



Creating A Single Global Electronic Market

1

2 **ebXML Registry Services**

3 **ebXML Registry Project Team**

4 **Working Draft 3/19/2001**

5 **This version: Version 0.88**

6

7 **1 Status of this Document**

8

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

11

12 Distribution of this document is unlimited.

13

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

16

17 ***This version:***

18 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.88.pdf

19

20 ***Latest version:***

21 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecification.pdf

22

23 ***Previous version:***

24 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.87.pdf

25

26

27 **2 ebXML participants**

28 The authors wish to acknowledge the support of the members of the Registry
29 Project Team who contributed ideas to this specification by the group's
30 discussion e-mail list, on conference calls and during face-to-face meetings.

31
32 Lisa Carnahan – NIST
33 Joe Dalman - Tie
34 Philippe DeSmedt - Viquity
35 Sally Fuger – AIAG
36 Len Gallagher - NIST
37 Steve Hanna - Sun Microsystems
38 Scott Hinkelman - IBM
39 Michael Kass, NIST
40 Jong.L Kim – Innodigital
41 Sangwon Lim, Korea Institute for Electronic Commerce
42 Bob Miller - GXS
43 Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan
44 Dale Moberg – Sterling Commerce
45 Ron Monzillo – Sun Microsystems
46 JP Morgenthal – eThink Systems, Inc.
47 Joel Munter - Intel
48 Farrukh Najmi - Sun Microsystems
49 Scott Nieman - Norstan Consulting
50 Frank Olken – Lawrence Berkeley National Laboratory
51 Michael Park - eSum Technologies
52 Bruce Peat - eProcess Solutions
53 Mike Rowley – Excelon Corporation
54 Waqar Sadiq - Vitria
55 Krishna Sankar – Cisco Systems Inc.
56 Kim Tae Soo - Government of Korea
57 Nikola Stojanovic - Encoda Systems, Inc.
58 David Webber – XML Global
59 Yutaka Yoshida - Sun Microsystems
60 Prasad Yendluri - webmethods
61 Peter Z. Zhoo - Knowledge For the new Millennium
62

62 **Table of Contents**

63 **1 Status of this Document 1**

64 **2 ebXML participants 2**

65 **Table of Contents..... 3**

66 **Table of Tables 7**

67 **3 Introduction 8**

68 3.1 Summary of Contents of Document8

69 3.2 General Conventions8

70 3.3 Audience.....8

71 3.4 Related Documents8

72 **4 Design Objectives..... 9**

73 4.1 Goals9

74 4.2 Caveats and Assumptions9

75 **5 System Overview 9**

76 5.1 What The ebXML Registry Does9

77 5.2 How The ebXML Registry Works 10

78 5.2.1 Schema Documents Are Submitted..... 10

79 5.2.2 Business Process Documents Are Submitted..... 10

80 5.2.3 Seller’s Collaboration Protocol Profile Is Submitted 10

81 5.2.4 Buyer Discovers The Seller 10

82 5.2.5 CPA Is Established 11

83 5.3 Where the Registry Services May Be Implemented..... 11

84 5.4 Implementation Conformance 11

85 5.4.1 Conformance as an ebXML Registry..... 11

86 5.4.2 Conformance as an ebXML Registry Client..... 11

87 **6 Registry Architecture..... 12**

88 6.1 Implicit CPA Between Clients And Registry..... 13

89 6.2 Client To Registry Communication Bootstrapping 14

90 6.3 Interfaces Exposed By The Registry..... 15

91 6.3.1 Interface *RegistryService*..... 15

92 6.3.2 Interface *ObjectManager* 16

93 6.3.3 Interface *ObjectQueryManager* 16

94 6.4 Interfaces Exposed By Registry Clients 18

95 6.4.1 Interface *RegistryClient*..... 18

96 6.4.2 Interface *ObjectManagerClient*..... 18

97 6.4.3 Interface *ObjectQueryManagerClient*..... 20

98 **7 Object Management Service 20**

99 7.1 Life Cycle of a Registry Entry 21

100	7.2	Object Attributes	21
101	7.3	The Submit Objects Protocol	22
102	7.3.1	Universally Unique ID Generation.....	22
103	7.3.2	ID Attribute And Object References	23
104	7.3.3	Sample SubmitObjectsRequest	23
105	7.4	The Add Slots Protocol	26
106	7.5	The Remove Slots Protocol.....	26
107	7.6	The Approve Objects Protocol.....	27
108	7.7	The Deprecate Objects Protocol	27
109	7.8	The Remove Objects Protocol.....	28
110	7.8.1	Deletion Scope DeleteRepositoryItemOnly	28
111	7.8.2	Deletion Scope DeleteAll	28
112	8	Object Query Management Service	29
113	8.1	Browse and Drill Down Query Support	30
114	8.1.1	Get Root Classification Nodes Request.....	30
115	8.1.2	Get Classification Tree Request	31
116	8.1.3	Get Classified Objects Request	32
117	8.1.3.1	Get Classified Objects Request Example	32
118	8.2	Filter Query Support	33
119	8.2.1	FilterQuery.....	34
120	8.2.2	RegistryEntryQuery	36
121	8.2.3	AuditableEventQuery.....	42
122	8.2.4	ClassificationNodeQuery	45
123	8.2.5	RegistryPackageQuery	48
124	8.2.6	OrganizationQuery.....	50
125	8.2.7	GetRegistryEntry.....	54
126	8.2.8	GetRepositoryItem	58
127	8.2.9	Registry Filters.....	63
128	8.2.10	XML Clause Constraint Representation.....	66
129	8.3	SQL Query Support	70
130	8.3.1	SQL Query Syntax Binding To [RIM]	70
131	8.3.1.1	Interface and Class Binding	70
132	8.3.1.2	Accessor Method To Attribute Binding	71
133	8.3.1.3	Primitive Attributes Binding	71
134	8.3.1.4	Reference Attribute Binding	71
135	8.3.1.5	Complex Attribute Binding	72
136	8.3.1.6	Collection Attribute Binding	72
137	8.3.2	Semantic Constraints On Query Syntax.....	72
138	8.3.3	SQL Query Results	73
139	8.3.4	Simple Metadata Based Queries.....	73
140	8.3.5	RegistryEntry Queries	73
141	8.3.6	Classification Queries.....	74
142	8.3.6.1	Identifying ClassificationNodes	74
143	8.3.6.2	Getting Root Classification Nodes.....	74

144	8.3.6.3	Getting Children of Specified ClassificationNode	74
145	8.3.6.4	Getting Objects Classified By a ClassificationNode	74
146	8.3.6.5	Getting ClassificationNodes That Classify an Object...	75
147	8.3.7	Association Queries	75
148	8.3.7.1	Getting All Association With Specified Object As Its	
149		Source	75
150	8.3.7.2	Getting All Association With Specified Object As Its	
151		Target	75
152	8.3.7.3	Getting Associated Objects Based On Association	
153		Attributes	75
154	8.3.7.4	Complex Association Queries	75
155	8.3.8	Package Queries	76
156	8.3.8.1	Complex Package Queries	76
157	8.3.9	ExternalLink Queries	76
158	8.3.9.1	Complex ExternalLink Queries	76
159	8.3.10	Audit Trail Queries	76
160	8.3.11	Content Based Ad Hoc Queries	76
161	8.3.11.1	Automatic Classification of XML Content	77
162	8.3.11.2	Index Definition	77
163	8.3.11.3	Example Of Index Definition	77
164	8.3.11.4	Example of Automatic Classification	78
165	8.3.12	Ad Hoc Query Request/Response	78
166	8.4	Content Retrieval	79
167	8.4.1	Retrieval of Registry Profile	79
168	8.4.2	Identification Of Content Payloads	80
169	8.4.3	GetContentResponse Message Structure	80
170	8.5	Query And Retrieval: Typical Sequence	81
171	9	Registry Security	81
172	9.1	Integrity of Registry Content	82
173	9.1.1	Message Payload Signature	82
174	9.2	Authentication	82
175	9.2.1	Message Header Signature	83
176	9.3	Confidentiality	83
177	9.3.1	On-the-wire Message Confidentiality	83
178	9.3.2	Confidentiality of Registry Content	83
179	9.4	Authorization	83
180	9.4.1	Pre-defined Roles For Registry Users	83
181	9.4.2	Default Access Control Policies	84
182	Appendix A	Schemas and DTD Definitions	85
183	A.1	ebXMLError Message DTD	85
184	A.2	ebXML Registry DTD	85
185	Appendix B	Interpretation of UML Diagrams	93
186	B.1	UML Class Diagram	93

187 B.2 UML Sequence Diagram..... 93

188 **Appendix C SQL Query 94**

189 C.1 SQL Query Syntax Specification 94

190 C.2 Non-Normative BNF for Query Syntax Grammar 95

191 C.3 Relational Schema For SQL Queries 96

192 **Appendix D Security Implementation Guideline..... 103**

193 D.1 Authentication 103

194 D.2 Authorization 103

195 D.3 Registry Bootstrap 103

196 D.4 Content Submission – Client Responsibility 103

197 D.5 Content Submission – Registry Responsibility..... 104

198 D.6 Content Delete/Deprecate – Client Responsibility..... 104

199 D.7 Content Delete/Deprecate – Registry Responsibility..... 104

200 **Appendix E Native Language Support (NLS)..... 104**

201 E.1 Definitions 104

202 E.1.1 Coded Character Set (CCS): 105

203 E.1.2 Character Encoding Scheme (CES): 105

204 E.1.3 Character Set (charset):..... 105

205 E.2 NLS And Request / Response Messages..... 105

206 E.3 NLS And Storing of RegistryEntry..... 105

207 E.3.1 Character Set of *RegistryEntry* 106

208 E.3.2 Language Information of *RegistryEntry* 106

209 E.4 NLS And Storing of Repository Items 106

210 E.4.1 Character Set of Repository Items 106

211 E.4.2 Language information of repository item 106

212 **Appendix F Terminology Mapping..... 107**

213 **10 References..... 107**

214 **11 Disclaimer 109**

215 **12 Contact Information 110**

216 **Copyright Statement..... 111**

217 **Table of Figures**

218 Figure 1: Registry Architecture Supports Flexible Topologies..... 13

219 Figure 2: ebXML Registry Interfaces 15

220 Figure 3: Life Cycle of a Registry Entry..... 21

221 Figure 4: Submit Objects Sequence Diagram 22

222 Figure 5: Add Slots Sequence Diagram 26

223 Figure 6: Remove Slots Sequence Diagram 26

224 Figure 7: Approve Objects Sequence Diagram..... 27

225 Figure 8: Deprecate Objects Sequence Diagram 28

226 Figure 9: Remove Objects Sequence Diagram..... 29

227 Figure 10: Get Root Classification Nodes Sequence Diagram 30

228 Figure 11: Get Root Classification Nodes Asynchronous Sequence Diagram 31

229 Figure 12: Get Classification Tree Sequence Diagram 31

230 Figure 13: Get Classification Tree Asynchronous Sequence Diagram 31

231 Figure 14: A Sample Geography Classification 32

232 Figure 15: Submit Ad Hoc Query Sequence Diagram 78

233 Figure 16: Submit Ad Hoc Query Asynchronous Sequence Diagram..... 79

234 Figure 17: Typical Query and Retrieval Sequence..... 81

235 **Table of Tables**

236 Table 1: Terminology Mapping Table 107

237

238

238 3 Introduction

239 3.1 Summary of Contents of Document

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

242 A separate document, *ebXML Registry Information Model* [RIM], provides
243 information on the types of metadata that is stored in the Registry as well as the
244 relationships among the various metadata classes.

245 3.2 General Conventions

246 o UML diagrams are used as a way to concisely describe concepts. They are
247 not intended to convey any specific implementation or methodology
248 requirements.

249 o The term "*repository item*" is used to refer to actual Registry content (e.g. a
250 DTD, as opposed to metadata about the DTD).

251 o The term "*RegistryEntry*" is used to refer to an object that provides metadata
252 about a content instance (*repository item*).

253 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,
254 SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in
255 this 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
258 developers who 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
263 the reader:

264 a) *ebXML Registry Business Domain Model* [BDM] - defines requirements
265 for ebXML Registry Services

266 b) *ebXML Registry Information Model* [RIM]- specifies the information model
267 for the ebXML Registry

268 c) *ebXML Messaging Service Specification* [MS]

269 d) *ebXML Business Process Specification Schema* [BPM]

- 270 e) *Collaboration Protocol Specification* [CPA] - defines how profiles can be
271 defined for a party and how two parties' profiles may be used to define a
272 party agreement

273

274 **4 Design Objectives**

275 **4.1 Goals**

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

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

282 **4.2 Caveats and Assumptions**

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

286 It is assumed that:

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

294 **5 System Overview**

295 **5.1 What The ebXML Registry Does**

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

301 **5.2 How The ebXML Registry Works**

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

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

313 **5.2.1 Schema Documents Are Submitted**

314 A third party such as an industry consortium or standards group can submit the
315 necessary schema documents required by the RosettaNet PIP3A4 Purchase
316 Order business protocol with the Registry using the Object Manager service of
317 the Registry described in section 7.3.

318 **5.2.2 Business Process Documents Are Submitted**

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

323 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

329 **5.2.4 Buyer Discovers The Seller**

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

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

336 **5.2.5 CPA Is Established**

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

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

343 **5.3 Where the Registry Services May Be Implemented**

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

347 **5.4 Implementation Conformance**

348 An implementation may claim conformance as an ebXML Registry, an ebXML
349 Registry Client or both.

350 **5.4.1 Conformance as an ebXML Registry**

351 An implementation claims conformance to this specification if it meets the
352 following conditions:

- 353 1. Conforms to *the ebXML Registry Information Model [RIM]*.
- 354 2. Supports the syntax and semantics of the Registry Interfaces and Security
355 Model.
- 356 3. Supports the defined ebXML Error Message DTD.
- 357 4. Supports the defined ebXML Registry DTD.
- 358 5. Optionally supports the syntax and semantics of Section 8.3, SQL Query
359 Support.

