT TargetAssociations ( Association* )>

```

### 1614 Semantic Rules

- 1615 1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink  
 1616 elements contained in the ReturnRegistryEntryResult are defined by the ebXML  
 1617 Registry DTD specified in Appendix A.2.
- 1618 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in  
 1619 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let  
 1620 S be the set of status elements returned in the StatusResult. If any status element  
 1621 in S is an exception condition, then stop execution and return the same StatusResult  
 1622 element in the ReturnRegistryEntryResult.

- 1623 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the  
1624 ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*.  
1625 Add this warning to the StatusResult returned by the RegistryEntryQuery and return  
1626 this enhanced StatusResult with the ReturnRegistryEntryResult.
- 1627 4. For each registry entry E referenced by an element of R, use the attributes of E to  
1628 create a new RegistryEntry element as defined in Appendix A.2. Then create a new  
1629 RegistryEntryMetadata element as defined above to be the parent element of that  
1630 RegistryEntry element.
- 1631 5. If no With option is specified, then the resulting RegistryEntryMetadata element has  
1632 no Classification, SourceAssociations, TargetAssociations, AuditableEvent, or  
1633 ExternalData subelements. The set of RegistryEntryMetadata elements, with the  
1634 StatusResult from the RegistryEntryQuery, is returned as the  
1635 ReturnRegistryEntryResult.
- 1636 6. If WithClassifications is specified, then for each E in R do the following: If a  
1637 ClassificationFilter is not present, then let C be any classification instance linked to  
1638 E; otherwise, let C be a classification instance linked to E that satisfies the  
1639 ClassificationFilter (Section 8.2.9). For each such C, create a new Classification  
1640 element as defined in Appendix A.2. Add these Classification elements to their  
1641 parent RegistryEntryMetadata element.
- 1642 7. If WithSourceAssociations is specified, then for each E in R do the following: If an  
1643 AssociationFilter is not present, then let A be any association instance whose source  
1644 object is E; otherwise, let A be an association instance that satisfies the  
1645 AssociationFilter (Section 8.2.9) and whose source object is E. For each such A,  
1646 create a new Association element as defined in Appendix A.2. Add these  
1647 Association elements as subelements of the WithSourceAssociations and add that  
1648 element to its parent RegistryEntryMetadata element.
- 1649 8. If WithTargetAssociations is specified, then for each E in R do the following: If an  
1650 AssociationFilter is not present, then let A be any association instance whose target  
1651 object is E; otherwise, let A be an association instance that satisfies the  
1652 AssociationFilter (Section 8.2.9) and whose target object is E. For each such A,  
1653 create a new Association element as defined in Appendix A.2. Add these  
1654 Association elements as subelements of the WithTargetAssociations and add that  
1655 element to its parent RegistryEntryMetadata element.
- 1656 9. If WithAuditableEvents is specified, then for each E in R do the following: If an  
1657 AuditableEventFilter is not present, then let A be any auditable event instance linked  
1658 to E; otherwise, let A be any auditable event instance linked to E that satisfies the  
1659 AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent  
1660 element as defined in Appendix A.2. Add these AuditableEvent elements to their  
1661 parent RegistryEntryMetadata element.

- 1662 10. If WithExternalLinks is specified, then for each E in R do the following: If an  
 1663 ExternalLinkFilter is not present, then let L be any external link instance linked to E;  
 1664 otherwise, let L be any external link instance linked to E that satisfies the  
 1665 ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink  
 1666 element as defined in Appendix A.2. Add these ExternalLink elements to their parent  
 1667 RegistryEntryMetadata element.
- 1668 11. If any warning or exception condition results, then add the code and the message to  
 1669 the StatusResult that came from the RegistryEntryQuery result.
- 1670 12. Return the set of RegistryEntryMetadata elements and the revised StatusResult as  
 1671 the content of the ReturnRegistryEntryResult.

1672

### 1673 Examples

1674 A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by  
 1675 XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer wishes to  
 1676 check on the current status of that DTD, especially if it has been superseded or  
 1677 replaced, and get all of its current classifications. The following query request will return  
 1678 an XML document with the registry entry for the existing DTD as the root, with all of its  
 1679 classifications, and with associations to registry entries for any items that have  
 1680 superseded or replaced it.

```

1681
1682 <ReturnRegistryEntry>
1683   <RegistryEntryQuery>
1684     <RegistryEntryFilter>
1685       id EQUAL "urn:com:xyz:po:325"           -- code by Clause, Section 8.2.10
1686     </RegistryEntryFilter>
1687   </RegistryEntryQuery>
1688   <WithClassifications/>
1689   <WithSourceAssociations>
1690     <AssociationFilter>                       -- code by Clause, Section 8.2.10
1691       associationType EQUAL "SupersededBy" OR
1692       associationType EQUAL "ReplacedBy"
1693     </AssociationFilter>
1694   </WithSourceAssociations>
1695 </ReturnRegistryEntry>

```

1696

1697 A client of the Registry registered an XML DTD several years ago and is now thinking of  
 1698 replacing it with a revised version. The identifier for the existing DTD is  
 1699 "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the  
 1700 existing DTD. The client desires a list of all registered items that use the existing DTD  
 1701 so they can assess the impact of an incompatible change. The following query returns  
 1702 an XML document that is a list of all RegistryEntry elements that represent registered  
 1703 items that use, contain, or extend the given DTD. The document also links each  
 1704 RegistryEntry element in the list to an element for the identified association.

```

1705 <ReturnRegistryEntry>
1706 <RegistryEntryQuery>

```

```

1707     <SourceAssociationBranch>
1708         <AssociationFilter>                                -- code by Clause, Section 8.2.10
1709             associationType EQUAL "Contains" OR
1710             associationType EQUAL "Uses" OR
1711             associationType EQUAL "Extends"
1712         </AssociationFilter>
1713     <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1714         id EQUAL "urn:xyz:dtd:po97"
1715     </RegistryEntryFilter>
1716 </SourceAssociationBranch>
1717 </RegistryEntryQuery>
1718 <WithSourceAssociations>
1719     <AssociationFilter>                                -- code by Clause, Section 8.2.10
1720         associationType EQUAL "Contains" OR
1721         associationType EQUAL "Uses" OR
1722         associationType EQUAL "Extends"
1723     </AssociationFilter>
1724 </WithSourceAssociations>
1725 </ReturnRegistryEntry>

```

1726

1727 A user has been browsing the registry and has found a registry entry that describes a  
 1728 package of core-components that should solve the user's problem. The package URN  
 1729 identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to know what's in the package.  
 1730 The following query returns an XML document with a registry entry for each member of  
 1731 the package along with that member's Uses and HasMemberBranch associations.

```

1732 <ReturnRegistryEntry>
1733     <RegistryEntryQuery>
1734         <TargetAssociationBranch>
1735             <AssociationFilter>                                -- code by Clause, Section 8.2.10
1736                 associationType EQUAL "HasMember"
1737             </AssociationFilter>
1738             <RegistryEntryFilter>                            -- code by Clause, Section 8.2.10
1739                 id EQUAL " urn:com:cc:pkg:ccstuff "
1740             </RegistryEntryFilter>
1741         </TargetAssociationBranch>
1742     </RegistryEntryQuery>
1743     <WithSourceAssociations>
1744         <AssociationFilter>                                -- code by Clause, Section 8.2.10
1745             associationType EQUAL "HasMember" OR
1746             associationType EQUAL "Uses"
1747         </AssociationFilter>
1748     </WithSourceAssociations>
1749 </ReturnRegistryEntry>
1750
1751

```



1751 **8.2.8 ReturnRepositoryItem**1752 **Purpose**

1753 To construct an XML document that contains one or more repository items, and some  
 1754 associated metadata, by submitting a RegistryEntryQuery to the registry/repository that  
 1755 holds the desired objects. NOTE: Initially, the RegistryEntryQuery could be the URN  
 1756 identifier for a single registry entry.

1757 **Definition**

```

1758
1759 <!ELEMENT ReturnRepositoryItem
1760 ( RegistryEntryQuery,
1761   RecursiveAssociationOption?,
1762   WithDescription? )>
1763
1764 <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1765 <!ATTLIST RecursiveAssociationOption
1766   depthLimit CDATA #IMPLIED >
1767
1768 <!ELEMENT AssociationType EMPTY >
1769 <!ATTLIST AssociationType
1770   role CDATA #REQUIRED >
1771
1772 <!ELEMENT WithDescription EMPTY >
1773
1774 <!ELEMENT ReturnRepositoryItemResult
1775 ( RepositoryItem*, StatusResult )>
1776
1777 <!ELEMENT RepositoryItem
1778 ( ClassificationScheme
1779   | RegistryPackage
1780   | ExtrinsicObject
1781   | WithdrawnObject
1782   | ExternalLinkItem )>
1783 <!ATTLIST RepositoryItem
1784   identifier CDATA #REQUIRED
1785   name CDATA #REQUIRED
1786   contentURI CDATA #REQUIRED
1787   objectType CDATA #REQUIRED
1788   status CDATA #REQUIRED
1789   stability CDATA #REQUIRED
1790   description CDATA #IMPLIED >
1791
1792 <!ELEMENT ExtrinsicObject (#PCDATA) >
1793 <!ATTLIST ExtrinsicObject
1794   byteEncoding CDATA "Base64" >
1795
1796 <!ELEMENT WithdrawnObject EMPTY >
1797
1798 <!ELEMENT ExternalLinkItem EMPTY >
1799
1800
1801
```

1802 **Semantic Rules**

- 1803 1. If the RecursiveOption element is not present , then set Limit=0. If the  
1804 RecursiveOption element is present, interpret its depthLimit attribute as an integer  
1805 literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0  
1806 means that no recursion occurs. A Limit of -1 means that recursion occurs  
1807 indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive  
1808 integer, then stop execution and raise the exception: *invalid depth limit*; otherwise,  
1809 set Limit=N, where N is that positive integer. A Limit of N means that exactly N  
1810 recursive steps will be executed unless the process terminates prior to that limit.
- 1811 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned  
1812 as part of the ReturnRepositoryItemResult. Initially Result is empty. Semantic rules  
1813 4 through 10 determine the content of Result.
- 1814 3. If the WithDescription element is present, then set WSD="yes"; otherwise, set  
1815 WSD="no".
- 1816 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in  
1817 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let  
1818 S be the set of status elements returned in the StatusResult. If any status element  
1819 in S is an exception condition, then stop execution and return the same StatusResult  
1820 element in the ReturnRepositoryItemResult.
- 1821 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from  
1822 R. After execution of these rules, if Depth is now equal to Limit, then return the  
1823 content of Result as the set of RepositoryItem elements in the  
1824 ReturnRepositoryItemResult element; otherwise, continue with Semantic Rule 8.
- 1825 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the  
1826 following:
- 1827 a) If E.contentURI references a repository item in this registry/repository, then  
1828 create a new RepositoryItem element, with values for its attributes derived as  
1829 specified in Semantic Rule 7.
- 1830 1) If E.objectType="ClassificationScheme", then put the referenced  
1831 ClassificationScheme DTD as the subelement of this RepositoryItem.  
1832 [NOTE: Requires DTD specification!]
- 1833 2) If E.objectType="RegistryPackage", then put the referenced  
1834 RegistryPackage DTD as the subelement of this RepositoryItem. [NOTE:  
1835 Requires DTD specification!]
- 1836 3) Otherwise, i.e., if the object referenced by E has an unknown internal  
1837 structure, then put the content of the repository item as the #PCDATA of a  
1838 new ExtrinsicObject subelement of this RepositoryItem.
- 1839 b) If E.objectURL references a registered object in some other registry/repository,  
1840 then create a new RepositoryItem element, with values for its attributes derived  
1841 as specified in Semantic Rule 7, and create a new ExternalLink element as the  
1842 subelement of this RepositoryItem.

- 1843 c) If E.objectURL is void, i.e. the object it would have referenced has been  
1844 withdrawn, then create a new RepositoryItem element, with values for its  
1845 attributes derived as specified in Semantic Rule 7, and create a new  
1846 WithdrawnObject element as the subelement of this RepositoryItem.
- 1847 7. Let E be a registry entry and let RO be the RepositoryItem element created in  
1848 Semantic Rule 6. Set the attributes of RO to the values derived from the  
1849 corresponding attributes of E. If WSD="yes", include the value of the description  
1850 attribute; otherwise, do not include it. Insert this new RepositoryItem element into the  
1851 Result set.
- 1852 8. Let R be defined as in Semantic Rule 4. Execute Semantic Rule 9 with Y as the set  
1853 of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.
- 1854 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty  
1855 set of RegistryEntry instances. For each registry entry E in Y, and for each  
1856 AssociationType A of the RecursiveAssociationOption, do the following:
- 1857 a) Let Z be the set of target items E' linked to E under association instances having  
1858 E as the source object, E' as the target object, and A as the AssociationType.
- 1859 b) Add the elements of Z to NextLevel.
- 1860 10. Let X be the set of new registry entries that are in NextLevel but are not yet  
1861 represented in the Result set.
- 1862 Case:
- 1863 a) If X is empty, then return the content of Result as the set of RepositoryItem  
1864 elements in the ReturnRepositoryItemResult element.
- 1865 b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set.  
1866 When finished, add the elements of X to Y and set Depth=Depth+1. If Depth is  
1867 now equal to Limit, then return the content of Result as the set of RepositoryItem  
1868 elements in the ReturnRepositoryItemResult element; otherwise, repeat  
1869 Semantic Rules 9 and 10 with the new set Y of registry entries.
- 1870 11. If any exception, warning, or other status condition results during the execution of  
1871 the above, then return appropriate status elements as the StatusResult of the  
1872 ReturnRepositoryItemResult element created in Semantic Rule 5 or Semantic Rule  
1873 10.

1874

### 1875 **Examples**

1876 A registry client has found a registry entry for a core-component item. The item's URN  
1877 identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many  
1878 other registered items. The client desires the collection of all items needed for a  
1879 complete implementation of "goodthing". The following query returns an XML document  
1880 that is a collection of all needed items.

1881

```
1882     <ReturnRepositoryItem>
1883       <RegistryEntryQuery>
1884         <RegistryEntryFilter>           -- code by Clause, Section 8.2.10
1885           id EQUAL "urn:ebxml:cc:goodthing"
1886         </RegistryEntryFilter>
1887       </RegistryEntryQuery>
1888       <RecursiveAssociationOption>
1889         <AssociationType role="Uses" />
1890         <AssociationType role="ValidatesTo" />
1891       </RecursiveAssociationOption>
1892     </ReturnRepositoryItem>
```

1893

1894 A registry client has found a reference to a core-component routine  
1895 ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client knows  
1896 that all routines have a required association to its defining UML specification. The  
1897 following query returns both the routine and its UML specification as a collection of two  
1898 items in a single XML document.