360 **5.4.2 Conformance as an ebXML Registry Client**

361 An implementation claims conformance to this specification, as an ebXML
362 Registry Client if it meets the following conditions:

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

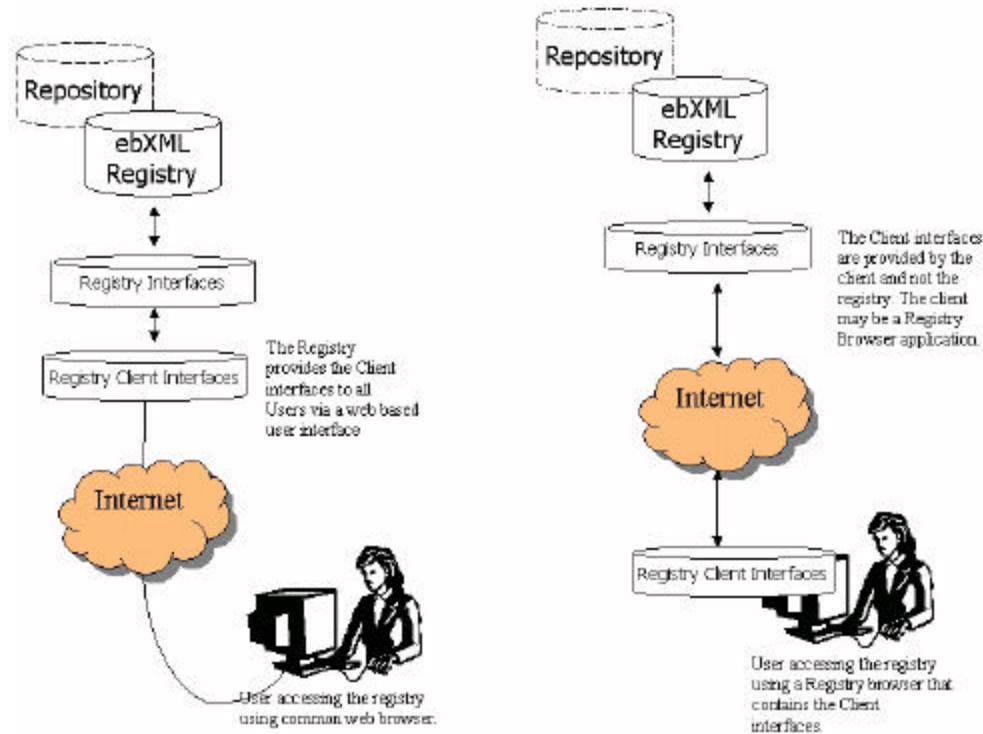
367 **6 Registry Architecture**

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

372 The picture on the left side shows the scenario where the Registry provides a
373 web based thin client application for accessing the Registry that is available to
374 the user using a common web browser. In this scenario the Registry Client
375 interfaces reside across the internet and local to the Registry from the user's
376 perspective.

377 The picture on the right side shows the scenario where the user is using a fat
378 client Registry Browser application to access the registry. In this scenario the
379 Registry Client interfaces reside within the Registry Browser tool and are local to
380 the Registry from the user's perspective. The Registry Client interfaces
381 communicate with the Registry over the internet in this scenario.

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



388

389

Figure 1: Registry Architecture Supports Flexible Topologies

390 Clients communicate with the Registry using the ebXML Messaging Service in
 391 the same manner as any two ebXML applications communicating with each
 392 other. Future versions of this specification may extend the Registry architecture
 393 to support distributed Registries.

394 This specification defines the interaction between a Registry client and the
 395 Registry. Although these interaction protocols are specific to the Registry, they
 396 are identical in nature to the interactions between two parties conducting B2B
 397 message communication using the ebXML Messaging Service as defined by
 398 [MS] and [CPA].

399 As such, these Registry specific interaction protocols are a special case of
 400 interactions between two parties using the ebXML Messaging Service.

401 **6.1 Implicit CPA Between Clients And Registry**

402 ebXML defines that a Collaboration Protocol Agreement [CPA] must exist
 403 between two parties in order for them to engage in B2B interactions.

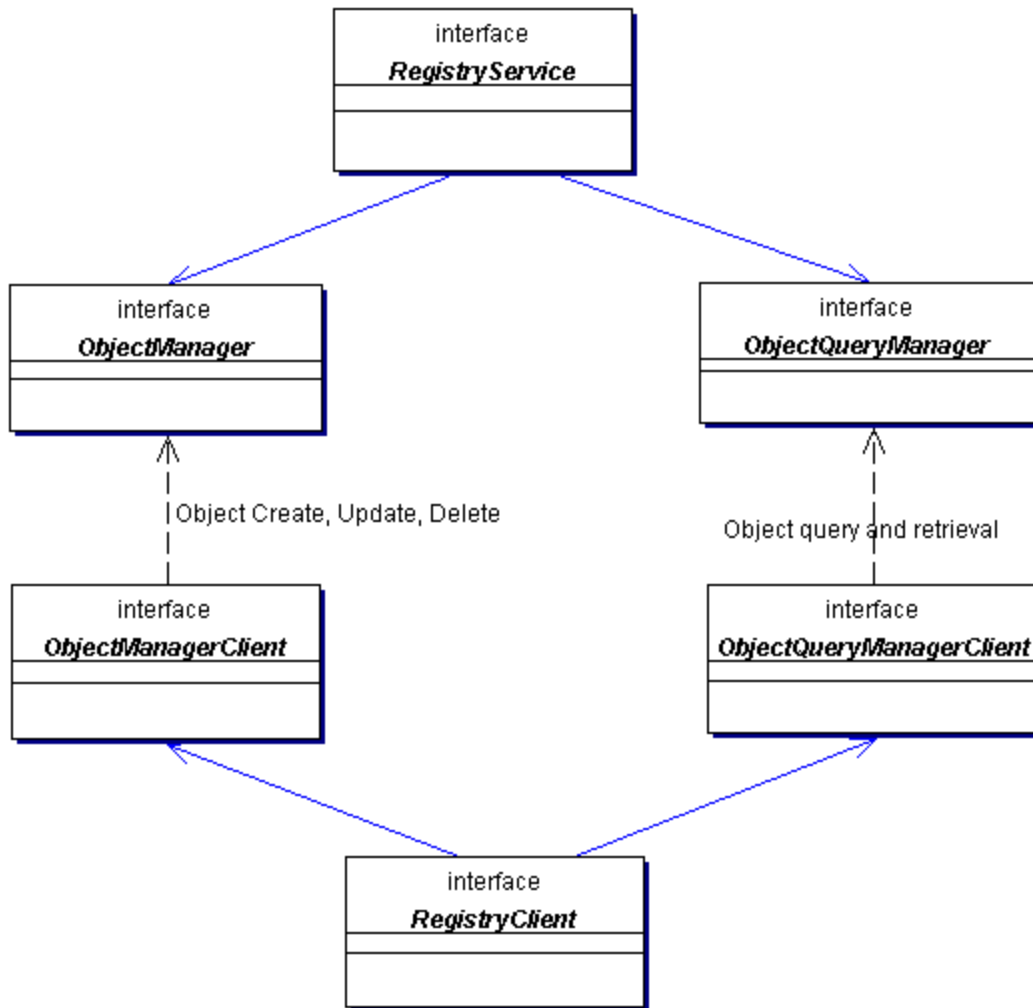
404 Similarly, this specification defines a CPA between a Registry client and the
405 Registry. Typical B2B interactions in ebXML require an explicit CPA to be
406 negotiated between parties. However, the CPA between clients and the Registry
407 is an implicit CPA that describes the interfaces that the Registry and the client
408 expose to each other for Registry specific interactions. These interfaces are
409 described in Figure 2 and subsequent sections.

410 **6.2 Client To Registry Communication Bootstrapping**

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

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

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



425

426

Figure 2: ebXML Registry Interfaces

427

6.3 Interfaces Exposed By The Registry

428

The ebXML Registry is shown to implement the following interfaces as its services (Registry Services).

429

430

6.3.1 Interface *RegistryService*

431

432

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

434

435

Method Summary	
<u>ObjectManager</u>	<u>getObjectManager()</u> Returns the ObjectManager interface implemented by the Registry service.
<u>ObjectQueryManager</u>	<u>getObjectQueryManager()</u> Returns the ObjectQueryManager interface implemented by the Registry service.

436

437 **6.3.2 Interface *ObjectManager***

438

439 This is the interface exposed by the Registry Service that implements the Object
 440 life cycle management functionality of the Registry. Its methods are invoked by
 441 the Registry Client. For example, the client may use this interface to submit
 442 objects, classify and associate objects and to deprecate and remove objects.

443

Method Summary	
Void	<u>approveObjects(ApproveObjectsRequest req)</u> Approves one or more previously submitted objects.
Void	<u>deprecateObjects(DeprecateObjectsRequest req)</u> Deprecates one or more previously submitted objects.
Void	<u>removeObjects(RemoveObjectsRequest req)</u> Removes one or more previously submitted objects from the Registry.
void	<u>submitObjects(SubmitObjectsRequest req)</u> Submits one or more objects and possibly metadata related to object such as Associations and Classifications.
void	<u>addSlots(AddSlotsRequest req)</u> Add slots to one or more registry entries.
void	<u>removeSlots(RemoveSlotsRequest req)</u> Remove specified slots from one or more registry entries.

444 **6.3.3 Interface *ObjectQueryManager***

445

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

450

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

AdhocQueryResponse	submitAdhocQuery (AdhocQueryRequest req) Submit an ad hoc query request.
void	submitAdhocQueryAsync (AdhocQueryRequest req) Async version of submitAdhocQuery.

451 **6.4 Interfaces Exposed By Registry Clients**

452 An ebXML Registry client is shown to implement the following interfaces.

453 **6.4.1 Interface *RegistryClient***

454 _____
 455 This is the principal interface implemented by a Registry client. The client
 456 provides this interface when creating a connection to the Registry. It provides the
 457 methods that are used by the Registry to discover service specific interfaces
 458 implemented by the client.

459

Method Summary	
ObjectManagerClient	getObjectManagerClient () Returns the ObjectManagerClient interface implemented by the client.
ObjectQueryManagerClient	getObjectQueryManagerClient () Returns the ObjectQueryManagerClient interface implemented by the client.

460

461 **6.4.2 Interface *ObjectManagerClient***

462 _____
 463 This is the client callback interface for the ObjectManager service of the Registry.
 464 The ObjectManager invokes its methods to notify the client about the results of a
 465 previously submitted request from the client to the ObjectManager service.

466

Method Summary	
void	addSlotsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted AddSlotsRequest was accepted by the Registry.

void	addSlotsError (ebXMLError error) Notifies client that a previously submitted AddSlotsRequest was not accepted by the Registry due to an error.
void	approveObjectsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.
void	approveObjectsError (ebXMLError error) Notifies client that a previously submitted ApproveObjectsRequest was not accepted by the Registry due to an error.
void	deprecateObjectsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted DeprecateObjectsRequest was accepted by the Registry.
void	deprecateObjectsError (ebXMLError error) Notifies client that a previously submitted DeprecateObjectsRequest was not accepted by the Registry due to an error.
void	removeObjectsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted RemoveObjectsRequest was accepted by the Registry.
void	removeSlotsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted RemoveSlotsRequest was accepted by the Registry.
void	removeObjectsError (ebXMLError error) Notifies client that a previously submitted RemoveObjectsRequest was not accepted by the Registry due to an error.
void	removeSlotsError (ebXMLError error) Notifies client that a previously submitted RemoveSlotsRequest was not accepted by the Registry due to an error.
void	submitObjectsAccepted (RequestAcceptedResponse resp) Notifies client that a previously submitted SubmitObjectsRequest was accepted by the Registry.
void	submitObjectsError (ebXMLError error) Notifies client that a previously submitted SubmitObjectsRequest was not accepted by the Registry due to an error.

468 **6.4.3 Interface *ObjectQueryManagerClient***

469

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

474

Method Summary	
void	getClassificationTreeAsyncResponse (GetClassificationTreeResponse resp) Async response for getClassificationTreeAsync request.
void	getClassifiedObjectsAsyncResponse (GetClassifiedObjectsResponse resp) Async response for getClassifiedObjectsAsync request.
void	getContentAsyncResponse (GetContentResponse resp) Async response for getContent request.
void	getRootClassificationNodesAsyncResponse (GetRootClassificationNodesResponse resp) Async response for getRootClassificationNodesAsync request.
void	submitAdhocQueryAsyncResponse (AdhocQueryResponse resp) Async response for submitAdhocQueryAsync request.

475 **7 Object Management Service**

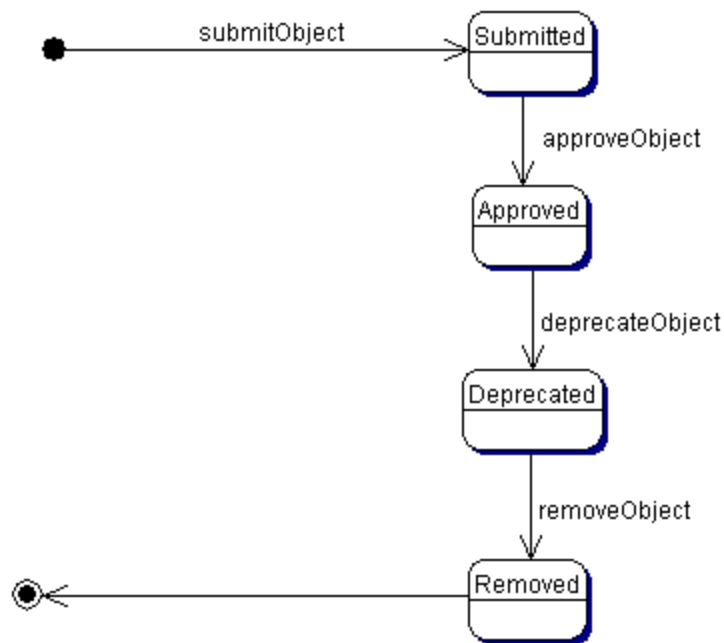
476 This section defines the Object Management service of the Registry. The Object
 477 Management Service is a sub-service of the Registry service. It provides the
 478 functionality required by RegistryClients to manage the life cycle of repository
 479 items (e.g. XML documents required for ebXML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

483 In the current version of this specification, any client may submit content as long
 484 as the content is digitally signed by a certificate issued by a Certificate Authority
 485 recognized by this registry. Submitting Organizations do not have to register prior
 486 to submitting content.

487 **7.1 Life Cycle of a Registry Entry**

488 The main purpose of the Object Management service is to manage the life cycle
 489 of repository items in the Registry.