```
1899
1900     <ReturnRepositoryItem>
1901       <RegistryEntryQuery>
1902         <RegistryEntryFilter>           -- code by Clause, Section 8.2.10
1903           id EQUAL "urn:ebxml:cc:rtn:nice87"
1904         </RegistryEntryFilter>
1905       </RegistryEntryQuery>
1906       <RecursiveAssociationOption depthLimit="1" >
1907         <AssociationType role="ValidatesTo" />
1908       </RecursiveAssociationOption>
1909     </ReturnRepositoryItem>
```

1910

1911 A user has been told that the 1997 version of the North American Industry Classification  
1912 System (NAICS) is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The  
1913 following query would retrieve the complete classification scheme, with all 1810 nodes,  
1914 as an XML document that validates to a classification scheme DTD.

```
1915
1916     <ReturnRepositoryItem>
1917       <RegistryEntryQuery>
1918         <RegistryEntryFilter>           -- code by Clause, Section 8.2.10
1919           id EQUAL "urn:nist:cs:naics-1997"
1920         </RegistryEntryFilter>
1921       </RegistryEntryQuery>
1922     </ReturnRepositoryItem>
```

1923

1924 Note: The ReturnRepositoryItemResult would include a single RepositoryItem that  
1925 consists of a ClassificationScheme document whose content is determined by the URL  
1926 <ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt>.

1927

## 1927 8.2.9 Registry Filters

### 1928 Purpose

1929 To identify a subset of the set of all persistent instances of a given registry class.

### 1930 Definition

1931  
1932 <!ELEMENT ObjectFilter ( Clause )>  
1933  
1934 <!ELEMENT RegistryEntryFilter ( Clause )>  
1935  
1936 <!ELEMENT IntrinsicObjectFilter ( Clause )>  
1937  
1938 <!ELEMENT ExtrinsicObjectFilter ( Clause )>  
1939  
1940 <!ELEMENT PackageFilter ( Clause )>  
1941  
1942 <!ELEMENT OrganizationFilter ( Clause )>  
1943  
1944 <!ELEMENT ContactFilter ( Clause )>  
1945  
1946 <!ELEMENT ClassificationNodeFilter ( Clause )>  
1947  
1948 <!ELEMENT AssociationFilter ( Clause )>  
1949  
1950 <!ELEMENT ClassificationFilter ( Clause )>  
1951  
1952 <!ELEMENT ExternalLinkFilter ( Clause )>  
1953  
1954 <!ELEMENT AuditableEventFilter ( Clause )>  
1955  
1956 <!ELEMENT UserFilter ( Clause )>

1957

### 1958 Semantic Rules

- 1959 1. The Clause element is defined in Section 8.2.10, Clause.
- 1960 2. For every ObjectFilter XML element, the leftArgument attribute of any containing  
1961 SimpleClause shall identify a public attribute of the RegistryObject UML class  
1962 defined in [ebRIM]. If not, raise exception: *object attribute error*. The ObjectFilter  
1963 returns a set of identifiers for RegistryObject instances whose attribute values  
1964 evaluate to *True* for the Clause predicate.
- 1965 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any  
1966 containing SimpleClause shall identify a public attribute of the RegistryEntry UML  
1967 class defined in [ebRIM].  
1968 If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a  
1969 set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*  
1970 for the Clause predicate.

- 1971 4. For every IntrinsicObjectFilter XML element, the leftArgument attribute of any  
1972 containing SimpleClause shall identify a public attribute of the IntrinsicObject UML  
1973 class defined in [ebRIM]. If not, raise exception: *intrinsic object attribute error*. The  
1974 IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject instances whose  
1975 attribute values evaluate to *True* for the Clause predicate.
- 1976 5. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any  
1977 containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML  
1978 class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The  
1979 ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose  
1980 attribute values evaluate to *True* for the Clause predicate.
- 1981 6. For every PackageFilter XML element, the leftArgument attribute of any containing  
1982 SimpleClause shall identify a public attribute of the Package UML class defined in  
1983 [ebRIM]. If not, raise exception: *package attribute error*. The PackageFilter returns a  
1984 set of identifiers for Package instances whose attribute values evaluate to *True* for  
1985 the Clause predicate.
- 1986 7. For every OrganizationFilter XML element, the leftArgument attribute of any  
1987 containing SimpleClause shall identify a public attribute of the Organization or  
1988 PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization*  
1989 *attribute error*. The OrganizationFilter returns a set of identifiers for Organization  
1990 instances whose attribute values evaluate to *True* for the Clause predicate.
- 1991 8. For every ContactFilter XML element, the leftArgument attribute of any containing  
1992 SimpleClause shall identify a public attribute of the Contact or PostalAddress UML  
1993 class defined in [ebRIM]. If not, raise exception: *contact attribute error*. The  
1994 ContactFilter returns a set of identifiers for Contact instances whose attribute values  
1995 evaluate to *True* for the Clause predicate.
- 1996 9. For every ClassificationNodeFilter XML element, the leftArgument attribute of any  
1997 containing SimpleClause shall identify a public attribute of the ClassificationNode  
1998 UML class defined in [ebRIM]. If not, raise exception: *classification node attribute*  
1999 *error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode  
2000 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2001 10. For every AssociationFilter XML element, the leftArgument attribute of any  
2002 containing SimpleClause shall identify a public attribute of the Association UML  
2003 class defined in [ebRIM]. If not, raise exception: *association attribute error*. The  
2004 AssociationFilter returns a set of identifiers for Association instances whose attribute  
2005 values evaluate to *True* for the Clause predicate.
- 2006 11. For every ClassificationFilter XML element, the leftArgument attribute of any  
2007 containing SimpleClause shall identify a public attribute of the Classification UML  
2008 class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The  
2009 ClassificationFilter returns a set of identifiers for Classification instances whose  
2010 attribute values evaluate to *True* for the Clause predicate.

- 2011 12. For every ExternalLinkFilter XML element, the leftArgument attribute of any  
2012 containing SimpleClause shall identify a public attribute of the ExternalLink UML  
2013 class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The  
2014 ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose  
2015 attribute values evaluate to *True* for the Clause predicate.
- 2016 13. For every AuditableEventFilter XML element, the leftArgument attribute of any  
2017 containing SimpleClause shall identify a public attribute of the AuditableEvent UML  
2018 class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The  
2019 AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose  
2020 attribute values evaluate to *True* for the Clause predicate.
- 2021 14. For every UserFilter XML element, the leftArgument attribute of any containing  
2022 SimpleClause shall identify a public attribute of the User UML class defined in  
2023 [ebRIM]. If not, raise exception: *auditable identity attribute error*. The UserFilter  
2024 returns a set of identifiers for User instances whose attribute values evaluate to *True*  
2025 for the Clause predicate.

2026

### 2027 Example

2028 The following is a complete example of RegistryEntryQuery combined with Clause  
2029 expansion of RegistryEntryFilter to return a set of RegistryEntry instances whose  
2030 objectType attribute is "CPP" and whose status attribute is "Approved".

```
2031 <RegistryEntryQuery>
2032   <RegistryEntryFilter>
2033     <Clause>
2034       <CompoundClause   connectivePredicate="And" >
2035         <Clause>
2036           <SimpleClause leftArgument="objectType" >
2037             <StringClause stringPredicate="equal" >CPP</StringClause>
2038           </SimpleClause>
2039         </Clause>
2040       </CompoundClause>
2041     </Clause>
2042     <SimpleClause leftArgument="status" >
2043       <StringClause stringPredicate="equal" >Approved</StringClause>
2044     </SimpleClause>
2045   </RegistryEntryFilter>
2046 </RegistryEntryQuery>
```

2051

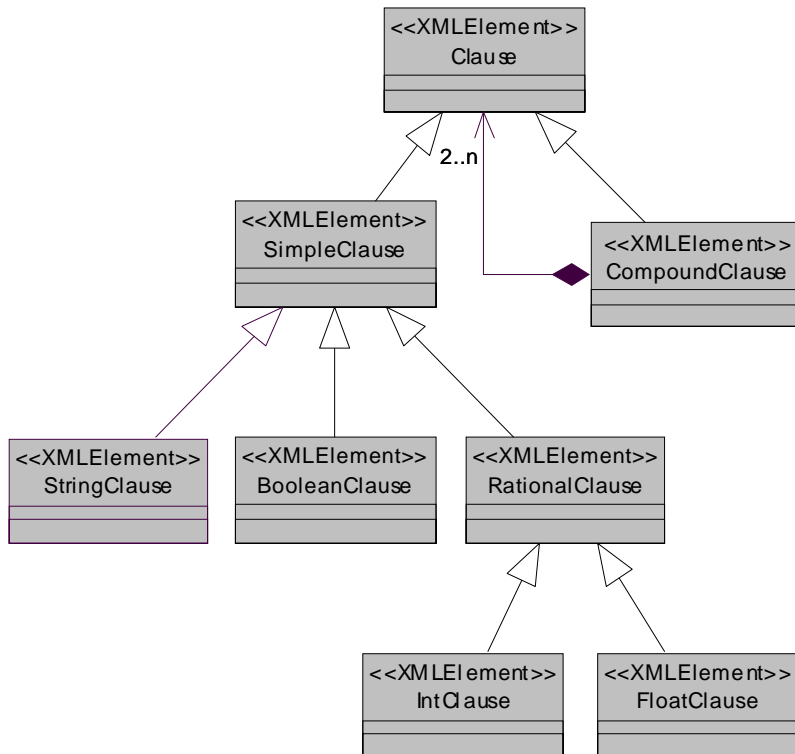
2051 **8.2.10 XML Clause Constraint Representation**

2052 **Purpose**

2053 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate*  
 2054 *Clauses*. Predicate Clauses are utilized to formally define the constraint mechanism,  
 2055 and are referred to simply as **Clauses** in this specification.

2056 **Conceptual UML Diagram**

2057 The following is a conceptual diagram outlining the Clause base structure. It is  
 2058 expressed in UML for visual depiction.



2059

2060 **Semantic Rules**

2061 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate -  
 2062 RightArgument" format to form a *Clause*. There are two types of Clauses:  
 2063 *SimpleClauses* and *CompoundClauses*.

2064 SimpleClauses

2065 A SimpleClause always defines the leftArgument as a text string, sometimes referred to  
 2066 as the *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be  
 2067 extended. SimpleClause is extended to support BooleanClause, StringClause, and  
 2068 RationalClause (abstract).



2069 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a  
 2070 boolean. StringClause defines the predicate as an enumerated attribute of appropriate  
 2071 string-compare operations and a right argument as the element's text data. Rational  
 2072 number support is provided through a common RationalClause providing an  
 2073 enumeration of appropriate rational number compare operations, which is further  
 2074 extended to IntClause and FloatClause, each with appropriate signatures for the right  
 2075 argument.

#### 2076 CompoundClauses

2077 A CompoundClause contains two or more Clauses (Simple or Compound) and a  
 2078 connective predicate. This provides for arbitrarily complex Clauses to be formed.

2079

#### 2080 **Definition**

```

2081
2082     <!ELEMENT Clause ( SimpleClause | CompoundClause )>
2083
2084     <!ELEMENT SimpleClause
2085       ( BooleanClause | RationalClause | StringClause )>
2086     <!ATTLIST SimpleClause
2087       leftArgument CDATA #REQUIRED >
2088
2089     <!ELEMENT CompoundClause ( Clause, Clause+ )>
2090     <!ATTLIST CompoundClause
2091       connectivePredicate ( And | Or ) #REQUIRED>
2092
2093     <!ELEMENT BooleanClause EMPTY >
2094     <!ATTLIST BooleanClause
2095       booleanPredicate ( True | False ) #REQUIRED>
2096
2097     <!ELEMENT RationalClause ( IntClause | FloatClause )>
2098     <!ATTLIST RationalClause
2099       logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
2100
2101     <!ELEMENT IntClause ( #PCDATA )
2102     <!ATTLIST IntClause
2103       e-dtype NMTOKEN #FIXED 'int' >
2104
2105     <!ELEMENT FloatClause ( #PCDATA )>
2106     <!ATTLIST FloatClause
2107       e-dtype NMTOKEN #FIXED 'float' >
2108
2109     <!ELEMENT StringClause ( #PCDATA )>
2110     <!ATTLIST StringClause
2111       stringPredicate
2112         ( contains | -contains |
2113           startswith | -startswith |
2114           equal | -equal
2115           endswith | -endswith ) #REQUIRED >
2116
  
```

2117 **Examples**2118 **Simple BooleanClause: "Smoker" = True**

```
2119
2120     <?xml version="1.0" encoding="UTF-8"?>
2121     <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2122     <Clause>
2123         <SimpleClause leftArgument="Smoker">
2124             <BooleanClause booleanPredicate="True"/>
2125         </SimpleClause>
2126     </Clause>
2127
```

2128 **Simple StringClause: "Smoker" contains "mo"**

```
2129
2130     <?xml version="1.0" encoding="UTF-8"?>
2131     <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2132     <Clause>
2133         <SimpleClause leftArgument="Smoker">
2134             <StringClause stringcomparepredicate="contains">
2135                 mo
2136             </StringClause>
2137         </SimpleClause>
2138     </Clause>
2139
```

2139

2140 **Simple IntClause: "Age" >= 7**

```
2141
2142     <?xml version="1.0" encoding="UTF-8"?>
2143     <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2144     <Clause>
2145         <SimpleClause leftArgument="Age">
2146             <RationalClause logicalPredicate="GE">
2147                 <IntClause e-dtype="int">7</IntClause>
2148             </RationalClause>
2149         </SimpleClause>
2150     </Clause>
2151
```

2152 **Simple FloatClause: "Size" = 4.3**

```
2153
2154     <?xml version="1.0" encoding="UTF-8"?>
2155     <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2156     <Clause>
2157         <SimpleClause leftArgument="Size">
2158             <RationalClause logicalPredicate="E">
2159                 <FloatClause e-dtype="float">4.3</FloatClause>
2160             </RationalClause>
2161         </SimpleClause>
2162     </Clause>
2163
```

2163

## 2164 Compound with two Simples (("Smoker" = False)AND("Age" =&lt; 45))

```
2165
2166 <?xml version="1.0" encoding="UTF-8"?>
2167 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2168 <Clause>
2169   <CompoundClause connectivePredicate="And">
2170     <Clause>
2171       <SimpleClause leftArgument="Smoker">
2172         <BooleanClause booleanPredicate="False"/>
2173       </SimpleClause>
2174     </Clause>
2175     <Clause>
2176       <SimpleClause leftArgument="Age">
2177         <RationalClause logicalPredicate="EL">
2178           <IntClause e-dtype="int">45</IntClause>
2179         </RationalClause>
2180       </SimpleClause>
2181     </Clause>
2182   </CompoundClause>
2183 </Clause>
```

2184

## 2185 Coumpound with one Simple and one Compound

2186 ( ("Smoker" = False)And(("Age" =&lt; 45)Or("American"=True)) )

```
2187
2188 <?xml version="1.0" encoding="UTF-8"?>
2189 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2190 <Clause>
2191   <CompoundClause connectivePredicate="And">
2192     <Clause>
2193       <SimpleClause leftArgument="Smoker">
2194         <BooleanClause booleanPredicate="False"/>
2195       </SimpleClause>
2196     </Clause>
2197     <Clause>
2198       <CompoundClause connectivePredicate="Or">
2199         <Clause>
2200           <SimpleClause leftArgument="Age">
2201             <RationalClause logicalPredicate="EL">
2202               <IntClause e-dtype="int">45</IntClause>
2203             </RationalClause>
2204           </SimpleClause>
2205         </Clause>
2206         <Clause>
2207           <SimpleClause leftArgument="American">
2208             <BooleanClause booleanPredicate="True"/>
2209           </SimpleClause>
2210         </Clause>
2211       </CompoundClause>
2212     </Clause>
2213   </CompoundClause>
2214 </Clause>
```

## 2215 8.3 SQL Query Support

2216 The Registry may optionally support an SQL based query capability that is designed for  
2217 Registry clients that demand more complex query capability. The optional SQLQuery  
2218 element in the AdhocQueryRequest allows a client to submit complex SQL queries  
2219 using a declarative query language.

2220 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper  
2221 subset of the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992,  
2222 Database Language SQL [SQL], extended to include `<sql invoked routines>`  
2223 (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-  
2224 defined routines defined in template form in Appendix C.3. The exact syntax of the  
2225 Registry query language is defined by the BNF grammar in C.1.

2226 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement  
2227 to use relational databases in a Registry implementation.

### 2228 8.3.1 SQL Query Syntax Binding To [ebRIM]

2229 SQL Queries are defined based upon the query syntax in in Appendix C.1 and a fixed  
2230 relational schema defined in Appendix C.3. The relational schema is an algorithmic  
2231 binding to [ebRIM] as described in the following sections.

#### 2232 8.3.1.1 Interface and Class Binding

2233 A subset of the Interface and class names defined in [ebRIM] map to table names that  
2234 may be queried by an SQL query. Appendix C.3 defines the names of the ebRIM  
2235 interfaces and classes that may be queried by an SQL query.

2236 The algorithm used to define the binding of [ebRIM] classes to table definitions in  
2237 Appendix C.3 is as follows:

- 2238 • Only those classes and interfaces that have concrete instances are mapped to  
2239 relational tables. This results in intermediate interfaces in the inheritance  
2240 hierarchy, such as RegistryObject and IntrinsicObject, to not map to SQL tables.  
2241 An exception to this rule is RegistryEntry, which is defined next.
- 2242 • A special view called RegistryEntry is defined to allow SQL queries to be made  
2243 against RegistryEntry instances. This is the only interface defined in [ebRIM] that  
2244 does not have concrete instances but is queryable by SQL queries.
- 2245 • The names of relational tables are the same as the corresponding [ebRIM] class  
2246 or interface name. However, the name binding is case insensitive.
- 2247 • Each [ebRIM] class or interface that maps to a table in Appendix C.3 includes  
2248 column definitions in Appendix C.3 where the column definitions are based on a  
2249 subset of attributes defined for that class or interface in [ebRIM]. The attributes  
2250 that map to columns include the inherited attributes for the [ebRIM] class or  
2251 interface. Comments in Appendix C.3 indicate which ancestor class or interface  
2252 contributed which column definitions.

2253 An SQLQuery against a table not defined in Appendix C.3 may result in an ebXMLError  
2254 message with an InvalidQueryException.

2255 The following sections describe the algorithm for mapping attributes of [ebRIM] to  
2256 SQLcolumn definitions.

#### 2257 **8.3.1.2 Accessor Method To Attribute Binding**

2258 Most of the [ebRIM] interfaces methods are simple get methods that map directly to  
2259 attributes. For example the getName method on RegistryObject maps to a name  
2260 attribute of type String. Each get method in [ebRIM] defines the exact attribute name  
2261 that it maps to in the interface definitions in [ebRIM].

#### 2262 **8.3.1.3 Primitive Attributes Binding**

2263 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the  
2264 same way as column names in SQL. Again the exact attribute names are defined in the  
2265 interface definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is  
2266 case insensitive. It is therefore valid for a query to contain attribute names that do not  
2267 exactly match the case defined in [ebRIM].

#### 2268 **8.3.1.4 Reference Attribute Binding**

2269 A few of the [ebRIM] interface methods return references to instances of interfaces or  
2270 classes defined by [ebRIM]. For example, the getAccessControlPolicy method of the  
2271 RegistryObject class returns a reference to an instance of an AccessControlPolicy  
2272 object.

2273 In such cases the reference maps to the id attribute for the referenced object. The  
2274 name of the resulting column is the same as the attribute name in [ebRIM] as defined by  
2275 8.3.1.3. The data type for the column is UUID as defined in Appendix C.3.

2276 When a reference attribute value holds a null reference, it maps to a null value in the  
2277 SQL binding and may be tested with the <null specification> as defined by [SQL].

2278 Reference attribute binding is a special case of a primitive attribute mapping.

#### 2279 **8.3.1.5 Complex Attribute Binding**

2280 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead  
2281 they are of a complex type as defined by an entity class in [ebRIM]. Examples include  
2282 attributes of type TelephoneNumber, Contact, PersonName etc. in interface  
2283 Organization and class Contact.

2284 The SQL query schema algorithmically maps such complex attributes as multiple  
2285 primitive attributes within the parent table. The mapping simply flattens out the entity  
2286 class attributes within the parent table. The attribute name for the flattened attributes  
2287 are composed of a concatenation of attribute names in the reference chain. For example  
2288 Organization has a contact attribute of type Contact. Contact has an address attribute of  
2289 type PostalAddress. PostalAddress has a String attribute named city. This city attribute  
2290 will be named contact\_address\_city.

### 2291 **8.3.1.6 Collection Attribute Binding**

2292 A few of the [ebRIM] interface methods return a collection of references to instances of  
2293 interfaces or classes defined by [ebRIM]. For example, the getPackages method of the  
2294 ManagedObject class returns a Collection of references to instances of Packages that  
2295 the object is a member of.

2296 Such collection attributes in [ebRIM] classes have been mapped to stored procedures in  
2297 Appendix C.3 such that these stored procedures return a collection of `id` attribute  
2298 values. The returned value of these stored procedures can be treated as the result of a  
2299 table sub-query in SQL.

2300 These stored procedures may be used as the right-hand-side of an SQL IN clause to  
2301 test for membership of an object in such collections of references.

### 2302 **8.3.2 Semantic Constraints On Query Syntax**

2303 This section defines simplifying constraints on the query syntax that cannot be  
2304 expressed in the BNF for the query syntax. These constraints must be applied in the  
2305 semantic analysis of the query.

- 2306 1. Class names and attribute names must be processed in a case insensitive manner.
- 2307 2. The syntax used for stored procedure invocation must be consistent with the syntax  
2308 of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 2309 3. For this version of the specification, the SQL select column list consists of exactly  
2310 one column, and must always be `t.id`, where `t` is a table reference in the FROM  
2311 clause.

### 2312 **8.3.3 SQL Query Results**

2313 The results of an SQL query is always an ObjectRefList as defined by the  
2314 AdHocQueryResponse in 8.4. This means the result of an SQL query is always a  
2315 collection of references to instances of a sub-class of the RegistryObject interface in  
2316 [ebRIM]. This is reflected in a semantic constraint that requires that the SQL select  
2317 column specified must always be an `id` column in a table in Appendix C.3 for this  
2318 version of the specification.

### 2319 **8.3.4 Simple Metadata Based Queries**

2320 The simplest form of an SQL query is based upon metadata attributes specified for a  
2321 single class within [ebRIM]. This section gives some examples of simple metadata  
2322 based queries.

2323 For example, to get the collection of ExtrinsicObjects whose name contains the word  
2324 'Acme' and that have a version greater than 1.3, the following query predicates must be  
2325 supported:

2326  
2327  
2328  
2329

```
SELECT id FROM ExtrinsicObject WHERE name LIKE '%Acme%' AND  
majorVersion >= 1 AND  
(majorVersion >= 2 OR minorVersion > 3);
```

2330 Note that the query syntax allows for conjugation of simpler predicates into more  
2331 complex queries as shown in the simple example above.

### 2332 **8.3.5 RegistryEntry Queries**

2333 Given the central role played by the RegistryEntry interface in ebRIM, the schema for  
2334 the SQL query defines a special view called RegistryEntry that allows doing a  
2335 polymorphic query against all RegistryEntry instances regardless of their actual  
2336 concrete type or table name.

2337 The following example is the same as Section 8.3.4 except that it is applied against all  
2338 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will  
2339 include id for all qualifying RegistryEntry instances whose name contains the word  
2340 'Acme' and that have a version greater than 1.3.

```
2341 SELECT id FROM RegistryEntry WHERE name LIKE '%Acme%' AND  
2342 objectType = 'ExtrinsicObject' AND  
2343 majorVersion >= 1 AND  
2344 (majorVersion >= 2 OR minorVersion > 3);
```

### 2345 **8.3.6 Classification Queries**

2346 This section describes the various classification related queries that must be supported.

#### 2347 **8.3.6.1 Identifying ClassificationNodes**

2348 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they  
2349 may also be identified as a path attribute that specifies an XPATH expression [XPT]  
2350 from a root classification node to the specified classification node in the XML document  
2351 that would represent the ClassificationNode tree including the said ClassificationNode.

#### 2352 **8.3.6.2 Getting Root Classification Nodes**

2353 To get the collection of root ClassificationNodes the following query predicate must be  
2354 supported:

```
2355 SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

2356 The above query returns all ClassificationNodes that have their parent attribute set to  
2357 null. Note that the above query may also specify a predicate on the name if a specific  
2358 root ClassificationNode is desired.

#### 2359 **8.3.6.3 Getting Children of Specified ClassificationNode**

2360 To get the children of a ClassificationNode given the ID of that node the following style  
2361 of query must be supported:

```
2362 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

2363 The above query returns all ClassificationNodes that have the node specified by <id> as  
2364 their parent attribute.

2365 **8.3.6.4 Getting Objects Classified By a ClassificationNode**

2366 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the  
 2367 following style of query must be supported:

```
2368
2369 SELECT id FROM ExtrinsicObject
2370 WHERE
2371     id IN (SELECT classifiedObject FROM Classification
2372           WHERE
2373             classificationNode IN (SELECT id FROM ClassificationNode
2374                                   WHERE path = '/Geography/Asia/Japan'))
2375 AND
2376     id IN (SELECT classifiedObject FROM Classification
2377           WHERE
2378             classificationNode IN (SELECT id FROM ClassificationNode
2379                                   WHERE path = '/Industry/Automotive'))
```

2380 The above query gets the collection of ExtrinsicObjects that are classified by the  
 2381 Automotive Industry and the Japan Geography. Note that according to the semantics  
 2382 defined for GetClassifiedObjectsRequest, the query will also contain any objects that  
 2383 are classified by descendents of the specified ClassificationNodes.

2384 **8.3.6.5 Getting ClassificationNodes That Classify an Object**

2385 To get the collection of ClassificationNodes that classify a specified Object the following  
 2386 style of query must be supported:

```
2387 SELECT id FROM ClassificationNode
2388 WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

2389 **8.3.7 Association Queries**

2390 This section describes the various Association related queries that must be supported.

2391 **8.3.7.1 Getting All Association With Specified Object As Its Source**

2392 To get the collection of Associations that have the specified Object as its source, the  
 2393 following query must be supported:

```
2394 SELECT id FROM Association WHERE sourceObject = <id>
```

2395 **8.3.7.2 Getting All Association With Specified Object As Its Target**

2396 To get the collection of Associations that have the specified Object as its target, the  
 2397 following query must be supported:

```
2398 SELECT id FROM Association WHERE targetObject = <id>
```

2399 **8.3.7.3 Getting Associated Objects Based On Association Attributes**

2400 To get the collection of Associations that have specified Association attributes, the  
 2401 following queries must be supported:

2402 Select Associations that have the specified name.

```
2403 SELECT id FROM Association WHERE name = <name>
```

2404 Select Associations that have the specified source role name.

```
2405 SELECT id FROM Association WHERE sourceRole = <roleName>
```



2406 Select Associations that have the specified target role name.

```
2407 SELECT id FROM Association WHERE targetRole = <roleName>
```

2408 Select Associations that have the specified association type, where association type is a string containing the corresponding field name described in [ebRIM].

```
2409 SELECT id FROM Association WHERE
2410 associationType = <associationType>
```

### 2412 8.3.7.4 Complex Association Queries

2413 The various forms of Association queries may be combined into complex predicates.  
2414 The following query selects Associations from an object with a specified id, that have the sourceRole "buysFrom" and targetRole "sellsTo":

```
2416 SELECT id FROM Association WHERE
2417 sourceObject = <id> AND
2418 sourceRole = 'buysFrom' AND
2419 targetRole = 'sellsTo'
```

### 2420 8.3.8 Package Queries

2421 To find all Packages that a specified ExtrinsicObject belongs to, the following query is specified:

```
2422 SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

#### 2424 8.3.8.1 Complex Package Queries

2425 The following query gets all Packages that a specified object belongs to, that are not deprecated and where name contains "RosettaNet."