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



493
 494

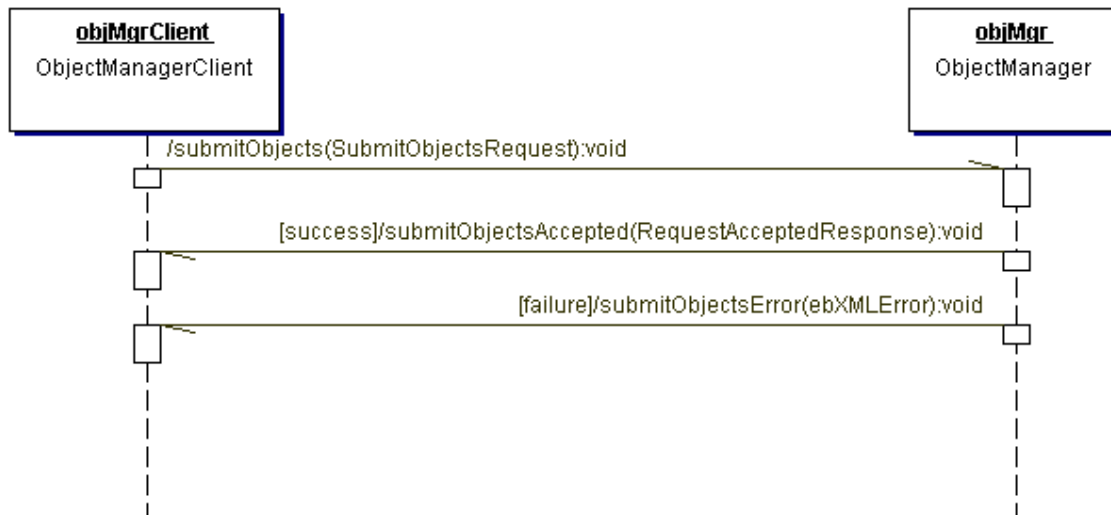
Figure 3: Life Cycle of a Registry Entry

495 **7.2 Object Attributes**

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

502 **7.3 The Submit Objects Protocol**

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



507

508

Figure 4: Submit Objects Sequence Diagram

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

511 The SubmitObjectRequest message includes 1 or more SubmittedObject
 512 elements.

513 Each SubmittedObject element specifies an ExtrinsicObject along with any
 514 Classifications, Associations, ExternalLinks, or Packages related to the object
 515 being submitted.

516 An ExtrinsicObject element provides required metadata about the content being
 517 submitted to the Registry as defined by [RIM]. Note that these standard
 518 ExtrinsicObject attributes are separate from the repository item itself, thus
 519 allowing the ebXML Registry to catalog arbitrary objects. In addition each
 520 SubmittedObject in the request may optionally specify any number of
 521 Classifications, Associations and ExternalLinks for the SubmittedObject.

522 **7.3.1 Universally Unique ID Generation**

523 As specified by [RIM], all objects in the registry have a unique id. This id is
 524 usually generated by the registry. The *id* attribute for various submitted objects
 525 may optionally be supplied by the client. If the client supplies the *id* and it
 526 conforms to the format of a URN that specifies a DCE 128 bit UUID

527 (e.g. urn:uuid:a2345678-1234-1234-123456789012)
528 then the registry assumes that the client wishes to specify the `id` for the object.
529 In this case, the registry must honor a client-supplied `id` and use it as the `id`
530 attribute of the object in the registry. If the `id` is found by the registry to not be
531 globally unique, the registry must send an `ebXML` Error in response with an
532 `InvalidIdError` message.

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

537 7.3.2 ID Attribute And Object References

538 The `id` attribute of an object may be used by other objects to reference the first
539 object. Such references are common both within the `SubmitObjectsRequest` as
540 well as within the registry. Within a `SubmitObjectsRequest`, the `id` attribute may
541 be used to refer to an object within the `SubmitObjectsRequest` as well as to refer
542 to an object within the registry. An object in the `SubmitObjectsRequest` that
543 needs to be referred to within the request document may be assigned an `id` by
544 the submitter so that it can be referenced within the request. The submitter may
545 give the object a proper `uuid` URN in which case the `id` is permanently assigned
546 to the object within the registry.

547 Alternatively, the submitter may assign an arbitrary `id` (not a proper `uuid` URN) as
548 long as the `id` is unique within the request document. In this case the `id` serves as
549 a linkage mechanism within the request document but must be ignored by the
550 registry and replaced with a registry generated `id` upon submission.

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

556 7.3.3 Sample SubmitObjectsRequest

557 The following example shows several different use cases in a single
558 `SubmitObjectRequest`. It does not show the complete `ebXML` Message with the
559 message header and additional payloads in the message for the repository items.

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

563
564

```
<?xml version = "1.0" encoding = "UTF-8"?>
```

```

565 <!DOCTYPE SubmitObjectsRequest SYSTEM "file:///home/najmi/Registry.dtd">
566
567 <SubmitObjectsRequest>
568   <RegistryEntryList>
569
570     <!--
571     The following 3 objects package specified ExtrinsicObject in specified
572     Package, where both the Package and the ExtrinsicObject are
573     being submitted
574     -->
575     <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
576     <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
577       objectType = "CPP" name = "Widget Profile"
578       description = "ACME's profile for selling widgets"/>
579     <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
580       sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>
581
582     <!--
583     The following 3 objects package specified ExtrinsicObject in specified Package,
584     Where the Package is being submitted and the ExtrinsicObject is
585     already in registry
586     -->
587     <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
588     <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
589     <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
590       associationType = "Packages" sourceObject = "acmePackage2"
591       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
592
593     <!--
594     The following 3 objects package specified ExtrinsicObject in specified Package,
595     where the Package and the ExtrinsicObject are already in registry
596     -->
597     <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
598     <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
599     <!-- id is unspecified implying that registry must create a uuid for this object -->
600     <Association associationType = "Packages"
601       sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
602       targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
603
604     <!--
605     The following 3 objects externally link specified ExtrinsicObject using
606     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
607     are being submitted
608     -->
609     <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
610     <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
611       name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
612     <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
613       sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
614
615     <!--
616     The following 2 objects externally link specified ExtrinsicObject using specified
617     ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
618     is already in registry. Note that the targetObject points to an ObjectRef in a
619     previous line
620     -->
621     <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
622     <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
623       associationType = "ExternallyLinks" sourceObject = "acmeLink2"
624       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
625
626     <!--
627     The following 2 objects externally identify specified ExtrinsicObject using specified
628     ExternalIdentifier, where the ExternalIdentifier is being submitted and the
629     ExtrinsicObject is already in registry. Note that the targetObject points to an
630     ObjectRef in a previous line
631     -->
632     <ExternalIdentifier id = "acmeDUNSID" name = "DUNS" description = "DUNS ID for ACME"

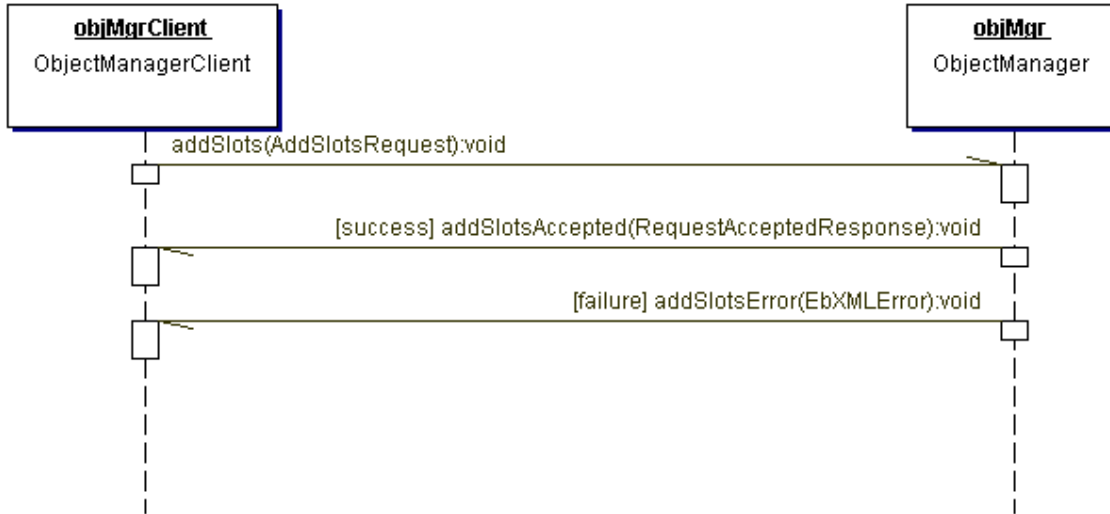
```



```
633     value = "13456789012"/>
634 <Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc"
635   associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSID"
636   targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
637
638 <!--
639 The following show submission of a brand new classification scheme in its entirety
640 -->
641 <ClassificationNode id = "geographyNode" name = "Geography"
642   description = "The Geography scheme example from Registry Services Spec" />
643 <ClassificationNode id = "asiaNode" name = "Asia"
644   description = "The Asia node under the Geography node" parent="geographyNode" />
645 <ClassificationNode id = "japanNode" name = "Japan"
646   description = "The Japan node under the Asia node" parent="asiaNode" />
647 <ClassificationNode id = "koreaNode" name = "Korea"
648   description = "The Korea node under the Asia node" parent="asiaNode" />
649 <ClassificationNode id = "europeNode" name = "Europe"
650   description = "The Europe node under the Geography node" parent="geographyNode" />
651 <ClassificationNode id = "germanyNode" name = "Germany"
652   description = "The Germany node under the Asia node" parent="europeNode" />
653 <ClassificationNode id = "northAmericaNode" name = "North America"
654   description = "The North America node under the Geography node"
655   parent="geographyNode" />
656 <ClassificationNode id = "usNode" name = "US"
657   description = "The US node under the Asia node" parent="asiaNode" />
658
659 <!--
660 The following show submission of a Automotive sub-tree of ClassificationNodes that
661 gets added to an existing classification scheme named 'Industry'
662 that is already in the registry
663 -->
664 <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
665 <ClassificationNode id = "automotiveNode" name = "Automotive"
666   description = "The Automotive sub-tree under Industry scheme"
667   parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
668 <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
669   description = "The Parts Supplier node under the Automotive node"
670   parent="automotiveNode" />
671 <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
672   description = "The Engine Supplier node under the Automotive node"
673   parent="automotiveNode" />
674
675 <!--
676 The following show submission of 2 Classifications of an object that is already in
677 the registry using 2 ClassificationNodes. One ClassificationNode
678 is being submitted in this request (Japan) while the other is already in the registry.
679 -->
680 <Classification id = "japanClassification"
681   description = "Classifies object by /Geography/Asia/Japan node"
682   classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
683   classificationNode="japanNode" />
684 <Classification id = "classificationUsingExistingNode"
685   description = "Classifies object using a node in the registry"
686   classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
687   classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
688 <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />
689
690
691 </RegistryEntryList>
692 </SubmitObjectsRequest>
```

693 **7.4 The Add Slots Protocol**

694 This section describes the protocol of the Registry Service that allows a client to
 695 add slots to a previously submitted registry entry using the Object Manager. Slots
 696 provide a dynamic mechanism for extending registry entries as defined by [RIM].



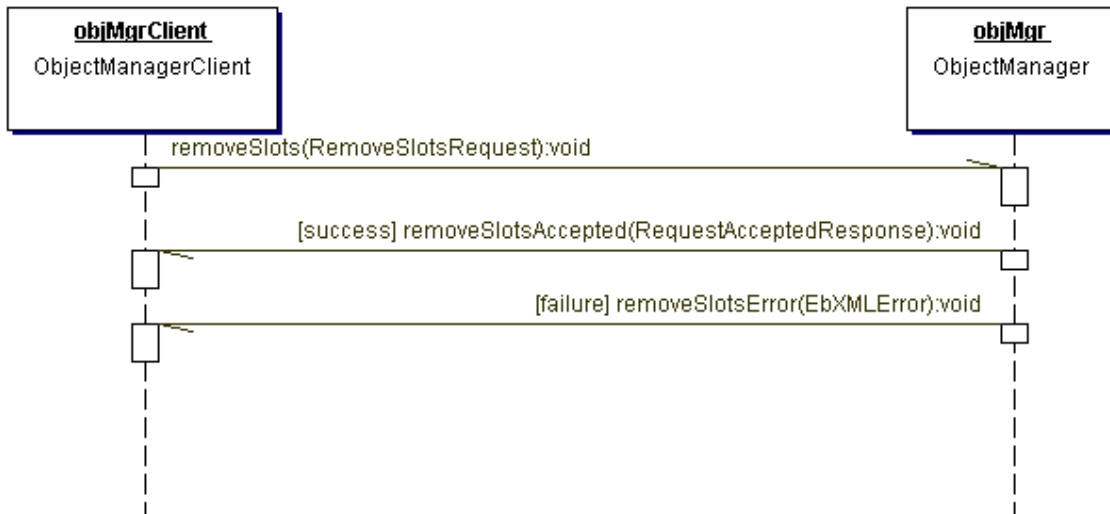
697

698 **Figure 5: Add Slots Sequence Diagram**

699

700 **7.5 The Remove Slots Protocol**

701 This section describes the protocol of the Registry Service that allows a client to
 702 add slots to a previously submitted registry entry using the Object Manager.

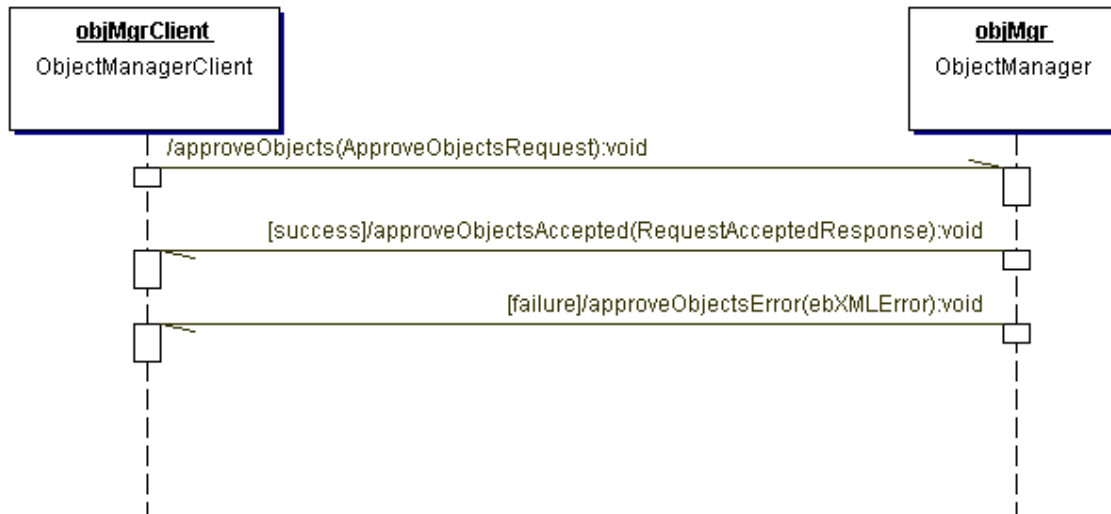


703

704 **Figure 6: Remove Slots Sequence Diagram**

705 **7.6 The Approve Objects Protocol**

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



711

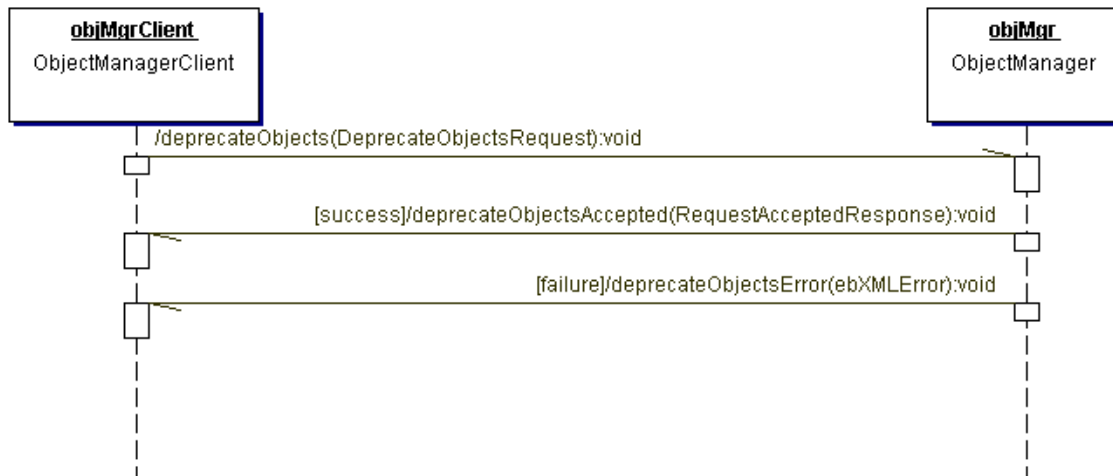
712

Figure 7: Approve Objects Sequence Diagram

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

715 **7.7 The Deprecate Objects Protocol**

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



722

723

Figure 8: Deprecate Objects Sequence Diagram

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

726 **7.8 The Remove Objects Protocol**

727 This section describes the protocol of the Registry Service that allows a client to
728 remove one or more Registry Entries and/or repository items using the Object
729 Manager.

730 The RemoveObjectsRequest message is sent by a client to remove Registry
731 Entries and/or repository items. The RemoveObjectsRequest element includes
732 an XML attribute called *deletionScope* which is an enumeration that can have the
733 values as defined by the following sections.

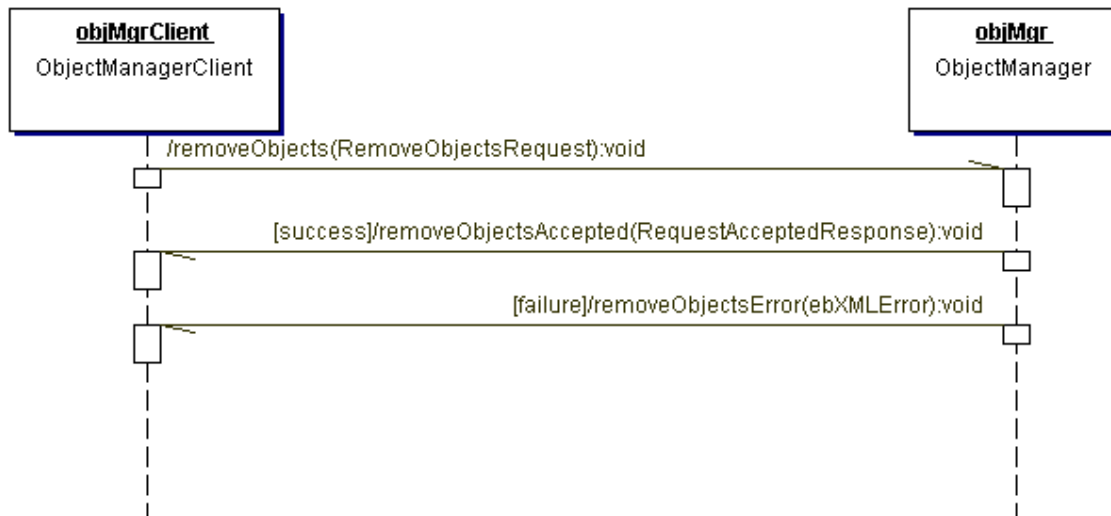
734 **7.8.1 Deletion Scope DeleteRepositoryItemOnly**

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

738 **7.8.2 Deletion Scope DeleteAll**

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

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



749

750

Figure 9: Remove Objects Sequence Diagram

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

753 **8 Object Query Management Service**

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

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

- 758 1. Browse and Drill Down Query
- 759 2. Filtered Query
- 760 3. SQL Query

761 The browse and drill down query [8.1] and the filtered query mechanism [8.2]
 762 shall be supported by every Registry implementation. The SQL query mechanism
 763 is an optional feature and may be provided by a registry implementation.

764 However, if a vendor provides an SQL query capability to an ebXML Registry
 765 they must conform to this document. As such it is this capability is a normative
 766 yet optional capability.

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

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

772 **8.1 Browse and Drill Down Query Support**

773 The browse and drill down query style is completely supported by a set of
 774 interaction protocols between the ObjectQueryManagerClient and the
 775 ObjectQueryManager as described next.

776 **8.1.1 Get Root Classification Nodes Request**

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



783
 784

Figure 10: Get Root Classification Nodes Sequence Diagram



785

786 **Figure 11: Get Root Classification Nodes Asynchronous Sequence Diagram**

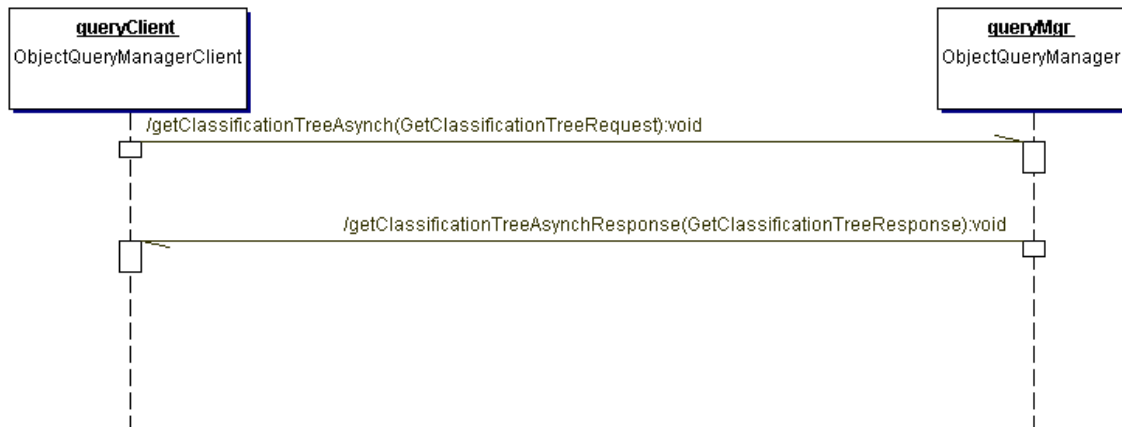
787 For details on the schema for the business documents shown in this process
 788 refer to Appendix A.2.

789 **8.1.2 Get Classification Tree Request**

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



797
 798 **Figure 12: Get Classification Tree Sequence Diagram**



799
 800 **Figure 13: Get Classification Tree Asynchronous Sequence Diagram**

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

803 **8.1.3 Get Classified Objects Request**

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

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

810 When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it
811 should return Objects that are:

- 812 1. Either directly classified by the specified ClassificationNode
- 813 2. Or are directly classified by a descendant of the specified
814 ClassificationNode

815 **8.1.3.1 Get Classified Objects Request Example**



816

817 Figure 14: A Sample Geography Classification

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

819 ?? If the Geography node is specified in the GetClassifiedObjectsRequest then
820 the GetClassifiedObjectsResponse should include all RegistryEntries that are
821 directly classified by Geography *or* North America *or* US *or* Asia *or* Japan *or*
822 Korea *or* Europe *or* Germany.

823 ?? If the Asia node is specified in the GetClassifiedObjectsRequest then the
824 GetClassifiedObjectsResponse should include all RegistryEntries that are
825 directly classified by Asia *or* Japan *or* Korea.

826 ?? If the Japan *and* Korea nodes are specified in the
827 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should
828 include all RegistryEntries that are directly classified by both Japan *and*
829 Korea.

830 ?? If the North America *and* Asia node is specified in the
831 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should
832 include all RegistryEntries that are directly classified by (North America *or*
833 US) *and* (Asia *or* Japan *or* Korea).

834

835 **8.2 Filter Query Support**

836 The simple XML FilterQuery specified below shall be supported by every
837 Registry implementation.

838 The FilterQuery syntax is tied to the structures defined in the Registry Information
839 Model [RIM] and is not intended to be extensible. If new structures are added to
840 the RIM, then the FilterQuery syntax and semantics can be extended at the same
841 time. Each query alternative requires a binding to the structures defined by RIM.

842 The Registry will hold a self-describing profile that identifies all supported Query
843 options. The RegistryProfile DTD is defined in appendix A.2. This profile can be
844 retrieved as defined by section 8.4.1.

845 An XML FilterQuery element provides alternatives to query selected classes from
846 the RIM. Each choice of a class pre-determines a virtual XML document that can
847 be queried as a tree. The RIM Binding paragraphs in Sections 8.2.2 through
848 8.2.6 below identify the virtual hierarchy for each query alternative. The Semantic
849 Rules for each query alternative specify the effect of that binding on query
850 semantics.

851 Each FilterQuery alternative depends upon one or more *registry filters*, where a
852 registry filter is a restricted *predicate clause* over the attributes of a single class.
853 The supported registry filters are specified in Section 8.2.9 and the supported
854 predicate clauses are defined in Section 8.2.10.

855 The GetRegistryEntry and GetRepositoryItem services defined below provide a
856 way to structure an XML document as an expansion of the result of a
857 RegistryEntryQuery. The GetRegistryEntry specified in Section 8.2.7 allows one
858 to specify what metadata one wants returned with each registry entry identified in
859 the result of a RegistryEntryQuery. The GetRepositoryItem specified in section
860 8.2.8 below allows one to specify what repository items one wants returned
861 based on their relationships to the registry entries identified in the result of a
862 RegistryEntryQuery.

863 A client submits a query to the ObjectQueryManager by sending an Adhoc
864 QueryRequest. The ObjectQueryManager sends an AdhocQueryResponse back
865 to the client. The request and the response for each query alternative, and the
866 sequence diagrams for AdhocQueryRequest and AdhocQueryResponse, are all
867 specified in section 8.3.12 below. A FilterQuery is one of the query options in an
868 AdhocQueryRequest and a FilterQueryResult is the response that is to be
869 returned as part of the AdhocQueryResponse.
870

870 **8.2.1 FilterQuery**871 **Purpose**

872 To identify a set of registry instances from a specific registry class. Each
 873 alternative assumes a specific binding to RIM. The query result for each query
 874 alternative is a set of references to instances of the root class specified by the
 875 binding. The StatusResult is a success indication or a collection of warnings
 876 and/or exceptions.

877 **Definition**

```

878
879 <!ELEMENT FilterQuery
880   (
881     | RegistryEntryQuery
882     | AuditableEventQuery
883     | ClassificationNodeQuery
884     | RegistryPackageQuery
885     | OrganizationQuery      )>
886
887 <!ELEMENT FilterQueryResult
888   (
889     | RegistryEntryQueryResult
890     | AuditableEventQueryResult
891     | ClassificationNodeQueryResult
892     | RegistryPackageQueryResult
893     | OrganizationQueryResult  )>
894
895 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
896
897 <!ELEMENT RegistryEntryView EMPTY >
898 <!ATTLIST RegistryEntryView
899   objectURN      CDATA      #REQUIRED
900   contentURL     CDATA      #IMPLIED
901   objectID       CDATA      #IMPLIED >
902
903 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
904
905 <!ELEMENT AuditableEventView EMPTY >
906 <!ATTLIST AuditableEventView
907   objectID       CDATA      #REQUIRED
908   timestamp      CDATA      #REQUIRED >
909
910 <!ELEMENT ClassificationNodeQueryResult
911   (ClassificationNodeView*)>
912
913 <!ELEMENT ClassificationNodeView EMPTY >
914 <!ATTLIST ClassificationNodeView
915   objectURN      CDATA      #REQUIRED
916   contentURL     CDATA      #IMPLIED
917   objectID       CDATA      #IMPLIED >
918
919 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>

```

```

919     <!ELEMENT RegistryPackageView EMPTY >
920     <!ATTLIST RegistryPackageView
921         objectURN      CDATA      #REQUIRED
922         contentURL     CDATA      #IMPLIED
923         objectID       CDATA      #IMPLIED >
924
925     <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
926
927     <!ELEMENT OrganizationView EMPTY >
928     <!ATTLIST OrganizationView
929         orgURN          CDATA      #REQUIRED
930         contactURL     CDATA      #IMPLIED
931         objectID       CDATA      #IMPLIED >
932
933     <!ELEMENT StatusResult ( Success | ( Exception | Warning )+ >
934
935         <!ELEMENT Success EMPTY >
936
937         <!ELEMENT Exception ( #PCDATA )>
938         <!ATTLIST Exception
939             code      CDATA      #REQUIRED >
940
941         <!ELEMENT Warning ( #PCDATA )>
942         <!ATTLIST Warning
943             code      CDATA      #REQUIRED >

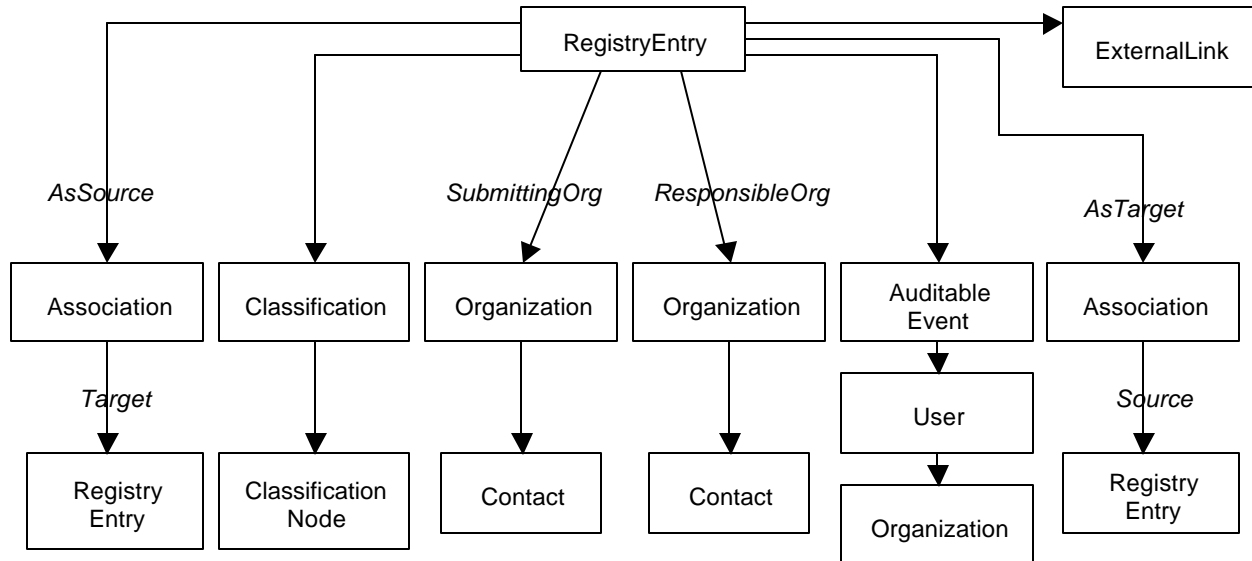
```