```
2427 SELECT id FROM Package WHERE
2428 id IN (RegistryEntry_packages(<id>)) AND
2429 name LIKE '%RosettaNet%' AND
2430 status <> 'Deprecated'
```

### 2431 8.3.9 ExternalLink Queries

2432 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is specified:

```
2433 SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

2435 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is specified:

```
2436 SELECT id From ExtrinsicObject WHERE id IN (RegistryEntry_linkedObjects(<id>))
```

#### 2438 8.3.9.1 Complex ExternalLink Queries

2439 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that contain the word 'legal' in their description and have a URL for their externalURI.

```
2441 SELECT id FROM ExternalLink WHERE
2442 id IN (RegistryEntry_externalLinks(<id>)) AND
2443 description LIKE '%legal%' AND
2444 externalURI LIKE '%http://%'
```

2445 **8.3.10 Audit Trail Queries**

2446 To get the complete collection of AuditableEvent objects for a specified ManagedObject,  
 2447 the following query is specified:

```
2448 SELECT id FROM AuditableEvent WHERE registryEntry = <id>
```

2449 **8.4 Ad Hoc Query Request/Response**

2450 A client submits an ad hoc query to the ObjectQueryManager by sending an  
 2451 AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a  
 2452 query in one of the supported Registry query mechanisms.

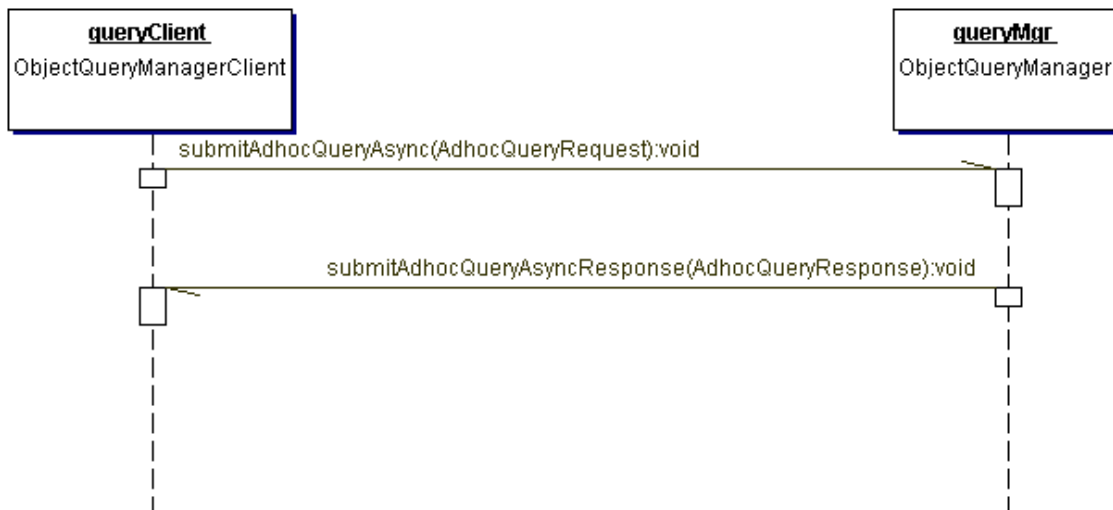
2453 The ObjectQueryManager sends an AdhocQueryResponse either synchronously or  
 2454 asynchronously back to the client. The AdhocQueryResponse return a collection of  
 2455 objects whose element type is in the set of element types represented by the leaf nodes  
 2456 of the RegistryEntry hierarchy in [ebRIM].



2457

2458

**Figure 16: Submit Ad Hoc Query Sequence Diagram**



2459

2460

**Figure 17: Submit Ad Hoc Query Asynchronous Sequence Diagram**

2461 For details on the schema for the business documents shown in this process refer to  
 2462 Appendix A.2.

2463 **8.5 Content Retrieval**

2464 A client retrieves content via the Registry by sending the GetContentRequest to the  
 2465 ObjectQueryManager. The GetContentRequest specifies a list of Object references for  
 2466 Objects that need to be retrieved. The ObjectQueryManager returns the specified  
 2467 content by sending a GetContentResponse message to the ObjectQueryManagerClient  
 2468 interface of the client. If there are no errors encountered, the GetContentResponse  
 2469 message includes the specified content as additional payloads within the message. In  
 2470 addition to the GetContentResponse payload, there is one additional payload for each  
 2471 content that was requested. If there are errors encountered, the GetContentResponse  
 2472 payload includes an ebXMLError and there are no additional content specific payloads.

2473 **8.5.1 Identification Of Content Payloads**

2474 Since the GetContentResponse message may include several repository items as  
 2475 additional payloads, it is necessary to have a way to identify each payload in the  
 2476 message. To facilitate this identification, the Registry must do the following:

- 2477 • Use the ID for each RegistryEntry instance that describes the repository item as  
 2478 the DocumentLabel element in the DocumentReference for that object in the  
 2479 Manifest element of the ebXMLHeader.

2480 **8.5.2 GetContentResponse Message Structure**

2481 The following message fragment illustrates the structure of the GetContentResponse  
 2482 Message that is returning a Collection of CPPs as a result of a GetContentRequest that  
 2483 specified the IDs for the requested objects. Note that the ID for each object retrieved in  
 2484 the message as additional payloads is used as its DocumentLabel in the Manifest of the  
 2485 ebXMLHeader.

```

2486 ...
2487 --7250537.978150567601.JavaMail.najmi.irian
2488 ...
2489 <ebXMLHeader MessageType="Normal" Version="1.0">
2490   <Manifest>
2491     <DocumentReference>
2492       <DocumentLabel>GetContentsResponse</DocumentLabel>
2493       <DocumentId>6835fb:e3be512ac8:-8000</DocumentId>
2494     </DocumentReference>
2495     <DocumentReference>
2496       <DocumentLabel> ID for CPP content #1 </DocumentLabel>
2497       <DocumentId>...</DocumentId>
2498     </DocumentReference>
2499     <DocumentReference>
2500       <DocumentLabel> ID for CPP content #2 </DocumentLabel>
2501       <DocumentId>... </DocumentId>
2502     </DocumentReference>
2503   </Manifest>
2504   <Header>
2505     ...
2506   </Header>
2507 --7250537.978150567601.JavaMail.najmi.irian
2508 Content-Type: application/xml
2509 Content-Description: GetContentsResponse
2510 Content-ID: 6835fb:e3be512ac8:-7ffc
2511 Content-Length: 97
2512
2513 <?xml version="1.0" encoding="UTF-8"?>
2514 <GetContentsResponse />
2515
2516 --7250537.978150567601.JavaMail.najmi.irian
2517 Content-Type: application/xml
2518 Content-Description: ID for CPP content #1
2519 Content-ID: ...
2520
2521 <CPP>
2522 ...
2523 </CPP>
2524
2525 --7250537.978150567601.JavaMail.najmi.irian
2526 Content-Type: application/xml
2527 Content-Description: ID for CPP content #2
2528 Content-ID: ...
2529
2530 <CPP>
2531 ...
2532 </CPP>
2533 --7250537.978150567601.JavaMail.najmi.irian--
    
```

2534

2535 **8.6 Query And Retrieval: Typical Sequence**

2536 The following diagram illustrates the use of both browse/drilldown and ad hoc queries  
 2537 followed by a retrieval of content that was selected by the queries.



2538

2539

Figure 20: Typical Query and Retrieval Sequence

2540

## 9 Registry Security

2541

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [ebRIM].

2542

2543

2544

In the current version of this specification, a minimalist approach has been specified for Registry security. The philosophy is that “Any *known* entity can publish content and *anyone* can view published content.” The Registry information model has been designed to allow more sophisticated security policies in future versions of this specification.

2545

2546

2547

2548

2549

### 9.1 Integrity of Registry Content

2550

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

2551

2552

2553

2554

2555

### 2556 **9.1.1 Message Payload Signature**

2557 Integrity of Registry content requires that all submitted content must be signed by the  
2558 Registry client as defined by [SEC]. The signature on the submitted content ensures  
2559 that:

- 2560 • The content has not been tampered with en-route or within the Registry.
- 2561 • The content's veracity can be ascertained by its association with a specific  
2562 submitting organization

## 2563 **9.2 Authentication**

2564 The Registry must be able to authenticate the identity of the Principal associated with  
2565 client requests. *Authentication* is required to identify the ownership of content as well as  
2566 to identify what "privileges" a Principal can be assigned with respect to the specific  
2567 objects in the Registry.

2568 The Registry must perform Authentication on a per request basis. From a security point  
2569 of view, all messages are independent and there is no concept of a session  
2570 encompassing multiple messages or conversations. Session support may be added as  
2571 an optimization feature in future versions of this specification.

2572 The Registry must implement a credential-based authentication mechanism based on  
2573 digital certificates and signatures. The Registry uses the certificate DN from the  
2574 signature to authenticate the user.

### 2575 **9.2.1 Message Header Signature**

2576 Message headers may be signed by the sending ebXML Messaging Service as defined  
2577 by [SEC]. Since this specification is not yet finalized, this version does not require that  
2578 the message header be signed. In the absence of a message header signature, the  
2579 payload signature is used to authenticate the identity of the requesting client.

## 2580 **9.3 Confidentiality**

### 2581 **9.3.1 On-the-wire Message Confidentiality**

2582 It is suggested but not required that message payloads exchanged between clients and  
2583 the Registry be encrypted during transmission. Payload encryption must abide by any  
2584 restrictions set forth in [SEC].

2585 **9.3.2 Confidentiality of Registry Content**

2586 In the current version of this specification, there are no provisions for confidentiality of  
 2587 Registry content. All content submitted to the Registry may be discovered and read by  
 2588 *any* client. Therefore, the Registry must be able to decrypt any submitted content after it  
 2589 has been received and prior to storing it in its repository. This implies that the Registry  
 2590 and the client have an a priori agreement regarding encryption algorithm, key exchange  
 2591 agreements, etc. This service is not addressed in this specification.

2592 **9.4 Authorization**

2593 The Registry must provide an authorization mechanism based on the information model  
 2594 defined in [ebRIM]. In this version of the specification the authorization mechanism is  
 2595 based on a default Access Control Policy defined for a pre-defined set of roles for  
 2596 Registry users. Future versions of this specification will allow for custom Access Control  
 2597 Policies to be defined by the Submitting Organization.

2598 **9.4.1 Pre-defined Roles For Registry Users**

2599 The following roles must be pre-defined in the Registry:

<b>Role</b>	<b>Description</b>
ContentOwner	The submitter or owner of a Registry content. Submitting Organization (SO) in ISO 11179
RegistryAdministrator	A "super" user that is an administrator of the Registry. Registration Authority (RA) in ISO 11179
RegistryGuest	Any unauthenticated user of the Registry. Clients that browse the Registry do not need to be authenticated.

2600 **9.4.2 Default Access Control Policies**

2601 The Registry must create a default AccessControlPolicy object that grants the default  
 2602 permissions to Registry users based upon their assigned role.

2603 The following table defines the Permissions granted by the Registry to the various pre-  
 2604 defined roles for Registry users based upon the default AccessControlPolicy.

2605

<b>Role</b>	<b>Permissions</b>
ContentOwner	Access to <i>all</i> methods on Registry Objects that are owned by the ContentOwner.

RegistryAdministrator	Access to <i>all</i> methods on <i>all</i> Registry Objects
RegistryGuest	Access to <i>all</i> read-only (getXXX) methods on <i>all</i> Registry Objects (read-only access to all content).

2606

2607 The following list summarizes the default role-based AccessControlPolicy:

- 2608 • The Registry must implement the default AccessControlPolicy and associate it  
2609 with all Objects in the Registry
- 2610 • Anyone can publish content, but needs to be authenticated
- 2611 • Anyone can access the content without requiring authentication
- 2612 • The ContentOwner has access to all methods for Registry Objects owned by  
2613 them
- 2614 • The RegistryAdministrator has access to all methods on all Registry Objects
- 2615 • Unauthenticated clients can access all read-only (getXXX) methods
- 2616 • At the time of content submission, the Registry must assign the default  
2617 ContentOwner role to the Submitting Organization (SO) as authenticated by the  
2618 credentials in the submission message. In the current version of this  
2619 specification, it will be the DN as identified by the certificate
- 2620 • Clients that browse the Registry need not use certificates. The Registry must  
2621 assign the default RegistryGuest role to such clients.

2622 **Appendix A Schemas and DTD Definitions**

2623 The following are definitions for the various ebXML Message payloads described in this  
2624 document.

2625 **A.1 ebXML Error Message**

2626 The following “error” syntax is copied from the ebXML Message Services Specification,  
2627 version 0.99, lines 2364 to 2389:

```

2628 <!-- ERROR LIST -->
2629 <element name="ErrorList">
2630   <complexType>
2631     <sequence>
2632       <element ref="tns:Error" maxOccurs="unbounded"/>
2633     </sequence>
2634     <attribute ref="tns:id"/>
2635     <attribute ref="tns:version"/>
2636     <attribute ref="soap:mustUnderstand" use="required"/>
2637   </complexType>

```



```
2638     <attribute
2639         name="highestSeverity"
2640         type="tns:severity.type"
2641         use="default"
2642         value="Warning" />
2643     <anyAttribute
2644         namespace="http://www.w3.org/2000/10/XMLSchema-instance"
2645         processContents="lax"/>
2646 </complexType>
2647 </element>
2648 <element name="Error">
2649     <complexType>
2650         <attribute ref="tns:id"/>
2651         <attribute
2652             name="codeContext"
2653             type="uriReference"
2654             use="required"/>
2655         <attribute
2656             name="errorCode"
2657             type="tns:non-empty-string"
2658             use="required"/>
2659         <attribute
2660             name="severity"
2661             type="tns:severity.type"
2662             use="default"
2663             value="Warning"/>
2664         <attribute
2665             name="location"
2666             type="tns:non-empty-string"/>
2667         <attribute
2668             ref="xml:lang"/>
2669         <attribute
2670             name="errorMessage"
2671             type="tns:non-empty-string"/>
2672     </complexType>
2673 </element>
```

## 2674 **A.2 ebXML Registry DTD**

```
2675
2676 <?xml version="1.0" encoding="UTF-8"?>
2677 <!-- Begin information model mapping. -->
2678
2679 <!ENTITY % errorSchema SYSTEM "ebXMLError.dtd">
2680 %errorSchema;
2681
2682 <!--
2683 ObjectAttributes are attributes from the RegistryObject interface in ebRIM.
2684
2685 id may be empty. If specified it may be in urn:uuid format or be in some
2686 arbitrary format. If id is empty registry must generate globally unique id.
2687
2688 If id is provided and in proper UUID syntax (starts with urn:uuid:)
2689 registry will honour it.
2690
```

2691 If id is provided and is not in proper UUID syntax then it is used for  
2692 linkage within document and is ignored by the registry. In this case the  
2693 registry generates a UUID for id attribute.  
2694