944 **Semantic Rules**

- 945 1. The semantic rules for each FilterQuery alternative are specified in
946 subsequent subsections.
- 947 2. Each FilterQueryResult is a set of XML reference elements to identify each
948 instance of the result set. Each XML attribute carries a value derived from the
949 value of an attribute specified in the Registry Information Model as follows:
 - 950 a) objectID is the value of the ID attribute of the Object class,
 - 951 b) objectURN and orgURN are URN values derived from the object ID,
 - 952 c) contentURL is a URL value derived from the contentURI attribute of the
953 RegistryEntry class,
 - 954 d) timestamp is a literal value to represent the value of the timestamp
955 attribute of the AuditableEvent class.
- 956 3. An Exception indicates that The FilterQuery was not successful, so the
957 FilterQueryResult is empty. A warning indicates that the FilterQuery was
958 successful, so the FilterQueryResult is accurate, but the warning may give
959 additional information back to the user.
- 960 4. If any exception or warning results, then it is returned as the appropriate
961 alternative of the StatusResult element. NOTE: This StatusResult may need
962 to be modified to fit more closely with the ebXML TRP specification.
963

963 **8.2.2 RegistryEntryQuery**

964 **Purpose**



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

967 **RIM Binding**

968

969 **Definition**

```

970
971 <!ELEMENT RegistryEntryQuery
972 ( RegistryEntryFilter?,
973   AsSourceAssociation*,
974   AsTargetAssociation*,
975   RegistryEntryClassification*,
976   SubmittingOrgFilter?,
977   ResponsibleOrgFilter?,
978   ExternalLinkFilter*,
979   RegistryEntryAuditableEvent* )>
980
981 <!ELEMENT AsSourceAssociation
982 ( AssociationFilter?,
983   RegistryEntryFilter? )>
984
985 <!ELEMENT AsTargetAssociation
986 ( AssociationFilter?,
987   RegistryEntryFilter? )>
988
989 <!ELEMENT RegistryEntryClassification
    
```

```
990     ( ClassificationFilter?,
991       ClassificationNodeFilter?     )>
992
993 <!ELEMENT SubmittingOrgFilter
994   ( OrganizationFilter?,
995     ContactFilter?                 )>
996
997 <!ELEMENT ResponsibleOrgFilter
998   ( OrganizationFilter?,
999     ContactFilter?                 )>
1000
1001 <!ELEMENT RegistryEntryAuditableEvent
1002   ( AuditableEventFilter?,
1003     UserFilter?,
1004     OrganizationFilter?           )>
```

1005 **Semantic Rules**

- 1006 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry.
1007 The following steps will eliminate instances in RE that do not satisfy the
1008 conditions of the specified filters.
 - 1009 a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue
1010 below; otherwise, let x be a registry entry in RE. If x does not satisfy the
1011 RegistryEntryFilter as defined in section 8.2.9, then remove x from RE.
 - 1012 b) If an AsSourceAssociation element is not specified, or if RE is empty, then
1013 continue below; otherwise, let x be a remaining registry entry in RE. If x is
1014 not the source object of some Association instance, then remove x from
1015 RE; otherwise, treat each AsSourceAssociation element separately as
1016 follows:
1017 If no AssociationFilter is specified within AsSourceAssociation, then let AF
1018 be the set of all Association instances that have x as a source object;
1019 otherwise, let AF be the set of Association instances that satisfy the
1020 AssociationFilter and have x as the source object. If AF is empty, then
1021 remove x from RE. If no RegistryEntryFilter is specified within
1022 AsSourceAssociation, then let RET be the set of all RegistryEntry
1023 instances that are the target object of some element of AF; otherwise, let
1024 RET be the set of RegistryEntry instances that satisfy the
1025 RegistryEntryFilter and are the target object of some element of AF. If
1026 RET is empty, then remove x from RE.
 - 1027 c) If an AsTargetAssociation element is not specified, or if RE is empty, then
1028 continue below; otherwise, let x be a remaining registry entry in RE. If x is
1029 not the target object of some Association instance, then remove x from
1030 RE; otherwise, treat each AsTargetAssociation element separately as
1031 follows:

- 1032 If no AssociationFilter is specified within AsTargetAssociation, then let AF
1033 be the set of all Association instances that have x as a target object;
1034 otherwise, let AF be the set of Association instances that satisfy the
1035 AssociationFilter and have x as the target object. If AF is empty, then
1036 remove x from RE. If no RegistryEntryFilter is specified within
1037 AsTargetAssociation, then let RES be the set of all RegistryEntry
1038 instances that are the source object of some element of AF; otherwise, let
1039 RES be the set of RegistryEntry instances that satisfy the
1040 RegistryEntryFilter and are the source object of some element of AF. If
1041 RES is empty, then remove x from RE.
- 1042 d) If a RegistryEntryClassification element is not specified, or if RE is empty,
1043 then continue below; otherwise, let x be a remaining registry entry in RE. If
1044 x is not the source object of some Classification instance, then remove x
1045 from RE; otherwise, treat each RegistryEntryClassification element
1046 separately as follows:
- 1047 If no ClassificationFilter is specified within the RegistryEntryClassification,
1048 then let CL be the set of all Classification instances that have x as a
1049 source object; otherwise, let CL be the set of Classification instances that
1050 satisfy the ClassificationFilter and have x as the source object. If CL is
1051 empty, then remove x from RE. If no ClassificationNodeFilter is specified
1052 within RegistryEntryClassification, then let CN be the set of all
1053 ClassificationNode instances that are the target object of some element of
1054 CL; otherwise, let CN be the set of RegistryEntry instances that satisfy the
1055 ClassificationNodeFilter and are the target object of some element of CL.
1056 If CN is empty, then remove x from RE.
- 1057 e) If a SubmittingOrgFilter element is not specified, or if RE is empty, then
1058 continue below; otherwise, let x be a remaining registry entry in RE. If x
1059 does not have a submitting organization, then remove x from RE. If no
1060 OrganizationFilter is specified within SubmittingOrgFilter, then let SO be
1061 the set of all Organization instances that are the submitting organization
1062 for x; otherwise, let SO be the set of Organization instances that satisfy
1063 the OrganizationFilter and are the submitting organization for x. If SO is
1064 empty, then remove x from RE. If no ContactFilter is specified within
1065 SubmittingOrgFilter, then let CT be the set of all Contact instances that
1066 are the contacts for some element of SO; otherwise, let CT be the set of
1067 Contact instances that satisfy the ContactFilter and are the contacts for
1068 some element of SO. If CT is empty, then remove x from RE.

- 1069 f) If a ResponsibleOrgFilter element is not specified, or if RE is empty, then
1070 continue below; otherwise, let x be a remaining registry entry in RE. If x
1071 does not have a responsible organization, then remove x from RE. If no
1072 OrganizationFilter is specified within ResponsibleOrgFilter, then let RO be
1073 the set of all Organization instances that are the responsible organization
1074 for x; otherwise, let RO be the set of Organization instances that satisfy
1075 the OrganizationFilter and are the responsible organization for x. If RO is
1076 empty, then remove x from RE. If no ContactFilter is specified within
1077 SubmittingOrgFilter, then let CT be the set of all Contact instances that
1078 are the contacts for some element of RO; otherwise, let CT be the set of
1079 Contact instances that satisfy the ContactFilter and are the contacts for
1080 some element of RO. If CT is empty, then remove x from RE.
- 1081 g) If an ExternalLinkFilter element is not specified, or if RE is empty, then
1082 continue below; otherwise, let x be a remaining registry entry in RE. If x is
1083 not linked to some ExternalLink instance, then remove x from RE;
1084 otherwise, treat each ExternalLinkFilter element separately as follows:
1085 Let EL be the set of ExternalLink instances that satisfy the
1086 ExternalLinkFilter and are linked to x. If EL is empty, then remove x from
1087 RE.
- 1088 h) If a RegistryEntryAuditableEvent element is not specified, or if RE is
1089 empty, then continue below; otherwise, let x be a remaining registry entry
1090 in RE. If x is not linked to some AuditableEvent instance, then remove x
1091 from RE; otherwise, treat each RegistryEntryAuditableEvent element
1092 separately as follows:
1093 If an AuditableEventsFilter is not specified within
1094 RegistryEntryAuditableEvent, then let AE be the set of all AuditableEvent
1095 instances for x; otherwise, let AE be the set of AuditableEvent instances
1096 that satisfy the AuditableEventFilter and are auditable events for x. If AE is
1097 empty, then remove x from RE. If an UserFilter is not specified within
1098 RegistryEntryAuditableEvent, then let AI be the set of all User instances
1099 linked to an element of AE; otherwise, let AI be the set of User instances
1100 that satisfy the UserFilter and are linked to an element of AE. If AI is
1101 empty, then remove x from RE. If an OrganizationFilter is not specified
1102 within RegistryEntryAuditableEvent, then let OG be the set of all
1103 Organization instances that are linked to an element of AI; otherwise, let
1104 OG be the set of Organization instances that satisfy the OrganizationFilter
1105 and are linked to an element of AI. If OG is empty, then remove x from
1106 RE.
- 1107 2. If RE is empty, then raise the warning: *registry entry query result is empty*;
1108 otherwise, return RE as the result of the RegistryEntryQuery.
- 1109 3. Return any accumulated warnings or exceptions as the StatusResult
1110 associated with the RegistryEntryQuery.

1111 **Examples**

1112 A client wants to establish a trading relationship with XYZ Corporation and wants
1113 to know if they have registered any of their business documents in the Registry.
1114 The following query returns a set of registry entry identifiers for currently
1115 registered items submitted by any organization whose name includes the string
1116 "XYZ". It does not return any registry entry identifiers for superceded, replaced,
1117 deprecated, or withdrawn items.

```
1118     <RegistryEntryQuery>
1119       <RegistryEntryFilter>
1120         Status EQ "Registered"           -- code by Clause, Section 8.2.10
1121       </RegistryEntryFilter>
1122       <SubmittingOrgFilter>
1123         <OrganizationFilter>
1124           Name CONTAINS "XYZ"           -- code by Clause, Section 8.2.10
1125         </Organizationfilter>
1126       </SubmittingOrgFilter>
1127     </RegistryEntryquery>
1128
1129
```

1130 A client is using the UNSPSC classification scheme and wants to identify all
1131 companies that deal with products classified as "Integrated circuit components",
1132 i.e. UNSPSC code "321118". The client knows that companies have registered
1133 their PartyProfile documents in the Registry, and that each profile has been
1134 classified by the products the company deals with. The following query returns a
1135 set of registry entry identifiers for profiles of companies that deal with integrated
1136 circuit components.

```
1137
1138     <RegistryEntryQuery>
1139       <RegistryEntryFilter>
1140         ObjectType EQ "PartyProfile" AND   -- code by Clause, Section 8.2.10
1141         Status EQ "Registered"
1142       </RegistryEntryFilter>
1143       <RegistryEntryClassification>
1144         <ClassificationNodeFilter>
1145           ID STARTSWITH "urn:un:spsc:321118" -- code by Clause, Section 8.2.10
1146         </ClassificationNodeFilter>
1147       </RegistryEntryClassification>
1148     </RegistryEntryQuery>
1149
```

1150 A client application needs all items that are classified by two different
1151 classification schemes, one based on "Industry" and another based on
1152 "Geography". Both schemes have been defined by ebXML and are registered.
1153 The root nodes of each scheme are identified by "urn:ebxml:cs:industry" and
1154 "urn:ebxml:cs:geography", respectively. The following query identifies registry
1155 entries for all registered items that are classified by "Industry/Automotive" and by
1156 "Geography/Asia/Japan".

1157


```

1158     <RegistryEntryQuery>
1159         <RegistryEntryClassification>
1160             <ClassificationNodeFilter>
1161                 ID STARTSWITH "urn:ebxml:cs:industry" AND
1162                 Path EQ "Industry/Automotive"      -- code by Clause, Section 8.2.10
1163             </ClassificationNodeFilter>
1164             <ClassificationNodeFilter>
1165                 ID STARTSWITH "urn:ebxml:cs:geography" AND
1166                 Path EQ "Geography/Asia/Japan"    -- code by Clause, Section 8.2.10
1167             </ClassificationNodeFilter>
1168         </RegistryEntryClassification>
1169     </RegistryEntryQuery>

```

1170

1171 A client application wishes to identify all registry Package instances that have a
 1172 given registry entry as a member of the package. The following query identifies
 1173 all registry packages that contain the registry entry identified by URN
 1174 "urn:path:myitem" as a member:

```

1175     <RegistryEntryQuery>
1176         <RegistryEntryFilter>
1177             objectType EQ "RegistryPackage"        -- code by Clause, Section 8.2.10
1178         </RegistryEntryFilter>
1179         <AsSourceAssociation>
1180             <AssociationFilter>                  -- code by Clause, Section 8.2.10
1181                 AssociationType EQ "HasMember" AND
1182                 TargetObject EQ "urn:path:myitem"
1183             </AssociationFilter>
1184         </AsSourceAssociation>
1185     </RegistryEntryQuery>

```

1187

1188 A client application wishes to identify all ClassificationNode instances that have
 1189 some given keyword as part of their name or description. The following query
 1190 identifies all registry classification nodes that contain the keyword "transistor" as
 1191 part of their name or as part of their description.

```

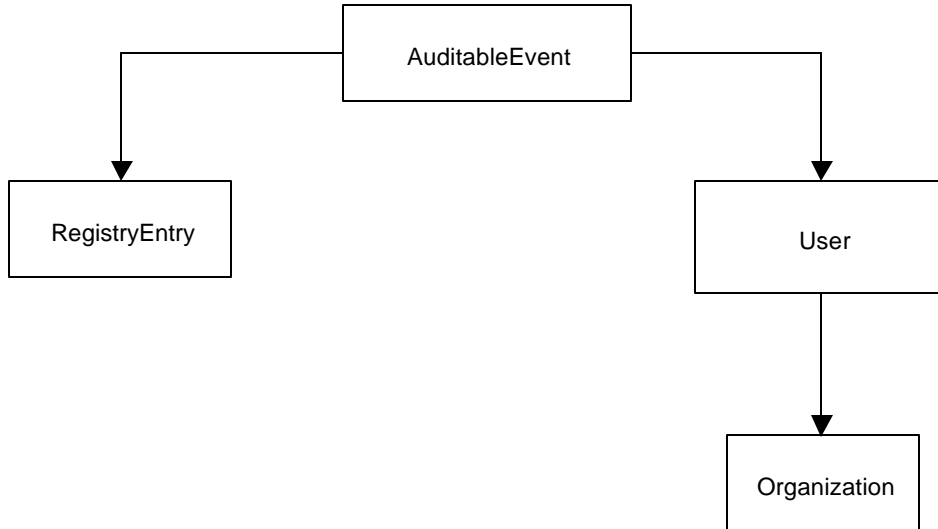
1192     <RegistryEntryQuery>
1193         <RegistryEntryFilter>
1194             ObjectType="ClassificationNode" AND
1195             (Name CONTAINS "transistor" OR      -- code by Clause, Section 8.2.10
1196              Description CONTAINS "transistor")
1197         </RegistryEntryFilter>
1198     </RegistryEntryQuery>

```

1200

1200 **8.2.3 AuditableEventQuery**

1201 **Purpose**



1202 To identify a set of auditable event instances as the result of a query over
 1203 selected registry metadata.

1204 **RIM Binding**

1205

1206

1207 **Definition**

1208

```

1209 <!ELEMENT AuditableEventQuery
1210 ( AuditableEventFilter?,
1211 RegistryEntryQuery*,
1212 UserFilter?,
1213 OrganizationQuery? )>
    
```

1214

1215 **Semantic Rules**

- 1216 1. Let AE denote the set of all persistent AuditableEvent instances in the
 1217 Registry. The following steps will eliminate instances in AE that do not satisfy
 1218 the conditions of the specified filters.