2695 id must not be null when object is being retrieved from the registry.  
2696 -->

```
2697 <!ENTITY % ObjectAttributes "  
2698     id          ID #IMPLIED  
2699     name        CDATA #IMPLIED  
2700     description CDATA #IMPLIED  
2701 ">  
2702  
2703 <!--  
2704 Use as a proxy for an Object that is in the registry already.  
2705 Specifies the id attribute of the object in the registry as its id attribute.  
2706 id attribute in ObjectAttributes is exactly the same syntax and semantics as  
2707 id attribute in RegistryObject.  
2708 -->  
2709 <!ELEMENT ObjectRef EMPTY>  
2710 <!ATTLIST ObjectRef  
2711     id ID #IMPLIED  
2712 >  
2713  
2714 <!ELEMENT ObjectRefList (ObjectRef)*>  
2715  
2716 <!--  
2717 RegistryEntryAttributes are attributes from the RegistryEntry interface  
2718 in ebRIM.  
2719 It inherits ObjectAttributes  
2720 -->  
2721 <!ENTITY % RegistryEntryAttributes "%ObjectAttributes;  
2722     majorVersion    CDATA '1'  
2723     minorVersion    CDATA '0'  
2724     objectType      CDATA #REQUIRED  
2725     status           CDATA #IMPLIED  
2726     userVersion     CDATA #IMPLIED  
2727     stability        CDATA 'Dynamic'  
2728     expirationDate  CDATA #IMPLIED">  
2729  
2730 <!ELEMENT RegistryEntry (SlotList?)>  
2731 <!ATTLIST RegistryEntry  
2732     %RegistryEntryAttributes; >  
2733 <!ELEMENT Value (#PCDATA)>  
2734 <!ELEMENT ValueList (Value*)>  
2735 <!ELEMENT Slot (ValueList?)>  
2736 <!ATTLIST Slot  
2737     name CDATA #REQUIRED  
2738     slotType CDATA #IMPLIED  
2739 >  
2740 <!ELEMENT SlotList (Slot*)>  
2741  
2742 <!--  
2743 ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.  
2744 It inherits RegistryEntryAttributes  
2745 -->  
2746
```

```
2747
2748 <!ELEMENT ExtrinsicObject EMPTY >
2749 <!ATTLIST ExtrinsicObject
2750     %RegistryEntryAttributes;
2751     contentURI CDATA #REQUIRED
2752     mimeType CDATA #IMPLIED
2753     opaque (true | false) "false"
2754 >
2755
2756
2757 <!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
2758
2759 <!-- Leaf classes that reflect the concrete classes in ebRIM -->
2760 <!ELEMENT RegistryEntryList (Association | Classification |
2761     ClassificationNode | Package |
2762     ExternalLink | ExternalIdentifier
2763     | Organization | ExtrinsicObject |
2764     ObjectRef)*>
2765
2766 <!--
2767 An ExternalLink specifies a link from a RegistryEntry and an external URI
2768 -->
2769 <!ELEMENT ExternalLink EMPTY>
2770 <!ATTLIST ExternalLink
2771     %IntrinsicObjectAttributes;
2772     externalURI CDATA #IMPLIED
2773 >
2774
2775 <!--
2776 An ExternalIdentifier provides an identifier for a RegistryEntry
2777
2778 The value is the value of the identifier (e.g. the social security number)
2779 -->
2780 <!ELEMENT ExternalIdentifier EMPTY>
2781 <!ATTLIST ExternalIdentifier
2782     %IntrinsicObjectAttributes;
2783     value CDATA #REQUIRED
2784 >
2785
2786 <!--
2787 An Association specifies references to two previously submitted
2788 registry entrys.
2789
2790 The sourceObject is id of the sourceObject in association
2791 The targetObject is id of the targetObject in association
2792 -->
2793 <!ELEMENT Association EMPTY>
2794 <!ATTLIST Association
2795     %IntrinsicObjectAttributes;
2796     sourceRole CDATA #IMPLIED
2797     targetRole CDATA #IMPLIED
2798     associationType CDATA #REQUIRED
2799     bidirection (true | false) "false"
2800     sourceObject IDREF #REQUIRED
2801     targetObject IDREF #REQUIRED
2802 >
```

```
2803
2804 <!--
2805 A Classification specifies references to two registry entrys.
2806
2807 The classifiedObject is id of the Object being classified.
2808 The classificationNode is id of the ClassificationNode classying the object
2809 -->
2810 <!ELEMENT Classification EMPTY>
2811 <!ATTLIST Classification
2812     %IntrinsicObjectAttributes;
2813     classifiedObject IDREF #REQUIRED
2814     classificationNode IDREF #REQUIRED
2815 >
2816
2817 <!--
2818 A Package is a named collection of objects.
2819 -->
2820 <!ELEMENT Package EMPTY>
2821 <!ATTLIST Package
2822     %IntrinsicObjectAttributes;
2823 >
2824
2825 <!-- Attributes inherited by various types of telephone number elements -->
2826 <!ENTITY % TelephoneNumberAttributes " areaCode CDATA #REQUIRED
2827     contryCode CDATA #REQUIRED
2828     extension CDATA #IMPLIED
2829     number CDATA #REQUIRED
2830     url CDATA #IMPLIED">
2831 <!ELEMENT TelephoneNumber EMPTY>
2832 <!ATTLIST TelephoneNumber
2833     %TelephoneNumberAttributes;
2834 >
2835 <!ELEMENT FaxNumber EMPTY>
2836 <!ATTLIST FaxNumber
2837     %TelephoneNumberAttributes;
2838 >
2839
2840 <!ELEMENT PagerNumber EMPTY>
2841 <!ATTLIST PagerNumber
2842     %TelephoneNumberAttributes;
2843 >
2844
2845 <!ELEMENT MobileTelephoneNumber EMPTY>
2846 <!ATTLIST MobileTelephoneNumber
2847     %TelephoneNumberAttributes;
2848 >
2849 <!-- PostalAddress -->
2850 <!ELEMENT PostalAddress EMPTY>
2851 <!ATTLIST PostalAddress
2852     city CDATA #REQUIRED
2853     country CDATA #REQUIRED
2854     postalCode CDATA #REQUIRED
2855     state CDATA #REQUIRED
2856     street CDATA #REQUIRED
2857 >
2858 <!-- PersonName -->
```

```
2859 <!ELEMENT PersonName EMPTY>
2860 <!ATTLIST PersonName
2861     firstName CDATA #REQUIRED
2862     middleName CDATA #REQUIRED
2863     lastName CDATA #REQUIRED
2864 >
2865
2866 <!-- Organization -->
2867 <!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
2868 <!ATTLIST Organization
2869     %IntrinsicObjectAttributes;
2870     parent IDREF #IMPLIED
2871     primaryContact IDREF #REQUIRED
2872 >
2873
2874 <!ELEMENT User (PersonName, PostalAddress, TelephoneNumber,
2875     MobileTelephoneNumber?,
2876     FaxNumber?, PagerNumber?)>
2877 <!ATTLIST User
2878     %ObjectAttributes;
2879     organization IDREF #IMPLIED
2880     email CDATA #IMPLIED
2881     url CDATA #IMPLIED
2882 >
2883
2884 <!ELEMENT AuditableEvent EMPTY>
2885 <!ATTLIST AuditableEvent
2886     %ObjectAttributes;
2887     eventType CDATA #REQUIRED
2888     registryEntry IDREF #REQUIRED
2889     timestamp CDATA #REQUIRED
2890     user IDREF #REQUIRED
2891 >
2892
2893 <!--
2894 ClassificationNode is used to submit a Classification tree to the Registry.
2895
2896 parent is the id to the parent node. code is an optional code value for a
2897     ClassificationNode
2898 often defined by an external taxonomy (e.g. NAICS)
2899 -->
2900 <!ELEMENT ClassificationNode EMPTY>
2901 <!ATTLIST ClassificationNode
2902     %IntrinsicObjectAttributes;
2903     parent IDREF #IMPLIED
2904     code CDATA #IMPLIED
2905 >
2906
2907 <!--
2908 End information model mapping.
2909
2910 Begin Registry Services Interface
2911 -->
2912 <!ELEMENT RequestAcceptedResponse EMPTY>
2913 <!ATTLIST RequestAcceptedResponse
2914     xml:lang NMTOKEN #REQUIRED
```

```
2915         interfaceId CDATA #REQUIRED
2916         requestMessage CDATA #REQUIRED
2917         actionId CDATA #REQUIRED
2918     >
2919     <!--
2920     The SubmittedObject provides meta data for submitted object
2921     Note object being submitted is in a separate document that is not
2922     in this DTD.
2923     -->
2924     <!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
2925     <!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
2926     <!-- Only need name in Slot within SlotList -->
2927     <!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
2928     <!--
2929     The ObjectRefList is the list of
2930     refs to the registry entrys being approved.
2931     -->
2932     <!ELEMENT ApproveObjectsRequest (ObjectRefList)>
2933     <!--
2934     The ObjectRefList is the list of
2935     refs to the registry entrys being deprecated.
2936     -->
2937     <!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
2938     <!--
2939     The ObjectRefList is the list of
2940     refs to the registry entrys being removed
2941     -->
2942     <!ELEMENT RemoveObjectsRequest (ObjectRefList)>
2943     <!ATTLIST RemoveObjectsRequest
2944         deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
2945     >
2946     <!ELEMENT GetRootClassificationNodesRequest EMPTY>
2947     <!--
2948     The namePattern follows SQL-92 syntax for the pattern specified in
2949     LIKE clause. It allows for selecting only those root nodes that match
2950     the namePattern. The default value of '*' matches all root nodes.
2951     -->
2952     <!ATTLIST GetRootClassificationNodesRequest
2953         namePattern CDATA "*"
2954     >
2955     <!--
2956     The response includes one or more ClassificationNodes
2957     -->
2958     <!ELEMENT GetRootClassificationNodesResponse ((ClassificationNode+ |
2959         ebXMLError)>
2960     <!--
2961     Get the classification tree under the ClassificationNode specified parentRef.
2962
2963     If depth is 1 just fetch immediate child
2964     nodes, otherwise fetch the descendant tree upto the specified depth level.
2965     If depth is 0 that implies fetch entire sub-tree
2966     -->
2967     <!ELEMENT GetClassificationTreeRequest EMPTY>
2968     <!ATTLIST GetClassificationTreeRequest
2969         parent CDATA #REQUIRED
2970         depth CDATA "1"
```

```
2971 >
2972 <!--
2973 The response includes one or more ClassificationNodes which includes only
2974 immediate ClassificationNode children nodes if depth attribute in
2975 GetClassificationTreeRequest was 1, otherwise the decendent nodes
2976 upto specified depth level are returned.
2977 -->
2978 <!ELEMENT GetClassificationTreeResponse ((ClassificationNode+) | ebXMLError)>
2979 <!--
2980 Get refs to all registry entrys that are classified by all the
2981 ClassificationNodes specified by ObjectRefList.
2982 Note this is an implicit logical AND operation
2983 -->
2984 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
2985 <!--
2986 objectType attribute can specify the type of objects that the registry
2987 client is interested in, that is classified by this ClassificationNode.
2988 It is a String that matches a choice in the type attribute of
2989                                     ExtrinsicObject.
2990 The default value of '*' implies that client is interested in all types
2991 of registry entrys that are classified by the specified ClassificationNode.
2992 -->
2993 <!--
2994 The response includes a RegistryEntryList which has zero or more
2995 RegistryEntrys that are classified by the ClassificationNodes
2996 specified in the ObjectRefList in GetClassifiedObjectsRequest.
2997 -->
2998 <!ELEMENT GetClassifiedObjectsResponse (RegistryEntryList | ebXMLError)>
2999 <!--
3000 An Ad hoc query request specifies a query string as defined by [RS] in the
3001                                     queryString attribute
3002 -->
3003 <!ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |
3004                                     ReturnRepositoryItem | SQLQuery)>
3005 <!ELEMENT SQLQuery (#PCDATA)>
3006 <!--
3007 The response includes a RegistryEntryList which has zero or more
3008 RegistryEntrys that match the query specified in AdhocQueryRequest.
3009 -->
3010 <!ELEMENT AdhocQueryResponse (RegistryEntryList | FilterQueryResult |
3011                                     ReturnRegistryEntryResult |
3012                                     ReturnRepositoryItemResult |
3013                                     ebXMLError)>
3014 <!--
3015 Gets the actual content (not metadata) specified by the ObjectRefList
3016 -->
3017 <!ELEMENT GetContentRequest (ObjectRefList)>
3018 <!--
3019 The GetObjectsResponse will have no sub-elements if there were no errors.
3020 The actual contents will be in the other payloads of the message.
3021 If any errors were encountered the message will contain the ebXMLError and
3022 the content payloads will be empty.
3023 -->
3024 <!ELEMENT GetContentResponse (ebXMLError?)>
3025 <!--
3026 Describes the capability profile for the registry and what optional features
```

```
3027 are supported
3028 -->
3029 <!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
3030 <!ATTLIST RegistryProfile
3031     version CDATA #REQUIRED
3032 >
3033
3034 <!ELEMENT OptionalFeaturesSupported EMPTY>
3035 <!ATTLIST OptionalFeaturesSupported
3036     sqlQuery (true | false) "false"
3037     xQuery (true | false) "false"
3038 >
3039 <!-- Begin FilterQuery DTD -->
3040 <!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
3041     ClassificationNodeQuery |
3042     RegistryPackageQuery |
3043     OrganizationQuery)>
3044 <!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
3045     AuditableEventQueryResult |
3046     ClassificationNodeQueryResult |
3047     RegistryPackageQueryResult |
3048     OrganizationQueryResult)>
3049 <!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
3050 <!ELEMENT RegistryEntryView EMPTY>
3051 <!ATTLIST RegistryEntryView
3052     objectURN CDATA #REQUIRED
3053     contentURI CDATA #IMPLIED
3054     objectID CDATA #IMPLIED
3055 >
3056 <!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
3057 <!ELEMENT AuditableEventView EMPTY>
3058 <!ATTLIST AuditableEventView
3059     objectID CDATA #REQUIRED
3060     timestamp CDATA #REQUIRED
3061 >
3062 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
3063 <!ELEMENT ClassificationNodeView EMPTY>
3064 <!ATTLIST ClassificationNodeView
3065     objectURN CDATA #REQUIRED
3066     contentURI CDATA #IMPLIED
3067     objectID CDATA #IMPLIED
3068 >
3069 <!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
3070 <!ELEMENT RegistryPackageView EMPTY>
3071 <!ATTLIST RegistryPackageView
3072     objectURN CDATA #REQUIRED
3073     contentURI CDATA #IMPLIED
3074     objectID CDATA #IMPLIED
3075 >
3076 <!ELEMENT OrganizationQueryResult (OrganizationView*)>
3077 <!ELEMENT OrganizationView EMPTY>
3078 <!ATTLIST OrganizationView
3079     orgURN CDATA #REQUIRED
3080     objectID CDATA #IMPLIED
3081 >
3082 <!ELEMENT StatusResult (Success | (Exception | Warning)+)>
```



```
3083 <!ELEMENT Success EMPTY>
3084 <!ELEMENT Exception (#PCDATA)>
3085 <!ATTLIST Exception
3086     code CDATA #REQUIRED
3087 >
3088 <!ELEMENT Warning (#PCDATA)>
3089 <!ATTLIST Warning
3090     code CDATA #REQUIRED
3091 >
3092 <!ELEMENT RegistryEntryQuery (RegistryEntryFilter?, SourceAssociationBranch*,
3093     TargetAssociationBranch*,
3094     HasClassificationBranch*,
3095     SubmittingOrganizationBranch?,
3096     ResponsibleOrganizationBranch?,
3097     ExternalLinkFilter*,
3098     HasAuditableEventBranch*)>
3099 <!ELEMENT SourceAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3100 <!ELEMENT TargetAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3101 <!ELEMENT HasClassificationBranch (ClassificationFilter?,
3102     ClassificationNodeFilter?)>
3103 <!ELEMENT SubmittingOrganizationBranch (OrganizationFilter?, ContactFilter?)>
3104 <!ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
3105     ContactFilter?)>
3106 <!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
3107     OrganizationFilter?)>
3108 <!ELEMENT AuditableEventQuery
3109     (AuditableEventFilter?, RegistryEntryQuery*, InvokedByBranch? )>
3110
3111 <!ELEMENT InvokedByBranch
3112     ( UserFilter?, OrganizationQuery? )>
3113
3114 <!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
3115     PermitsClassificationBranch*,
3116     HasParentNode?, HasSubnode*)>
3117 <!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
3118     RegistryEntryQuery?)>
3119 <!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
3120 <!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
3121 <!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
3122 <!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
3123 <!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
3124     HasParentOrganization?,
3125     InvokesEventBranch*,
3126     ContactFilter*)>
3127 <!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
3128 <!ELEMENT HasParentOrganization (OrganizationFilter?,
3129     HasParentOrganization?)>
3130 <!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
3131     RegistryEntryQuery?)>
3132 <!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
3133     WithSourceAssociations?,
3134     WithTargetAssociations?,
3135     WithAuditableEvents?,
3136     WithExternalLinks?)>
3137 <!ELEMENT WithClassifications (ClassificationFilter?)>
3138 <!ELEMENT WithSourceAssociations (AssociationFilter?)>
```

```
3139 <!ELEMENT WithTargetAssociations (AssociationFilter?)>
3140 <!ELEMENT WithAuditableEvents (AuditableEventFilter?)>
3141 <!ELEMENT WithExternalLinks (ExternalLinkFilter?)>
3142 <!ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*, StatusResult)>
3143 <!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
3144                               SourceAssociations?,
3145                               TargetAssociations?,
3146                               AuditableEvent*, ExternalLink*)>
3147 <!ELEMENT SourceAssociations (Association*)>
3148 <!ELEMENT TargetAssociations (Association*)>
3149 <!ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
3150                               RecursiveAssociationOption?,
3151                               WithDescription?)>
3152 <!ELEMENT RecursiveAssociationOption (AssociationType+)>
3153 <!ATTLIST RecursiveAssociationOption
3154     depthLimit CDATA #IMPLIED
3155 >
3156 <!ELEMENT AssociationType EMPTY>
3157 <!ATTLIST AssociationType
3158     role CDATA #REQUIRED
3159 >
3160 <!ELEMENT WithDescription EMPTY>
3161 <!ELEMENT ReturnRepositoryItemResult (RepositoryItem*, StatusResult)>
3162 <!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject
3163                          | ExternalLink)>
3164 <!ATTLIST RepositoryItem
3165     identifier CDATA #REQUIRED
3166     name CDATA #REQUIRED
3167     contentURI CDATA #REQUIRED
3168     objectType CDATA #REQUIRED
3169     status CDATA #REQUIRED
3170     stability CDATA #REQUIRED
3171     description CDATA #IMPLIED
3172 >
3173 <!ELEMENT RegistryPackage EMPTY>
3174 <!ELEMENT WithdrawnObject EMPTY>
3175 <!ELEMENT ExternalLinkItem EMPTY>
3176 <!ELEMENT ObjectFilter (Clause)>
3177 <!ELEMENT RegistryEntryFilter (Clause)>
3178 <!ELEMENT IntrinsicObjectFilter (Clause)>
3179 <!ELEMENT ExtrinsicObjectFilter (Clause)>
3180 <!ELEMENT PackageFilter (Clause)>
3181 <!ELEMENT OrganizationFilter (Clause)>
3182 <!ELEMENT ContactFilter (Clause)>
3183 <!ELEMENT ClassificationNodeFilter (Clause)>
3184 <!ELEMENT AssociationFilter (Clause)>
3185 <!ELEMENT ClassificationFilter (Clause)>
3186 <!ELEMENT ExternalLinkFilter (Clause)>
3187 <!ELEMENT AuditableEventFilter (Clause)>
3188 <!ELEMENT UserFilter (Clause)>
3189
3190 <!--
3191 The following lines define the XML syntax for Clause.
3192 -->
3193
3194 <!ELEMENT Clause (SimpleClause | CompoundClause)>
```

```
3195 <!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
3196 <!ATTLIST SimpleClause
3197     leftArgument CDATA #REQUIRED
3198 >
3199 <!ELEMENT CompoundClause (Clause, Clause+)>
3200 <!ATTLIST CompoundClause
3201     connectivePredicate (And | Or) #REQUIRED
3202 >
3203 <!ELEMENT BooleanClause EMPTY>
3204 <!ATTLIST BooleanClause
3205     booleanPredicate (true | false) #REQUIRED
3206 >
3207 <!ELEMENT RationalClause (IntClause | FloatClause)>
3208 <!ATTLIST RationalClause
3209     logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
3210 >
3211 <!ELEMENT IntClause (#PCDATA)>
3212 <!ATTLIST IntClause
3213     e-dtype NMTOKEN #FIXED "int"
3214 >
3215 <!ELEMENT FloatClause (#PCDATA)>
3216 <!ATTLIST FloatClause
3217     e-dtype NMTOKEN #FIXED "float"
3218 >
3219 <!ELEMENT StringClause (#PCDATA)>
3220 <!ATTLIST StringClause
3221     stringPredicate
3222     (contains | -contains |
3223      startswith | -startswith |
3224      equal | -equal |
3225      endswith | -endswith) #REQUIRED
3226 >
3227 <!-- End FilterQuery DTD -->
3228
3229 <!-- The contrived root node -->
3230 <!ELEMENT RootElement
3231     ( RequestAcceptedResponse |
3232       ebXMLError |
3233       SubmitObjectsRequest |
3234       ApproveObjectsRequest |
3235       DeprecateObjectsRequest |
3236       RemoveObjectsRequest |
3237       GetRootClassificationNodesRequest |
3238       GetRootClassificationNodesResponse |
3239       GetClassificationTreeRequest |
3240       GetClassificationTreeResponse |
3241       GetClassifiedObjectsRequest |
3242       GetClassifiedObjectsResponse |
3243       AdhocQueryRequest |
3244       AdhocQueryResponse |
3245       GetContentRequest |
3246       GetContentResponse |
3247       AddSlotsRequest |
3248       RemoveSlotsRequest |
3249       RegistryProfile) >
3250
```

3251 <!ELEMENT Href (#PCDATA )>  
3252  
3253 <!ELEMENT XMLDocumentErrorLocn (DocumentId , Xpath )>  
3254  
3255 <!ELEMENT DocumentId (#PCDATA )>  
3256  
3257 <!ELEMENT Xpath (#PCDATA)>  
3258

## 3259 **Appendix B Interpretation of UML Diagrams**

3260 This section describes in *abstract terms* the conventions used to define ebXML  
3261 business process description in UML.

### 3262 **B.1 UML Class Diagram**

3263 A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP])  
3264 required to implement an ebXML Registry Services and clients. See Figure 2 on page  
3265 14 for an example. The UML class diagram contains:

- 3266
- 3267 1. A collection of UML interfaces where each interface represents a Service  
3268 Interface for a Registry service.
  - 3269 2. Tabular description of methods on each interface where each method represents  
3270 an Action (as defined by [ebCPP]) within the Service Interface representing the  
3271 UML interface.
  - 3272 3. Each method within a UML interface specifies one or more parameters, where  
3273 the type of each method argument represents the ebXML message type that is  
3274 exchanged as part of the Action corresponding to the method. Multiple  
3275 arguments imply multiple payload documents within the body of the  
3276 corresponding ebXML message.

### 3277 **B.2 UML Sequence Diagram**

3278 A UML sequence diagram is used to specify the business protocol representing the  
3279 interactions between the UML interfaces for a Registry specific ebXML business  
3280 process. A UML sequence diagram provides the necessary information to determine the  
3281 sequencing of messages, request to response association as well as request to error  
3282 response association as described by [ebCPP].

3283 Each sequence diagram shows the sequence for a specific conversation protocol as  
3284 method calls from the requestor to the responder. Method invocation may be  
3285 synchronous or asynchronous based on the UML notation used on the arrow-head for  
3286 the link. A half arrow-head represents asynchronous communication. A full arrow-head  
3287 represents synchronous communication.

3288 Each method invocation may be followed by a response method invocation from the  
3289 responder to the requestor to indicate the ResponseName for the previous Request.  
3290 Possible error response is indicated by a conditional response method invocation from  
3291 the responder to the requestor. See Figure 4 on page 21 for an example.

## 3292 **Appendix C SQL Query**

### 3293 **C.1 SQL Query Syntax Specification**

3294 This section specifies the rules that define the SQL Query syntax as a subset of SQL-  
3295 92. The terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The  
3296 SQL query syntax conforms to the <query specification>, modulo the restrictions  
3297 identified below:

- 3298 1. A <select list> may contain at most one <select sublist>.
- 3299 2. In a <select list> must be is a single column whose data type is UUID, from the  
3300 table in the <from clause>.
- 3301 3. A <derived column> may not have an <as clause>.
- 3302 4. <table expression> does not contain the optional <group by clause> and <having  
3303 clause> clauses.
- 3304 5. A <table reference> can only consist of <table name> and <correlation name>.
- 3305 6. A <table reference> does not have the optional AS between <table name> and  
3306 <correlation name>.
- 3307 7. There can only be one <table reference> in the <from clause>.
- 3308 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in  
3309 predicate> allows for the right hand side of the <in predicate> to be limited to a  
3310 restricted <query specification> as defined above.
- 3311 9. A <search condition> within the <where clause> may not include a <query  
3312 expression>.
- 3313 10. The SQL query syntax allows for the use of <sql invoked routines>  
3314 invocation from [SQL/PSM] as the RHS of the <in predicate>.

3315 **C.2 Non-Normative BNF for Query Syntax Grammar**

3316 The following BNF exemplifies the grammar for the registry query syntax. It is provided  
 3317 here as an aid to implementors. Since this BNF is not based on [SQL] it is provided as  
 3318 non-normative syntax. For the normative syntax rules see Appendix C.1.

```

3319
3320
3321 /*****
3322  * The Registry Query (Subset of SQL-92) grammar starts here
3323  *****/
3324
3325 RegistryQuery = SQLSelect [ ";" ]
3326
3327 SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
3328
3329 SQLSelectCols = ID
3330
3331 SQLTableList = SQLTableRef
3332
3333 SQLTableRef = ID
3334
3335 SQLWhere = "WHERE" SQLOrExpr
3336
3337 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
3338
3339 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *
3340
3341 SQLNotExpr = [ "NOT" ] SQLCompareExpr
3342
3343 SQLCompareExpr =
3344     ( SQLColRef "IS" ) SQLIsClause
3345     | SQLSumExpr [ SQLCompareExprRight ]
3346
3347
3348 SQLCompareExprRight =
3349     SQLLikeClause
3350     | SQLInClause
3351     | SQLCompareOp SQLSumExpr
3352
3353 SQLCompareOp =
3354     "="
3355     | "<>"
3356     | ">"
3357     | ">="
3358     | "<"
3359     | "<="
3360
3361 SQLInClause = [ "NOT" ] "IN" "(" SQLValueList ")"
3362
3363 SQLValueList = SQLValueElement ( "," SQLValueElement ) *
3364
3365 SQLValueElement = "NULL" | SQLSelect
3366
3367 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
3368
3369 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
3370
3371 SQLPattern = STRING_LITERAL
3372
3373 SQLLiteral =
3374     STRING_LITERAL
3375     | INTEGER_LITERAL
3376     | FLOATING_POINT_LITERAL
3377
3378 SQLColRef = SQLValue
3379
    
```

```

3380 SQLLvalue = SQLLvalueTerm
3381
3382 SQLLvalueTerm = ID ( "." ID ) *
3383
3384 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr ) *
3385
3386 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr ) *
3387
3388 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
3389
3390 SQLTerm = "(" SQLOrExpr ")"
3391 | SQLColRef
3392 | SQLLiteral
3393
3394 INTEGER_LITERAL = ([ "0"-"9" ] ) +
3395
3396 FLOATING_POINT_LITERAL =
3397   ([ "0"-"9" ] ) + "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3398 | "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3399 | ([ "0"-"9" ] ) + EXPONENT
3400 | ([ "0"-"9" ] ) + ( EXPONENT ) ?
3401
3402 EXPONENT = [ "e", "E" ] ( [ "+", "-" ] ) ? ([ "0"-"9" ] ) +
3403
3404 STRING_LITERAL: "'" (~[ "'" ] ) * ( '"' (~[ '"' ] ) * ) * "'"
3405
3406 ID = ( <LETTER> ) + ( "-" | "$" | "#" | <DIGIT> | <LETTER> ) *
3407 LETTER = [ "A"-"Z", "a"-"z" ]
3408 DIGIT = [ "0"-"9" ]