1219

- 1220 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue
1221 below; otherwise, let x be an auditable event in AE. If x does not satisfy
1222 the AuditableEventFilter as defined in section 8.2.9, then remove x from
1223 AE.
- 1224 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then
1225 continue below; otherwise, let x be a remaining auditable event in AE.
1226 Treat each RegistryEntryQuery element separately as follows:
- 1227 Let RE be the result set of the RegistryEntryQuery as defined in section
1228 8.2.2. If x is not an auditable event for some registry entry in RE, then
1229 remove x from AE.
- 1230 c) If a UserFilter element is not specified, or if AE is empty, then continue
1231 below; otherwise, let x be a remaining auditable event in AE. Let AI be the
1232 set of all User instances that satisfy the UserFilter and are linked to x as
1233 their auditable event. If AI is empty, then remove x from AE.
- 1234 d) If an OrganizationQuery element is not specified, or if RE is empty, then
1235 continue below; otherwise, let x be a remaining registry entry in RE. If an
1236 UserFilter element is not specified, then let AI be the set of all User
1237 instances that are linked to x as their auditable event; otherwise, let AI be
1238 the set of all User instances that satisfy the UserFilter and are linked to x
1239 as their auditable event. Let OG be the set of Organization instances that
1240 are the organization of an element in AI and are in the result set of the
1241 OrganizationQuery. If OG is empty, then remove x from AE.
- 1242 2. If AE is empty, then raise the warning: *auditable event query result is empty*.
- 1243 3. Return AE as the result of the AuditableEventQuery.
- 1244 4. Return any accumulated warnings or exceptions as the StatusResult
1245 associated with the AuditableEventQuery.

1246 Examples

1247 A Registry client has registered an item and it has been assigned a URN
1248 identifier "urn:path:myitem". The client is now interested in all events in the past
1249 year that have impacted that item. The following query will return a set of
1250 AuditableEvent identifiers for all such events.

```
1251 <AuditableEventquery>  
1252   <AuditableEventFilter>  
1253     Timestamp GE "2001-01-01" AND           -- code by Clause, Section 8.2.10  
1254     RegistryEntry EQ "urn:path:myitem"  
1255   </AuditableEventFilter>  
1256 </AuditableEventQuery>
```

1258

1259 A client company has many registered objects in the Registry. The Registry
1260 allows events submitted by other organizations to have an impact on your
1261 registered items, e.g. new classifications and new associations. The following
1262 query will return a set of identifiers for all events that have an impact on an item
1263 that you submitted, and you are responsible for, but the event was initiated by
1264 some other party.

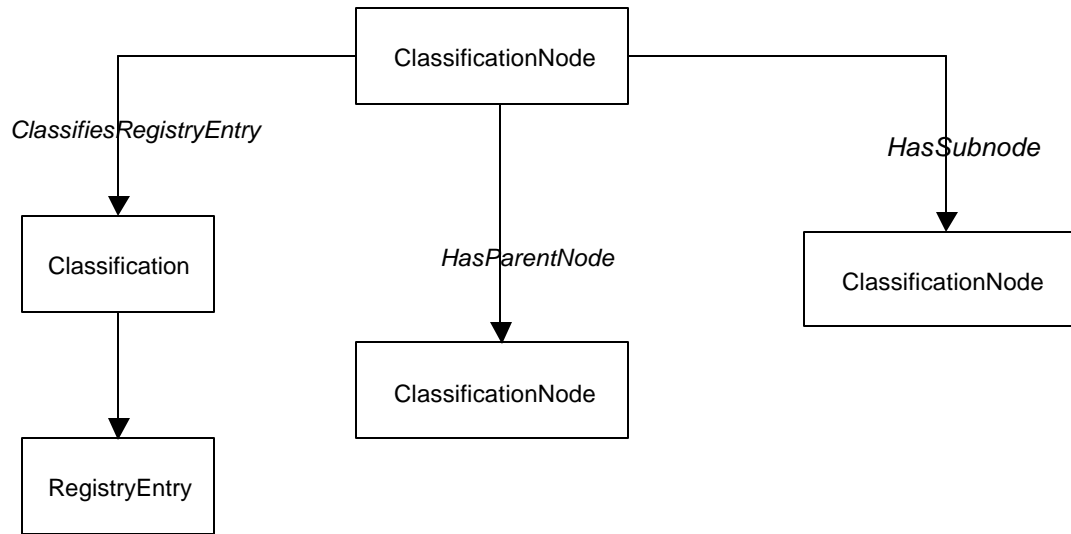
```
1265
1266     <AuditableEventquery>
1267         <RegistryEntryQuery>
1268             <SubmittingOrgFilter>
1269                 <OrganizationFilter>
1270                     ID EQ "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1271                 </OrganizationFilter>
1272             </SubmittingOrgFilter>
1273             <ResponsibleOrgFilter>
1274                 <OrganizationFilter>
1275                     ID EQ "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1276                 </OrganizationFilter>
1277             </ResponsibleOrgFilter>
1278         </RegistryEntryQuery>
1279         <UserFilter>
1280             <OrganizationFilter>
1281                 ID NE "urn:somepath:myorg"               -- code by Clause, Section 8.2.10
1282             </OrganizationFilter>
1283         </UserFilter>
1284     </AuditableEventQuery>
1285
```

1285 **8.2.4 ClassificationNodeQuery**

1286 **Purpose**

1287 To identify a set of classification node instances as the result of a query over
 1288 selected registry metadata.

1289 **RIM Binding**



1290 **Definition**

```

1291
1292 <!ELEMENT ClassificationNodeQuery
1293   ( ClassificationNodeFilter?,
1294     ClassifiesRegistryEntry*,
1295     HasParentNode?,
1296     HasSubnode*           )>
1297
1298 <!ELEMENT ClassifiesRegistryEntry
1299   ( ClassificationFilter?,
1300     RegistryEntryQuery?   )>
1301
1302 <!ELEMENT HasParentNode
1303   ( ClassificationNodeFilter?,
1304     HasParentNode?       )>
1305
1306 <!ELEMENT HasSubnode
1307   ( ClassificationNodeFilter?,
1308     HasSubnode*          )>
    
```

1309

1310 **Semantic Rules**

- 1311 1. Let CN denote the set of all persistent ClassificationNode instances in the
1312 Registry. The following steps will eliminate instances in CN that do not satisfy
1313 the conditions of the specified filters.
- 1314 a) If a ClassificationNodeFilter is not specified, or if CN is empty, then
1315 continue below; otherwise, let x be a classification node in CN. If x does
1316 not satisfy the ClassificationNodeFilter as defined in section 8.2.9, then
1317 remove x from AE.
- 1318 b) If a ClassifiesRegistryEntry element is not specified, or if CN is empty,
1319 then continue below; otherwise, let x be a remaining classification node in
1320 CN. If x is not the target object of some Classification instance, then
1321 remove x from CN; otherwise, treat each ClassifiesRegistryEntry element
1322 separately as follows:
- 1323 If no ClassificationFilter is specified within the ClassifiesRegistryEntry
1324 element, then let CL be the set of all Classification instances that have x
1325 as the target object; otherwise, let CL be the set of Classification instances
1326 that satisfy the ClassificationFilter and have x as the target object. If CL is
1327 empty, then remove x from CN. If no RegistryEntryQuery is specified
1328 within the ClassifiesRegistryEntry element, then let RES be the set of all
1329 RegistryEntry instances that are the source object of some classification
1330 instance in CL; otherwise, let RE be the result set of the
1331 RegistryEntryQuery as defined in section 8.2.2 and let RES be the set of
1332 all instances in RE that are the source object of some classification in CL.
1333 If RES is empty, then remove x from CN.
- 1334 c) If a HasParentNode element is not specified, or if CN is empty, then
1335 continue below; otherwise, let x be a remaining classification node in CN
1336 and execute the following paragraph with $n=x$.
- 1337 Let n be a classification node instance. If n does not have a parent node
1338 (i.e. if n is a root node), then remove x from CN. Let p be the parent node
1339 of n. If a ClassificationNodeFilter element is directly contained in
1340 HasParentNode and if p does not satisfy the ClassificationNodeFilter, then
1341 remove x from CN.
- 1342 If another HasParentNode element is directly contained within this
1343 HasParentNode element, then repeat the previous paragraph with $n=p$.
- 1344 d) If a HasSubnode element is not specified, or if CN is empty, then continue
1345 below; otherwise, let x be a remaining classification node in CN. If x is not
1346 the parent node of some ClassificationNode instance, then remove x from
1347 CN; otherwise, treat each HasSubnode element separately and execute
1348 the following paragraph with $n = x$.

1349 Let n be a classification node instance. If a `ClassificationNodeFilter` is not
 1350 specified within the `HasSubnode` element then let `CNC` be the set of all
 1351 classification nodes that have n as their parent node; otherwise, let `CNC`
 1352 be the set of all classification nodes that satisfy the
 1353 `ClassificationNodeFilter` and have n as their parent node. If `CNC` is empty
 1354 then remove x from `CN`; otherwise, let y be an element of `CNC` and
 1355 continue with the next paragraph.

1356 If the `HasSubnode` element is terminal, i.e. if it does not directly contain
 1357 another `HasSubnode` element, then continue below; otherwise, repeat the
 1358 previous paragraph with the new `HasSubnode` element and with $n = y$.

1359 2. If `CN` is empty, then raise the warning: *classification node query result is*
 1360 *empty*.

1361 3. Return `CN` as the result of the `ClassificationNodeQuery`.

1362 4. Return any accumulated warnings or exceptions as the `StatusResult`
 1363 associated with the `ClassificationNodeQuery`.

1364 Examples

1365 A client application wishes to identify all classification nodes defined in the
 1366 Registry that are root nodes and have a name that contains the phrase “product
 1367 code” or the phrase “product type”. Note: By convention, if a classification node
 1368 has no parent (i.e. is a root node), then the parent attribute of that instance is set
 1369 to null and is represented as a literal by a zero length string.
 1370

```
1371 <ClassificationNodeQuery>
1372   <ClassificationNodeFilter>
1373     (name CONTAINS "product code" OR      -- code by Clause, Section 8.2.10
1374      name CONTAINS "product type") AND
1375     parent EQ ""
1376   </ClassificationNodeFilter>
1377 </ClassificationNodeQuery>
```

1378

1379 A client application wishes to identify all of the classification nodes at the third
 1380 level of a classification scheme hierarchy. The client knows that the URN
 1381 identifier for the root node is `urn:ebxml:cs:myroot`. The following query identifies
 1382 all nodes at the second level under “myroot” (i.e. third level overall).
 1383

```
1384 <ClassificationNodeQuery>
1385   <HasParentNode>
1386     <HasParentNode>
1387       <ClassificationNodeFilter>
1388         ID EQ "urn:ebxml:cs:myroot"  -- code by Clause, Section 8.2.10
1389       </ClassificationNodeFilter>
1390     </HasParentNode>
1391   </HasParentNode>
1392 </ClassificationNodeQuery>
```

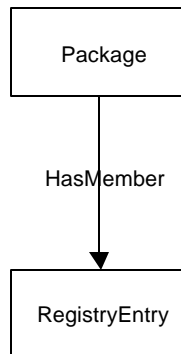
1393

1393 **8.2.5 RegistryPackageQuery**

1394 **Purpose**

1395 To identify a set of registry package instances as the result of a query over
 1396 selected registry metadata.

1397 **RIM Binding**



1398 **Definition**

```

    1399
    1400 <!ELEMENT RegistryPackageQuery
    1401   ( PackageFilter?,
    1402     PackageHasMember*   )>
    1403
    1404 <!ELEMENT PackageHasMember
    1405   ( RegistryEntryQuery?,   )>
    
```

1406

1407 **Semantic Rules**

- 1408 1. Let RP denote the set of all persistent Package instances in the Registry. The
 1409 following steps will eliminate instances in RP that do not satisfy the conditions
 1410 of the specified filters.
 - 1411 a) If a PackageFilter is not specified, or if RP is empty, then continue below;
 1412 otherwise, let x be a package instance in RP. If x does not satisfy the
 1413 PackageFilter as defined in section 8.2.9, then remove x from RP.
 - 1414 b) If a PackageHasMember element is not directly contained in the
 1415 RegistryPackageQuery, or if RP is empty, then continue below; otherwise,
 1416 let x be a remaining package instance in RP. If x is an empty package,
 1417 then remove x from RP; otherwise, treat each PackageHasMember
 1418 element separately as follows:

1419

- 1420 If a RegistryEntryQuery element is not directly contained in the
 1421 PackageHasMember element, then let PM be the set of all RegistryEntry
 1422 instances that are members of the package x; otherwise, let RE be the set
 1423 of RegistryEntry instances returned by the RegistryEntryQuery as defined
 1424 in section 8.2.2 and let PM be the subset of RE that are members of the
 1425 package x. If PM is empty, then remove x from RP.
- 1426 2. If RP is empty, then raise the warning: *registry package query result is empty*.
 - 1427 3. Return RP as the result of the RegistryPackageQuery.
 - 1428 4. Return any accumulated warnings or exceptions as the StatusResult
 1429 associated with the RegistryPackageQuery.