```

### 3409 C.3 Relational Schema For SQL Queries

```

3410 --SQL Load file for creating the ebXML Registry tables
3411
3412
3413
3414 --Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92
3415
3416
3417 CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
3418 CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
3419 CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
3420
3421 CREATE TYPE UUID UNDER ShortName FINAL;
3422 CREATE TYPE URI UNDER LongName FINAL;
3423
3424 CREATE TABLE ExtrinsicObject (
3425
3426 --RegistryObject Attributes
3427   id                                UUID PRIMARY KEY NOT NULL,
3428   name                              LongName,
3429   description                        FreeFormText,
3430   accessControlPolicy                UUID NOT NULL,
3431
3432 --Versionable attributes
3433   majorVersion                      INT DEFAULT 0 NOT NULL,
3434   minorVersion                      INT DEFAULT 1 NOT NULL,
3435
3436 --RegistryEntry attributes
3437   status                            INT DEFAULT 0 NOT NULL,
3438   userVersion                       ShortName,
3439   stability                          INT DEFAULT 0 NOT NULL,
3440   expirationDate                    TIMESTAMP,
3441
3442 --ExtrinsicObject attributes
3443   contentURI                        URI,
3444   mimeType                          ShortName,
3445   objectType                        INT DEFAULT 0 NOT NULL,

```

```

3446     opaque                                BOOLEAN DEFAULT false NOT NULL
3447
3448 );
3449
3450 CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
3451 --Must return a collection of UUIDs for related RegistryEntry instances
3452 }
3453
3454 CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
3455 --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
3456 --Collection must be in ascending order by timestamp
3457 }
3458
3459 CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
3460 --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
3461 }
3462
3463 CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
3464 --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
3465 }
3466
3467 CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
3468 --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
3469 }
3470
3471 CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
3472 --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
3473 }
3474
3475 CREATE TABLE Package (
3476
3477 --RegistryObject Attributes
3478     id                                UUID PRIMARY KEY NOT NULL,
3479     name                                LongName,
3480     description                        FreeFormText,
3481     accessControlPolicy                UUID NOT NULL,
3482
3483 --Versionable attributes
3484     majorVersion                       INT DEFAULT 0 NOT NULL,
3485     minorVersion                       INT DEFAULT 1 NOT NULL,
3486
3487 --RegistryEntry attributes
3488     status                              INT DEFAULT 0 NOT NULL,
3489     userVersion                         ShortName,
3490     stability                           INT     DEFAULT 0 NOT NULL,
3491     expirationDate                     TIMESTAMP,
3492
3493 --Package attributes
3494 );
3495
3496 CREATE PROCEDURE Package_memberbjects(packageId) {
3497 --Must return a collection of UUIDs for RegistryEntrys that are memebers of this Package.
3498 }
3499
3500 CREATE TABLE ExternalLink (
3501
3502 --RegistryObject Attributes
3503     id                                UUID PRIMARY KEY NOT NULL,
3504     name                                LongName,
3505     description                        FreeFormText,
3506     accessControlPolicy                UUID NOT NULL,
3507
3508 --Versionable attributes
3509     majorVersion                       INT DEFAULT 0 NOT NULL,
3510     minorVersion                       INT DEFAULT 1 NOT NULL,
3511
3512 --RegistryEntry attributes
3513     status                              INT DEFAULT 0 NOT NULL,
3514     userVersion                         ShortName,
3515     stability                           INT     DEFAULT 0 NOT NULL,

```



```

3516     expirationDate                TIMESTAMP,
3517
3518 --ExternalLink attributes
3519     externalURI                    URI NOT NULL
3520 );
3521
3522 CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
3523 --Must return a collection of UUIDs for objects in this relationship
3524 }
3525
3526 CREATE TABLE ExternalIdentifier (
3527
3528 --RegistryObject Attributes
3529     id                               UUID PRIMARY KEY NOT NULL,
3530     name                             LongName,
3531     description                       FreeFormText,
3532     accessControlPolicy               UUID NOT NULL,
3533
3534 --Versionable attributes
3535     majorVersion                     INT DEFAULT 0 NOT NULL,
3536     minorVersion                     INT DEFAULT 1 NOT NULL,
3537
3538 --RegistryEntry attributes
3539     status                            INT DEFAULT 0 NOT NULL,
3540     userVersion                       ShortName,
3541     stability                          INT     DEFAULT 0 NOT NULL,
3542     expirationDate                   TIMESTAMP,
3543
3544 --ExternalIdentifier attributes
3545     value                             ShortName NOT NULL
3546 );
3547
3548
3549 --A SlotValue row represents one value of one slot in some
3550 --RegistryEntry
3551 CREATE TABLE SlotValue (
3552
3553 --RegistryObject Attributes
3554     registryEntry                   UUID     PRIMARY KEY NOT NULL,
3555
3556 --Slot attributes
3557     name                             LongName NOT NULL PRIMARY KEY NOT NULL,
3558     value                             ShortName NOT NULL
3559 );
3560
3561 CREATE TABLE Association (
3562 --RegistryObject Attributes
3563     id                               UUID PRIMARY KEY NOT NULL,
3564     name                             LongName,
3565     description                       FreeFormText,
3566     accessControlPolicy               UUID NOT NULL,
3567
3568 --Versionable attributes
3569     majorVersion                     INT DEFAULT 0 NOT NULL,
3570     minorVersion                     INT DEFAULT 1 NOT NULL,
3571
3572 --RegistryEntry attributes
3573     status                            INT DEFAULT 0 NOT NULL,
3574     userVersion                       ShortName,
3575     stability                          INT     DEFAULT 0 NOT NULL,
3576     expirationDate                   TIMESTAMP,
3577
3578 --Association attributes
3579     associationType                   INT NOT NULL,
3580     bidirectional                     BOOLEAN DEFAULT false NOT NULL,
3581     sourceObject                      UUID NOT NULL,
3582     sourceRole                        ShortName,
3583     label                             ShortName,
3584     targetObject                      UUID NOT NULL,
3585     targetRole                        ShortName

```

```

3586 );
3587
3588 --Classification is currently identical to Association
3589 CREATE TABLE Classification (
3590 --RegistryObject Attributes
3591     id                UUID PRIMARY KEY NOT NULL,
3592     name              LongName,
3593     description       FreeFormText,
3594     accessControlPolicy  UUID NOT NULL,
3595
3596 --Versionable attributes
3597     majorVersion     INT DEFAULT 0 NOT NULL,
3598     minorVersion     INT DEFAULT 1 NOT NULL,
3599
3600 --RegistryEntry attributes
3601     status            INT DEFAULT 0 NOT NULL,
3602     userVersion       ShortName,
3603     stability         INT     DEFAULT 0 NOT NULL,
3604     expirationDate   TIMESTAMP,
3605
3606 --Classification attributes. Assumes not derived from Association
3607     sourceObject     UUID NOT NULL,
3608     targetObject     UUID NOT NULL,
3609 );
3610
3611
3612 CREATE TABLE ClassificationNode (
3613 --RegistryObject Attributes
3614     id                UUID PRIMARY KEY NOT NULL,
3615     name              LongName,
3616     description       FreeFormText,
3617     accessControlPolicy  UUID NOT NULL,
3618
3619 --Versionable attributes
3620     majorVersion     INT DEFAULT 0 NOT NULL,
3621     minorVersion     INT DEFAULT 1 NOT NULL,
3622
3623 --RegistryEntry attributes
3624     status            INT DEFAULT 0 NOT NULL,
3625     userVersion       ShortName,
3626     stability         INT     DEFAULT 0 NOT NULL,
3627     expirationDate   TIMESTAMP,
3628
3629 --ClassificationNode attributes
3630     parent           UUID,
3631     path             VARCHAR(512) NOT NULL,
3632     code             ShortName
3633 );
3634
3635 CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
3636 --Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
3637 }
3638
3639 --Begin Registry Audit Trail tables
3640
3641 CREATE TABLE AuditableEvent (
3642 --RegistryObject Attributes
3643     id                UUID PRIMARY KEY NOT NULL,
3644     name              LongName,
3645     description       FreeFormText,
3646     accessControlPolicy  UUID NOT NULL,
3647
3648 --AuditableEvent attributes
3649     user             UUID,
3650     eventType        INT DEFAULT 0 NOT NULL,
3651     registryEntry    UUID NOT NULL,
3652     timestamp        TIMESTAMP NOT NULL,
3653 );
3654
3655

```

```

3656 CREATE TABLE User (
3657 --RegistryObject Attributes
3658   id                                UUID PRIMARY KEY NOT NULL,
3659   name                              LongName,
3660   description                        FreeFormText,
3661   accessControlPolicy                UUID NOT NULL,
3662
3663 --User attributes
3664   organization                        UUID NOT NULL
3665
3666 --address attributes flattened
3667   address_city                       ShortName,
3668   address_country                     ShortName,
3669   address_postalCode                  ShortName,
3670   address_state                       ShortName,
3671   address_street                      ShortName,
3672
3673   email                               ShortName,
3674
3675 --fax attribute flattened
3676   fax_areaCode                       VARCHAR(4) NOT NULL,
3677   fax_countryCode                    VARCHAR(4),
3678   fax_extension                       VARCHAR(8),
3679   fax_umber                           VARCHAR(8) NOT NULL,
3680   fax_url                             URI
3681
3682 --mobilePhone attribute flattened
3683   mobilePhone_areaCode                VARCHAR(4) NOT NULL,
3684   mobilePhone_countryCode             VARCHAR(4),
3685   mobilePhone_extension                VARCHAR(8),
3686   mobilePhone_umber                   VARCHAR(8) NOT NULL,
3687   mobilePhone_url                     URI
3688
3689 --name attribute flattened
3690   name_firstName                      ShortName,
3691   name_middleName                     ShortName,
3692   name_lastName                       ShortName,
3693
3694 --pager attribute flattened
3695   pager_areaCode                      VARCHAR(4) NOT NULL,
3696   pager_countryCode                   VARCHAR(4),
3697   pager_extension                       VARCHAR(8),
3698   pager_umber                           VARCHAR(8) NOT NULL,
3699   pager_url                             URI
3700
3701 --telephone attribute flattened
3702   telephone_areaCode                  VARCHAR(4) NOT NULL,
3703   telephone_countryCode                VARCHAR(4),
3704   telephone_extension                  VARCHAR(8),
3705   telephone_umber                      VARCHAR(8) NOT NULL,
3706   telephone_url                       URI,
3707
3708   url                                  URI,
3709
3710 );
3711
3712 CREATE TABLE Organization (
3713 --RegistryObject Attributes
3714   id                                UUID PRIMARY KEY NOT NULL,
3715   name                              LongName,
3716   description                        FreeFormText,
3717   accessControlPolicy                UUID NOT NULL,
3718
3719 --Versionable attributes
3720   majorVersion                       INT DEFAULT 0 NOT NULL,
3721   minorVersion                       INT DEFAULT 1 NOT NULL,
3722
3723 --RegistryEntry attributes
3724   status                              INT DEFAULT 0 NOT NULL,
3725

```

```

3726 userVersion ShortName,
3727 stability INT DEFAULT 0 NOT NULL,
3728 expirationDate TIMESTAMP,
3729
3730 --Organization attributes
3731
3732 --Organization.address attribute flattened
3733 address_city ShortName,
3734 address_country ShortName,
3735 address_postalCode ShortName,
3736 address_state ShortName,
3737 address_street ShortName,
3738
3739 --primary contact for Organization, points to a User.
3740 --Note many Users may belong to the same Organization
3741 contact UUID NOT NULL,
3742
3743 --Organization.fax attribute flattened
3744 fax_areaCode VARCHAR(4) NOT NULL,
3745 fax_countryCode VARCHAR(4),
3746 fax_extension VARCHAR(8),
3747 fax_umber VARCHAR(8) NOT NULL,
3748 fax_url URI,
3749
3750 --Organization.parent attribute
3751 parent UUID,
3752
3753 --Organization.telephone attribute flattened
3754 telephone_areaCode VARCHAR(4) NOT NULL,
3755 telephone_countryCode VARCHAR(4),
3756 telephone_extension VARCHAR(8),
3757 telephone_umber VARCHAR(8) NOT NULL,
3758 telephone_url URI
3759 );
3760
3761
3762 --Note that the ebRIM security view is not visible through the public query mechanism
3763 --in the current release
3764
3765
3766 --The RegistryEntry View allows polymorphic queries over all ebRIM classes derived
3767 --from RegistryEntry
3768
3769 CREATE VIEW RegistryEntry (
3770 --RegistryObject Attributes
3771 id,
3772 name,
3773 description,
3774 accessControlPolicy,
3775
3776 --Versionable attributes
3777 majorVersion,
3778 minorVersion,
3779
3780 --RegistryEntry attributes
3781 status,
3782 userVersion,
3783 stability,
3784 expirationDate
3785 ) AS
3786 SELECT
3787 --RegistryObject Attributes
3788 id,
3789 name,
3790 description,
3791 accessControlPolicy,
3792
3793 --Versionable attributes
3794 majorVersion,

```

```
3796     minorVersion,
3797
3798 --RegistryEntry attributes
3799     status,
3800     userVersion,
3801     stability,
3802     expirationDate
3803
3804 FROM ExtrinsicObject
3805 UNION
3806
3807 SELECT
3808 --RegistryObject Attributes
3809     id,
3810     name,
3811     description,
3812     accessControlPolicy,
3813
3814 --Versionable attributes
3815     majorVersion,
3816     minorVersion,
3817
3818 --RegistryEntry attributes
3819     status,
3820     userVersion,
3821     stability,
3822     expirationDate
3823 FROM (Registry)Package
3824 UNION
3825
3826 SELECT
3827 --RegistryObject Attributes
3828     id,
3829     name,
3830     description,
3831     accessControlPolicy,
3832
3833 --Versionable attributes
3834     majorVersion,
3835     minorVersion,
3836
3837 --RegistryEntry attributes
3838     status,
3839     userVersion,
3840     stability,
3841     expirationDate
3842 FROM ClassificationNode;
```

3843

## 3844 **Appendix D Non-normative Content Based Ad Hoc Queries**

3845 The Registry SQL query capability supports the ability to search for content based not  
3846 only on metadata that catalogs the content but also the data contained within the  
3847 content itself. For example it is possible for a client to submit a query that searches for  
3848 all Collaboration Party Profiles that define a role named "seller" within a RoleName  
3849 element in the CPP document itself. Currently content-based query capability is  
3850 restricted to XML content.

### 3851 **D.1.1 Automatic Classification of XML Content**

3852 Content-based queries are indirectly supported through the existing classification  
3853 mechanism supported by the Registry.

3854 A submitting organization may define logical indexes on any XML schema or DTD when  
3855 it is submitted. An instance of such a logical index defines a link between a specific  
3856 attribute or element node in an XML document tree and a ClassificationNode in a  
3857 classification scheme within the registry.

3858 The registry utilizes this index to automatically classify documents that are instances of  
3859 the schema at the time the document instance is submitted. Such documents are  
3860 classified according to the data contained within the document itself.