1430 Examples

1431 A client application wishes to identify all package instances in the Registry that
 1432 contain an Invoice extrinsic object as a member of the package.

```

1433
1434 <RegistryPackageQuery>
1435   <PackageHasMember>
1436     <RegistryEntryQuery>
1437       <RegistryEntryFilter>
1438         objectType EQ "Invoice"           -- code by Clause, Section 8.2.10
1439       </RegistryEntryFilter>
1440     </RegistryEntryQuery>
1441   </PackageHasMember>
1442 </RegistryPackageQuery>
1443
  
```

1444 A client application wishes to identify all package instances in the Registry that
 1445 are not empty.

```

1446
1447 <RegistryEntryQuery>
1448   <PackageHasMember />
1449 </RegistryEntryQuery>
1450
  
```

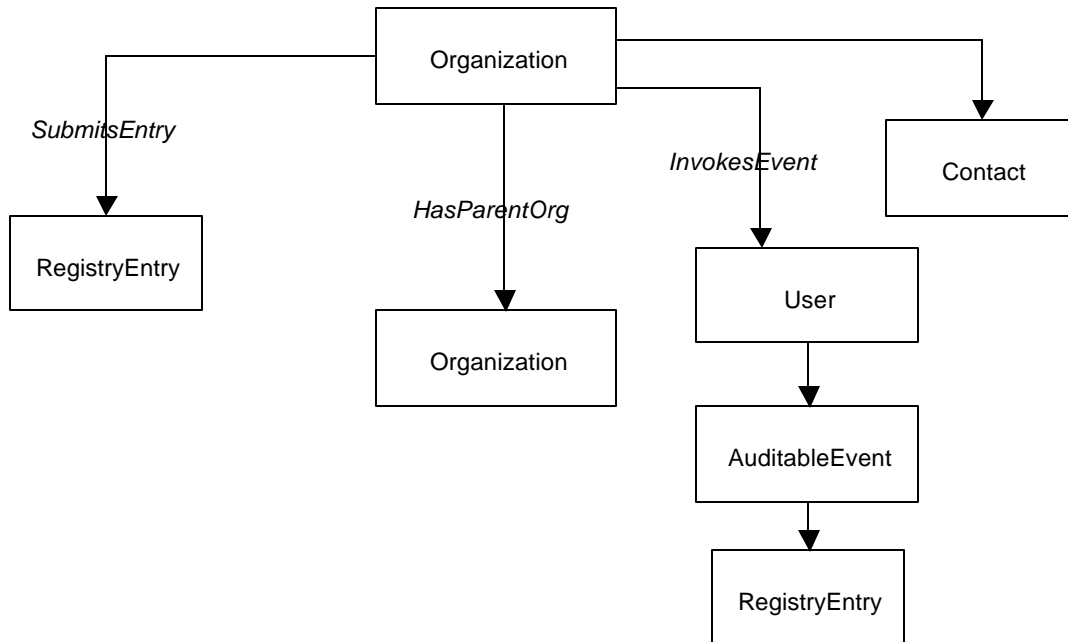
1451 A client application wishes to identify all package instances in the Registry that
 1452 are empty. Since the RegistryPackageQuery is not set up to do negations, clients
 1453 will have to do two separate RegistryPackageQuery requests, one to find all
 1454 packages and another to find all non-empty packages, and then do the set
 1455 difference themselves. Alternatively, they could do a more complex
 1456 RegistryEntryQuery and check that the packaging association between the
 1457 package and its members is non-existent.

1458 Note: A registry package is an intrinsic RegistryEntry instance that is completely
 1459 determined by its associations with its members. Thus a RegistryPackageQuery
 1460 can always be re-specified as an equivalent RegistryEntryQuery using
 1461 appropriate "AsSource" and "As Target" associations. However, the equivalent
 1462 RegistryEntryQuery is often more complicated to write.

1463

1463 **8.2.6 OrganizationQuery**

1464 **Purpose**



1465 To identify a set of organization instances as the result of a query over selected
 1466 registry metadata.

1467 **RIM Binding**

1468

1469 **Definition**

1470

```

1471 <!ELEMENT OrganizationQuery
1472 ( OrganizationFilter?,
1473 SubmitsEntry*,
1474 HasParentOrganization?,
1475 InvokesEvent*,
1476 ContactFilter* )>
1477
1478 <!ELEMENT SubmitsEntry ( RegistryEntryQuery? )>
1479
1480 <!ELEMENT HasParentOrganization
1481 ( OrganizationFilter?,
1482 HasParentOrganization? )>
1483
1484 <!ELEMENT InvokesEvent
1485 ( UserFilter?,
1486 AuditableEventFilter?,
    
```

1487 RegistryEntryQuery?)>

1488 **Semantic Rules**

- 1489 1. Let ORG denote the set of all persistent Organization instances in the
1490 Registry. The following steps will eliminate instances in ORG that do not
1491 satisfy the conditions of the specified filters.
- 1492 a) If an OrganizationFilter element is not directly contained in the
1493 OrganizationQuery element, or if ORG is empty, then continue below;
1494 otherwise, let x be an organization instance in ORG. If x does not satisfy
1495 the OrganizationFilter as defined in section 8.2.9, then remove x from RP.
- 1496 b) If a SubmitsEntry element is not specified within the OrganizationQuery, or
1497 if ORG is empty, then continue below; otherwise, consider each
1498 SubmitsEntry element separately as follows:
- 1499 If no RegistryEntryQuery is specified within the SubmitsEntry element,
1500 then let RES be the set of all RegistryEntry instances that have been
1501 submitted to the Registry by organization x; otherwise, let RE be the result
1502 of the RegistryEntryQuery as defined in section 8.2.2 and let RES be the
1503 set of all instances in RE that have been submitted to the Registry by
1504 organization x. If RES is empty, then remove x from ORG.
- 1505 c) If a HasParentOrganization element is not specified within the
1506 OrganizationQuery, or if ORG is empty, then continue below; otherwise,
1507 execute the following paragraph with $o = x$:
- 1508 Let o be an organization instance. If an OrganizationFilter is not specified
1509 within the HasParentOrganization and if o has no parent (i.e. if o is a root
1510 organization in the Organization hierarchy), then remove x from ORG;
1511 otherwise, let p be the parent organization of o. If p does not satisfy the
1512 OrganizationFilter, then remove x from ORG.
- 1513 If another HasParentOrganization element is directly contained within this
1514 HasParentOrganization element, then repeat the previous paragraph with
1515 $o = p$.
- 1516 d) If an InvokesEvent element is not specified within the OrganizationQuery,
1517 or if ORG is empty, then continue below; otherwise, consider each
1518 InvokesEvent element separately as follows:

- 1519 If an UserFilter is not specified, and if x is not the submitting organization
 1520 of some AuditableEvent instance, then remove x from ORG. If an
 1521 AuditableEventFilter is not specified, then let AE be the set of all
 1522 AuditableEvent instances that have x as the submitting organization;
 1523 otherwise, let AE be the set of AuditableEvent instances that satisfy the
 1524 AuditableEventFilter and have x as the submitting organization. If AE is
 1525 empty, then remove x from ORG. If a RegistryEntryQuery is not specified
 1526 in the InvokesEvent element, then let RES be the set of all RegistryEntry
 1527 instances associated with an event in AE; otherwise, let RE be the result
 1528 set of the RegistryEntryQuery, as specified in section 8.2.2, and let RES
 1529 be the subset of RE of entries submitted by x. If RES is empty, then
 1530 remove x from ORG.
- 1531 e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG
 1532 is empty, then continue below; otherwise, consider each ContactFilter
 1533 separately as follows:
- 1534 Let CT be the set of Contact instances that satisfy the ContactFilter and
 1535 are the contacts for organization x. If CT is empty, then remove x from
 1536 ORG.
- 1537 2. If ORG is empty, then raise the warning: *organization query result is empty*.
- 1538 3. Return ORG as the result of the OrganizationQuery.
- 1539 4. Return any accumulated warnings or exceptions as the StatusResult
 1540 associated with the OrganizationQuery.

1541 Examples

1542 A client application wishes to identify a set of organizations, based in France, that
 1543 have submitted a PartyProfile extrinsic object this year.

```

1545 <OrganizationQuery>
1546   <OrganizationFilter>
1547     country EQ "France"           -- code by Clause, Section 8.2.10
1548   </OrganizationFilter>
1549   <SubmitsEntry>
1550     <RegistryEntryQuery>
1551       <RegistryEntryFilter>
1552         objectType EQ "PartyProfile" -- code by Clause, Section 8.2.10
1553       </RegistryEntryFilter>
1554       <RegistryEntryAuditableEvent>
1555         <AuditableEventFilter>
1556           timestamp GE "2001-01-01" -- code by Clause, Section 8.2.10
1557         </AuditableEventFilter>
1558       </RegistryEntryAuditableEvent>
1559     </RegistryEntryQuery>
1560   </SubmitsEntry>
1561 </OrganizationQuery>

```

1562

1563 A client application wishes to identify all organizations that have XYZ,
1564 Corporation as a parent. The client knows that the URN for XYZ, Corp. is
1565 urn:ebxml:org:xyz, but there is no guarantee that subsidiaries of XYZ have a
1566 URN that uses the same format, so a full query is required.

```
1567
1568     <OrganizationQuery>
1569         <HasParentOrganization>
1570             <OrganizationFilter>
1571                 ID = "urn:ebxml:org:xyz"      -- code by Clause, Section 8.2.10
1572             </OrganizationFilter>
1573         </HasParentOrganization>
1574     </OrganizationQuery>
1575
```

1575 **8.2.7 GetRegistryEntry**

1576 **Purpose**

1577 To construct an XML document that contains selected registry metadata
 1578 associated with the registry entries identified by a RegistryEntryQuery. NOTE:
 1579 Initially, the RegistryEntryQuery could be the URN identifier for a single registry
 1580 entry.

1581 **Definition**

```

1582
1583 <!ELEMENT GetRegistryEntry
1584   ( RegistryEntryQuery,
1585     WithClassifications?,
1586     WithAsSourceAssociations?,
1587     WithAsTargetAssociations?,
1588     WithAuditableEvents?,
1589     WithExternalLinks? )>
1590
1591 <!ELEMENT WithClassifications ( ClassificationFilter? )>
1592 <!ELEMENT WithAsSourceAssociations ( AssociationFilter? )>
1593 <!ELEMENT WithAsTargetAssociations ( AssociationFilter? )>
1594 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>
1595 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
1596
1597 <!ELEMENT GetRegistryEntryResult
1598   ( RegistryEntryMetadata*, StatusResult )>
1599
1600 <!ELEMENT RegistryEntryMetadata
1601   ( RegistryEntry,
1602     Classification*,
1603     AsSourceAssociations?,
1604     AsTargetAssociations?,
1605     AuditableEvent*,
1606     ExternalLink* )>
1607
1608 <!ELEMENT AsSourceAssociations ( Association* )>
1609 <!ELEMENT AsTargetAssociations ( Association* )>

```

1610 **Semantic Rules**

- 1611 1. The RegistryEntry, Classification, Association, AuditableEvent, and
 1612 ExternalLink elements contained in the GetRegistryEntryResult are defined
 1613 by the ebXML Registry DTD specified in Appendix A.2.
- 1614 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in
 1615 section 8.2.2, and let R be the result set of identifiers for registry entry
 1616 instances. Let S be the set of status elements returned in the StatusResult. If
 1617 any status element in S is an exception condition, then stop execution and
 1618 return the same StatusResult element in the GetRegistryEntryResult.

- 1619 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement
1620 in the GetRegistryEntryResult. Instead, raise the warning: *no resulting registry*
1621 *entry*. Add this warning to the StatusResult returned by the
1622 RegistryEntryQuery and return this enhanced StatusResult with the
1623 GetRegistryEntryResult.
- 1624 4. For each registry entry E referenced by an element of R, use the attributes of
1625 E to create a new RegistryEntry element as defined in Appendix A.2. Then
1626 create a new RegistryEntryMetadata element as defined above to be the
1627 parent element of that RegistryEntry element.
- 1628 5. If no With option is specified, then the resulting RegistryEntryMetadata
1629 element has no Classification, AsSourceAssociations, AsTargetAssociations,
1630 AuditableEvent, or ExternalData subelements. The set of
1631 RegistryEntryMetadata elements, with the StatusResult from the
1632 RegistryEntryQuery, is returned as the GetRegistryEntryResult.
- 1633 6. If WithClassifications is specified, then for each E in R do the following: If a
1634 ClassificationFilter is not present, then let C be any classification instance
1635 linked to E; otherwise, let C be a classification instance linked to E that
1636 satisfies the ClassificationFilter (Section 8.2.9). For each such C, create a
1637 new Classification element as defined in Appendix A.2. Add these
1638 Classification elements to their parent RegistryEntryMetadata element.
- 1639 7. If WithAsSourceAssociations is specified, then for each E in R do the
1640 following: If an AssociationFilter is not present, then let A be any association
1641 instance whose source object is E; otherwise, let A be an association
1642 instance that satisfies the AssociationFilter (Section 8.2.9) and whose source
1643 object is E. For each such A, create a new Association element as defined in
1644 Appendix A.2. Add these Association elements as subelements of the
1645 WithAsSourceAssociations and add that element to its parent
1646 RegistryEntryMetadata element.
- 1647 8. If WithAsTargetAssociations is specified, then for each E in R do the
1648 following: If an AssociationFilter is not present, then let A be any association
1649 instance whose target object is E; otherwise, let A be an association instance
1650 that satisfies the AssociationFilter (Section 8.2.9) and whose target object is
1651 E. For each such A, create a new Association element as defined in Appendix
1652 A.2. Add these Association elements as subelements of the
1653 WithAsTargetAssociations and add that element to its parent
1654 RegistryEntryMetadata element.
- 1655 9. If WithAuditableEvents is specified, then for each E in R do the following: If an
1656 AuditableEventFilter is not present, then let A be any auditable event instance
1657 linked to E; otherwise, let A be any auditable event instance linked to E that
1658 satisfies the AuditableEventFilter (Section 8.2.9). For each such A, create a
1659 new AuditableEvent element as defined in Appendix A.2. Add these
1660 AuditableEvent elements to their parent RegistryEntryMetadata element.