3861 Such automatically classified content may subsequently be discovered by clients using  
3862 the existing classification-based discovery mechanism of the Registry and the query  
3863 facilities of the ObjectQueryManager.

3864 [Note] This approach is conceptually similar to the way databases support  
3865 indexed retrieval. DBAs define indexes on tables in the schema. When  
3866 data is added to the table, the data gets automatically indexed.

### 3867 **D.1.2 Index Definition**

3868 This section describes how the logical indexes are defined in the SubmittedObject  
3869 element defined in the Registry DTD. The complete Registry DTD is specified in  
3870 Appendix A.2.

3871 A SubmittedObject element for a schema or DTD may define a collection of  
3872 ClassificationIndexes in a ClassificationIndexList optional element. The  
3873 ClassificationIndexList is ignored if the content being submitted is not of the SCHEMA  
3874 objectType.

3875 The ClassificationIndex element inherits the attributes of the base class RegistryObject  
3876 in [ebRIM]. It then defines specialized attributes as follows:

- 3877 1. classificationNode: This attribute references a specific ClassificationNode by its  
3878 ID.
- 3879 2. contentIdentifier: This attribute identifies a specific data element within the  
3880 document instances of the schema using an XPATH expression as defined by  
3881 [XPT].

### 3882 **D.1.3 Example Of Index Definition**

3883 To define an index that automatically classifies a CPP based upon the roles defined  
3884 within its RoleName elements, the following index must be defined on the CPP schema  
3885 or DTD:

```
3886 <ClassificationIndex  
3887   classificationNode='id-for-role-classification-scheme'  
3888   contentIdentifier='/Role//RoleName'  
3889 />
```

#### 3890 **D.1.4 Proposed XML Definition**

```
3891 <!--
3892 A ClassificationIndexList is specified on ExtrinsicObjects of objectType
3893 'Schema' to define an automatic Classification of instance objects of the
3894 schema using the specified classificationNode as parent and a
3895 ClassificationNode created or selected by the object content as selected by
3896 the contentIdentifier
3897 -->
3898 <!ELEMENT ClassificationIndex EMPTY>
3899 <!ATTLIST ClassificationIndex
3900     %ObjectAttributes;
3901     classificationNode IDREF #REQUIRED
3902     contentIdentifier CDATA #REQUIRED
3903 >
3904
3905 <!-- ClassificationIndexList contains new ClassificationIndexes -->
3906 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

#### 3907 **D.1.5 Example of Automatic Classification**

3908 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the  
3909 CPP is submitted it will automatically be classified by two ClassificationNodes named  
3910 “buyer” and “seller” that are both children of the ClassificationNode (e.g. a node named  
3911 Role) specified in the classificationNode attribute of the ClassificationIndex. Note that if  
3912 either of the two ClassificationNodes named “buyer” and “seller” did not previously exist,  
3913 the ObjectManager would automatically create these ClassificationNodes.

## 3914 **Appendix E Security Implementation Guideline**

3915 This section provides a suggested blueprint for how security processing may be  
3916 implemented in the Registry. It is meant to be illustrative not prescriptive. Registries  
3917 may choose to have different implementations as long as they support the default  
3918 security roles and authorization rules described in this document.

### 3919 **E.1 Authentication**

- 3920 1. As soon as a message is received, the first work is the authentication. A principal  
3921 object is created.
- 3922 2. If the message is signed, it is verified (including the validity of the certificate) and the  
3923 DN of the certificate becomes the identity of the principal. Then the Registry is  
3924 searched for the principal and if found, the roles and groups are filled in.
- 3925 3. If the message is not signed, an empty principal is created with the role  
3926 RegistryGuest. This step is for symmetry and to decouple the rest of the processing.
- 3927 4. Then the message is processed for the command and the objects it will act on.

## 3928 **E.2 Authorization**

3929 For every object, the access controller will iterate through all the AccessControlPolicy  
3930 objects with the object and see if there is a chain through the permission objects to  
3931 verify that the requested method is permitted for the Principal. If any of the permission  
3932 objects which the object is associated with has a common role, or identity, or group with  
3933 the principal, the action is permitted.

## 3934 **E.3 Registry Bootstrap**

3935 When a Registry is newly created, a default Principal object should be created with the  
3936 identity of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any  
3937 message signed by the Registry Admin will get all the privileges.

3938 When a Registry is newly created, a singleton instance of AccessControlPolicy is  
3939 created as the default AccessControlPolicy. This includes the creation of the necessary  
3940 Permission instances as well as the Privileges and Privilege attributes.

## 3941 **E.4 Content Submission – Client Responsibility**

3942 The Registry client has to sign the contents before submission – otherwise the content  
3943 will be rejected.

## 3944 **E.5 Content Submission – Registry Responsibility**

- 3945 1. Like any other request, the client will be first authenticated. In this case, the Principal  
3946 object will get the DN from the certificate.
- 3947 2. As per the request in the message, the RegistryEntry will be created.
- 3948 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 3949 4. If a principal with the identity of the SO is not available, an identity object with the  
3950 SO's DN is created
- 3951 5. A principal with this identity is created

## 3952 **E.6 Content Delete/Deprecate – Client Responsibility**

3953 The Registry client has to sign the payload (not entire message) before submission, for  
3954 authentication purposes; otherwise, the request will be rejected



## 3955 **E.7 Content Delete/Deprecate – Registry Responsibility**

- 3956 1. Like any other request, the client will be first authenticated. In this case, the Principal  
3957 object will get the DN from the certificate. As there will be a principal with this identity  
3958 in the Registry, the Principal object will get all the roles from that object
- 3959 2. As per the request in the message (delete or deprecate), the appropriate method in  
3960 the RegistryObject class will be accessed.
- 3961 3. The access controller performs the authorization by iterating through the Permission  
3962 objects associated with this object via the singleton default AccessControlPolicy.
- 3963 4. If authorization succeeds then the action will be permitted. Otherwise an error  
3964 response is sent back with a suitable AuthorizationException error message.

## 3965 **Appendix F Native Language Support (NLS)**

### 3966 **F.1 Definitions**

3967 Although this section discusses only character set and language, the following terms  
3968 have to be defined clearly.  
3969

#### 3970 **F.1.1 Coded Character Set (CCS):**

3971 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130].  
3972 Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.  
3973

#### 3974 **F.1.2 Character Encoding Scheme (CES):**

3975 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of  
3976 CES are ISO-2022, UTF-8.

#### 3977 **F.1.3 Character Set (charset):**

3978 charset is a set of rules for mapping from a sequence of octets to a sequence of  
3979 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-  
3980 KR.

3981  
3982 A list of registered character sets can be found at [IANA].

### 3983 **F.2 NLS And Request / Response Messages**

3984 For the accurate processing of data in both registry client and registry services, it is  
3985 essential to know which character set is used. Although the body part of the transaction

3986 may contain the charset in xml encoding declaration, registry client and registry services  
3987 shall specify charset parameter in MIME header when they use text/xml. Because as  
3988 defined in [RFC 3023], if a text/xml entity is received with the charset parameter  
3989 omitted, MIME processors and XML processors MUST use the default charset value of  
3990 "us-ascii".

3991  
3992 Ex. Content-Type: text/xml; charset=ISO-2022-JP

3993  
3994 Also, when an application/xml entity is used, the charset parameter is optional, and  
3995 registry client and registry services must follow the requirements in Section 4.3.3 of  
3996 [REC-XML] which directly address this contingency.

3997  
3998 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

### 3999 **F.3 NLS And Storing of RegistryEntry**

4000 This section provides NLS guidelines on how a registry should store **RegistryEntry**  
4001 instances.

#### 4002 **F.3.1 Character Set of RegistryEntry**

4003 This is basically an implementation issue because the actual character set that the  
4004 **RegistryEntry** is stored with, does not affect the interface. However, it is highly  
4005 recommended to use UTF-16 or UTF-8 for covering various languages.

#### 4006 **F.3.2 Language Information of RegistryEntry**

4007 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]). If the  
4008 xml:lang attribute is specified, then the registry may use that language code as the  
4009 value of a special Slot with name **language** and sloType of **nls** in the **RegistryEntry**.  
4010 The value must be compliant to [RFC 1766]. Slots are defined in [ebRIM].

### 4011 **F.4 NLS And Storing of Repository Items**

4012 This section provides NLS guidelines on how a registry should store repository items.

#### 4013 **F.4.1 Character Set of Repository Items**

4014 Unlike the character set of **RegistryEntry**, the charset of a repository item must be  
4015 preserved as it is originally specified in the transaction. The registry may use a special  
4016 Slot with name **repositoryItemCharset**, and sloType of **nls** for the **RegistryEntry** for  
4017 storing the charset of the corresponding repository item. Value must be the one defined  
4018 in [RFC 2277], [RFC 2278]. The **repositoryItemCharset** is optional because not all  
4019 repository items require it.

4020

4021 **F.4.2 Language information of repository item**

4022 Specifying only character set is not enough to tell which language is used in the  
 4023 repository item. A registry may use a special Slot with name **repositoryItemLang**, and  
 4024 slotType of **nls** to store that information. This attribute is optional because not all  
 4025 repository items require it. Value must be compliant to [RFC 1766]

4026  
 4027 This document currently specifies only the method of sending the information of  
 4028 character set and language, and how it is stored in a registry. However, the language  
 4029 information may be used as one of the query criteria, such as retrieving only DTD  
 4030 written in French. Furthermore, a language negotiation procedure, like registry client is  
 4031 asking a favorite language for messages from registry services, could be another  
 4032 functionality for the future revision of this document.

4033 **Appendix G Terminology Mapping**

4034 While every attempt has been made to use the same terminology used in other works  
 4035 there are some terminology differences.

4036 The following table shows the terminology mapping between this specification and that  
 4037 used in other specifications and working groups.

4038

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	
----------------------	--------------------	--

4039

**Table 1: Terminology Mapping Table**

4040

## 4040 **References**

- 4041 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 4042 [GLS] ebXML Glossary, [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)
- 4043 [TA] ebXML Technical Architecture
- 4044 [http://www.ebxml.org/specdrafts/ebXML\\_TA\\_v1.0.pdf](http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf)
- 4045 [OAS] OASIS Information Model
- 4046 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 4047 [ISO] ISO 11179 Information Model
- 4048 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 4049
- 4050 [ebRIM] ebXML Registry Information Model
- 4051 [http://www.ebxml.org/project\\_teams/registry/private/registryInfoModelv0.54.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf)
- 4052 [ebBPM] ebXML Business Process Specification Schema
- 4053 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 4054 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 4055 [http://www.ebxml.org/project\\_teams/trade\\_partner/private/](http://www.ebxml.org/project_teams/trade_partner/private/)
- 4056 [CTB] Context table informal document from Core Components
- 4057 [ebMS] ebXML Messaging Service Specification, Version 0.21
- 4058 [http://ebxml.org/project\\_teams/transport/private/ebXML\\_Messaging\\_Service\\_Specification\\_v0-21.pdf](http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf)
- 4059 [ERR] ebXML TRP Error Handling Specification
- 4060 [http://www.ebxml.org/project\\_teams/transport/ebXML\\_Message\\_Service\\_Specification\\_v-0.8\\_001110.pdf](http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf)
- 4061 [SEC] ebXML Security Specification
- 4062 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 4063 [XPT] XML Path Language (XPath) Version 1.0
- 4064 <http://www.w3.org/TR/xpath>
- 4065 [SQL] Structured Query Language (FIPS PUB 127-2)
- 4066 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 4067
- 4068 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 4069 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 4070
- 4071 [IANA] IANA (Internet Assigned Numbers Authority).
- 4072 Official Names for Character Sets, ed. Keld Simonsen et al.

- 4073 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
- 4074
- 4075 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:  
4076 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.  
4077 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 4078
- 4079 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:  
4080 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.  
4081 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 4082
- 4083 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:  
4084 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.  
4085 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 4086
- 4087 [RFC 2130] IETF (Internet Engineering Task Force). RFC 2130:  
4088 The Report of the IAB Character Set Workshop held 29 February - 1 March,  
4089 1996,  
4090 C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M. Crispin, P.  
4091 Svanberg. 1997.  
4092 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2130.html>
- 4093
- 4094 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:  
4095 XML Media Types, ed. M. Murata. 2001.  
4096 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
- 4097
- 4098 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second  
4099 Edition)  
4100 <http://www.w3.org/TR/REC-xml>
- 4101
- 4102 [UUID] DCE 128 bit Universal Unique Identifier  
4103 [http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh_20)  
4104 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>

## 4105 **Disclaimer**

4106 The views and specification expressed in this document are those of the authors and  
4107 are not necessarily those of their employers. The authors and their employers  
4108 specifically disclaim responsibility for any problems arising from correct or incorrect  
4109 implementation or use of this design.

4110

**4110 Contact Information**

## 4111 Team Leader

4112 Name: Scott Nieman  
4113 Company: Norstan Consulting  
4114 Street: 5101 Shady Oak Road  
4115 City, State, Postal Code: Minnetonka, MN 55343  
4116 Country: USA  
4117 Phone: 952.352.5889  
4118 Email: Scott.Nieman@Norstan

4119

## 4120 Vice Team Lead

4121 Name: Yutaka Yoshida  
4122 Company: Sun Microsystems  
4123 Street: 901 San Antonio Road, MS UMPK17-102  
4124 City, State, Postal Code: Palo Alto, CA 94303  
4125 Country: USA  
4126 Phone: 650.786.5488  
4127 Email: Yutaka.Yoshida@eng.sun.com

4128

## 4129 Editor

4130 Name: Farrukh S. Najmi  
4131 Company: Sun Microsystems  
4132 Street: 1 Network Dr., MS BUR02-302  
4133 City, State, Postal Code: Burlington, MA, 01803-0902  
4134 Country: USA  
4135 Phone: 781.442.0703  
4136 Email: najmi@east.sun.com

4137

4138

**4138 Copyright Statement**

4139 Copyright © ebXML 2001. All Rights Reserved.

4140

4141 This document and translations of it may be copied and furnished to others, and  
4142 derivative works that comment on or otherwise explain it or assist in its implementation  
4143 MAY be prepared, copied, published and distributed, in whole or in part, without  
4144 restriction of any kind, provided that the above copyright notice and this paragraph are  
4145 included on all such copies and derivative works. However, this document itself MAY  
4146 not be modified in any way, such as by removing the copyright notice or references to  
4147 ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other  
4148 than English.

4149

4150 The limited permissions granted above are perpetual and will not be revoked by ebXML  
4151 or its successors or assigns.

4152

4153 This document and the information contained herein is provided on an  
4154 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
4155 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
4156 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
4157 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
4158 PURPOSE.