- 1661 10. If `WithExternalLinks` is specified, then for each E in R do the following: If an
1662 `ExternalLinkFilter` is not present, then let L be any external link instance linked
1663 to E; otherwise, let L be any external link instance linked to E that satisfies the
1664 `ExternalLinkFilter` (Section 8.2.9). For each such D, create a new `ExternalLink`
1665 element as defined in Appendix A.2. Add these `ExternalLink` elements to their
1666 parent `RegistryEntryMetadata` element.
- 1667 11. If any warning or exception condition results, then add the code and the
1668 message to the `StatusResult` that came from the `RegistryEntryQuery` result.
- 1669 12. Return the set of `RegistryEntryMetadata` elements and the revised
1670 `StatusResult` as the content of the `GetRegistryEntryResult`.

1671

1672 **Examples**

1673 A customer of XYZ Corporation has been using a `PurchaseOrder` DTD registered
1674 by XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer
1675 wishes to check on the current status of that DTD, especially if it has been
1676 superceded or replaced, and get all of its current classifications. The following
1677 query request will return an XML document with the registry entry for the existing
1678 DTD as the root, with all of its classifications, and with associations to registry
1679 entries for any items that have superceded or replaced it.

```
1680
1681 <GetRegistryEntry>
1682   <RegistryEntryQuery>
1683     <RegistryEntryFilter>
1684       ID EQ "urn:com:xyz:po:325"           -- code by Clause, Section 8.2.10
1685     </RegistryEntryFilter>
1686   </RegistryEntryQuery>
1687   <WithClassifications/>
1688   <WithAsSourceAssociations>
1689     <AssociationFilter>                   -- code by Clause, Section 8.2.10
1690       AssociationType EQUALS "SupercededBy" OR
1691       AssociationType EQUALS "ReplacedBy"
1692     </AssociationFilter>
1693   </WithAsSourceAssociations>
1694 </GetRegistryEntry>
```

1695

1696 A client of the Registry registered an XML DTD several years ago and is now
1697 thinking of replacing it with a revised version. The identifier for the existing DTD
1698 is "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible
1699 with the existing DTD. The client desires a list of all registered items that use the
1700 existing DTD so they can assess the impact of an incompatible change. The
1701 following query returns an XML document that is a list of all `RegistryEntry`
1702 elements that represent registered items that use, contain, or extend the given
1703 DTD. The document also links each `RegistryEntry` element in the list to an
1704 element for the identified association.


```

1705
1706 <GetRegistryEntry>
1707   <RegistryEntryQuery>
1708     <AsSourceAssociation>
1709       <AssociationFilter>           -- code by Clause, Section 8.2.10
1710         AssociationType EQ "Contains" OR
1711         AssociationType EQ "Uses" OR
1712         AssociationType EQ "Extends"
1713       </AssociationFilter>
1714     <RegistryEntryFilter>         -- code by Clause, Section 8.2.10
1715       ID = "urn:xyz:dtd:po97"
1716     </RegistryEntryFilter>
1717   </AsSourceAssociation>
1718 </RegistryEntryQuery>
1719 <WithAsSourceAssociations>
1720   <AssociationFilter>           -- code by Clause, Section 8.2.10
1721     AssociationType EQ "Contains" OR
1722     AssociationType EQ "Uses" OR
1723     AssociationType EQ "Extends"
1724   </AssociationFilter>
1725 </WithAsSourceAssociations>
1726 </GetRegistryEntry>

```

1727

1728 A user has been browsing the registry and has found a registry entry that
1729 describes a package of core-components that should solve the user's problem.
1730 The package URN identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to
1731 know what's in the package. The following query returns an XML document with
1732 a registry entry for each member of the package along with that member's Uses
1733 and PackageHasMember associations.

```

1734
1735 <GetRegistryEntry>
1736   <RegistryEntryQuery>
1737     <AsTargetAssociation>
1738       <AssociationFilter>           -- code by Clause, Section 8.2.10
1739         AssociationType EQ "HasMember"
1740       </AssociationFilter>
1741     <RegistryEntryFilter>         -- code by Clause, Section 8.2.10
1742       ID = " urn:com:cc:pkg:ccstuff "
1743     </RegistryEntryFilter>
1744   </AsTargetAssociation>
1745 </RegistryEntryQuery>
1746 <WithAsSourceAssociations>
1747   <AssociationFilter>           -- code by Clause, Section 8.2.10
1748     AssociationType EQ "HasMember" OR
1749     AssociationType EQ "Uses"
1750   </AssociationFilter>
1751 </WithAsSourceAssociations>
1752 </GetRegistryEntry>

```

1753
1754

1754 8.2.8 GetRepositoryItem

1755 Purpose

1756 To construct an XML document that contains one or more repository items, and
 1757 some associated metadata, by submitting a RegistryEntryQuery to the
 1758 registry/repository that holds the desired objects. NOTE: Initially, the
 1759 RegistryEntryQuery could be the URN identifier for a single registry entry.

1760 Definition

```

1761
1762 <!ELEMENT GetRepositoryItem
1763 ( RegistryEntryQuery,
1764   RecursiveAssociationOption?,
1765   WithShortDescription? )>
1766
1767 <!ELEMENT RecursiveAssociationOption ( AssociationRole+ )>
1768 <!ATTLIST RecursiveAssociationOption
1769   depthLimit CDATA #IMPLIED >
1770
1771 <!ELEMENT AssociationRole EMPTY >
1772 <!ATTLIST AssociationRole
1773   role CDATA #REQUIRED >
1774
1775 <!ELEMENT WithShortDescription EMPTY >
1776
1777 <!ELEMENT GetRepositoryItemResult
1778 ( RepositoryItem*, StatusResult )>
1779
1780 <!ELEMENT RepositoryItem
1781 ( ClassificationScheme
1782   | RegistryPackage
1783   | ExtrinsicObject
1784   | WithdrawnObject
1785   | ExternalItem )>
1786 <!ATTLIST RepositoryItem
1787   identifier CDATA #REQUIRED
1788   name CDATA #REQUIRED
1789   repositoryURL CDATA #REQUIRED
1790   objectType CDATA #REQUIRED
1791   status CDATA #REQUIRED
1792   stability CDATA #REQUIRED
1793   description CDATA #IMPLIED >
1794
1795 <!ELEMENT ExtrinsicObject (#PCDATA) >
1796 <!ATTLIST ExtrinsicObject
1797   byteEncoding CDATA "Base64" >
1798
1799 <!ELEMENT WithdrawnObject EMPTY >
1800
1801 <!ELEMENT ExternalItem EMPTY >
1802
```

1803

1804 **Semantic Rules**

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

- 1843 b) If E.objectURL references a registered object in some other
1844 registry/repository, then create a new RepositoryItem element, with values
1845 for its attributes derived as specified in Semantic Rule 7, and create a new
1846 ExternallItem element as the subelement of this RepositoryItem.
- 1847 c) If E.objectURL is void, i.e. the object it would have referenced has been
1848 withdrawn, then create a new RepositoryItem element, with values for its
1849 attributes derived as specified in Semantic Rule 7, and create a new
1850 WithdrawnObject element as the subelement of this RepositoryItem.
- 1851 7. Let E be a registry entry and let RO be the RepositoryItem element created in
1852 Semantic Rule 6. Set the attributes of RO to the values derived from the
1853 corresponding attributes of E. If WSD="yes", include the value of the
1854 description attribute; otherwise, do not include it. Insert this new
1855 RepositoryItem element into the Result set.
- 1856 8. Let R be defined as in Semantic Rule 4. Execute Semantic Rule 9 with Y as
1857 the set of RegistryEntry instances referenced by R. Then continue with
1858 Semantic rule 10.
- 1859 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an
1860 empty set of RegistryEntry instances. For each registry entry E in Y, and for
1861 each AssociationRole A of the RecursiveAssociationOption, do the following:
- 1862 a) Let Z be the set of target items E' linked to E under association instances
1863 having E as the source object, E' as the target object, and A as the
1864 AssociationType.
- 1865 b) Add the elements of Z to NextLevel.
- 1866 10. Let X be the set of new registry entries that are in NextLevel but are not yet
1867 represented in the Result set.
- 1868 Case:
- 1869 a) If X is empty, then return the content of Result as the set of
1870 RepositoryItem elements in the GetRepositoryItemResult element.
- 1871 b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input
1872 set. When finished, add the elements of X to Y and set Depth=Depth+1. If
1873 Depth is now equal to Limit, then return the content of Result as the set of
1874 RepositoryItem elements in the GetRepositoryItemResult element;
1875 otherwise, repeat Semantic Rules 9 and 10 with the new set Y of registry
1876 entries.
- 1877 11. If any exception, warning, or other status condition results during the
1878 execution of the above, then return appropriate status elements as the
1879 StatusResult of the GetRepositoryItemResult element created in Semantic
1880 Rule 5 or Semantic Rule 10.

1881

1882 **Examples**

1883 A registry client has found a registry entry for a core-component item. The item's
 1884 URN identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item
 1885 that uses many other registered items. The client desires the collection of all
 1886 items needed for a complete implementation of "goodthing". The following query
 1887 returns an XML document that is a collection of all needed items.

```
1888
1889     <GetRepositoryItem>
1890         <RegistryEntryQuery>
1891             <RegistryEntryFilter>                -- code by Clause, Section 8.2.10
1892                 ID EQ "urn:ebxml:cc:goodthing"
1893             </RegistryEntryFilter>
1894         </RegistryEntryQuery>
1895         <RecursiveAssociationOption>
1896             <AssociationRole role="Uses" />
1897             <AssociationRole role="ValidatesTo" />
1898         </RecursiveAssociationOption>
1899     </GetRepositoryItem>
```

1900

1901 A registry client has found a reference to a core-component routine
 1902 ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client
 1903 knows that all routines have a required association to its defining UML
 1904 specification. The following query returns both the routine and its UML
 1905 specification as a collection of two items in a single XML document.

```
1906
1907     <GetRepositoryItem>
1908         <RegistryEntryQuery>
1909             <RegistryEntryFilter>                -- code by Clause, Section 8.2.10
1910                 ID EQ "urn:ebxml:cc:rtn:nice87"
1911             </RegistryEntryFilter>
1912         </RegistryEntryQuery>
1913         <RecursiveAssociationOption depthLimit="1" >
1914             <AssociationRole role="ValidatesTo" />
1915         </RecursiveAssociationOption>
1916     </GetRepositoryItem>
```

1917

1918 A user has been told that the 1997 version of the North American Industry
 1919 Classification System (NAICS) is stored in the NIST registry with URN identifier
 1920 "urn:nist:cs:naics-1997". The following query would retrieve the complete
 1921 classification scheme, with all 1810 nodes, as an XML document that validates to
 1922 a classification scheme DTD.

```
1923
1924     <GetRepositoryItem>
1925         <RegistryEntryQuery>
1926             <RegistryEntryFilter>                -- code by Clause, Section 8.2.10
```

```
1927         ID EQ " urn:nist:cs:naics-1997"  
1928         </RegistryEntryFilter>  
1929     </RegistryEntryQuery>  
1930 </GetRepositoryItem>
```

1931

1932 Note: The GetRepositoryItemResult would include a single RepositoryItem that
1933 consists of the ClassificationScheme document with content:
1934 <ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt>
1935

1935 **8.2.9 Registry Filters**

1936 **Purpose**

1937 To identify a subset of the set of all persistent instances of a given registry class.

1938 **Definition**

1939
1940 <!ELEMENT ObjectFilter (Clause)>
1941
1942 <!ELEMENT RegistryEntryFilter (Clause)>
1943
1944 <!ELEMENT IntrinsicObjectFilter (Clause)>
1945
1946 <!ELEMENT ExtrinsicObjectFilter (Clause)>
1947
1948 <!ELEMENT PackageFilter (Clause)>
1949
1950 <!ELEMENT OrganizationFilter (Clause)>
1951
1952 <!ELEMENT ContactFilter (Clause)>
1953
1954 <!ELEMENT ClassificationNodeFilter (Clause)>
1955
1956 <!ELEMENT AssociationFilter (Clause)>
1957
1958 <!ELEMENT ClassificationFilter (Clause)>
1959
1960 <!ELEMENT ExternalLinkFilter (Clause)>
1961
1962 <!ELEMENT AuditableEventFilter (Clause)>
1963
1964 <!ELEMENT UserFilter (Clause)>

1965

1966 **Semantic Rules**

- 1967 1. The Clause element is defined in section 8.2.10, Clause.
- 1968 2. For every ObjectFilter XML element, the leftargument attribute of any
1969 containing SimpleClause shall identify a public attribute of the Object UML
1970 class defined in [RIM]. If not, raise exception: *object attribute error*. The
1971 ObjectFilter returns a set of identifiers for Object instances whose attribute
1972 values evaluate to *True* for the Clause predicate.
- 1973 3. For every RegistryEntryFilter XML element, the leftargument attribute of any
1974 containing SimpleClause shall identify a public attribute of the RegistryEntry
1975 UML class defined in [RIM].

- 1976 If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
1977 returns a set of identifiers for RegistryEntry instances whose attribute values
1978 evaluate to *True* for the Clause predicate.
- 1979 4. For every IntrinsicObjectFilter XML element, the leftargument attribute of any
1980 containing SimpleClause shall identify a public attribute of the IntrinsicObject
1981 UML class defined in [RIM]. If not, raise exception: *intrinsic object attribute*
1982 *error*. The IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject
1983 instances whose attribute values evaluate to *True* for the Clause predicate.
- 1984 5. For every ExtrinsicObjectFilter XML element, the leftargument attribute of any
1985 containing SimpleClause shall identify a public attribute of the ExtrinsicObject
1986 UML class defined in [RIM]. If not, raise exception: *extrinsic object attribute*
1987 *error*. The ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject
1988 instances whose attribute values evaluate to *True* for the Clause predicate.
- 1989 6. For every PackageFilter XML element, the leftargument attribute of any
1990 containing SimpleClause shall identify a public attribute of the Package UML
1991 class defined in [RIM]. If not, raise exception: *package attribute error*. The
1992 PackageFilter returns a set of identifiers for Package instances whose
1993 attribute values evaluate to *True* for the Clause predicate.
- 1994 7. For every OrganizationFilter XML element, the leftargument attribute of any
1995 containing SimpleClause shall identify a public attribute of the Organization or
1996 PostalAddress UML classes defined in [RIM]. If not, raise exception:
1997 *organization attribute error*. The OrganizationFilter returns a set of identifiers
1998 for Organization instances whose attribute values evaluate to *True* for the
1999 Clause predicate.
- 2000 8. For every ContactFilter XML element, the leftargument attribute of any
2001 containing SimpleClause shall identify a public attribute of the Contact or
2002 PostalAddress UML class defined in [RIM]. If not, raise exception: *contact*
2003 *attribute error*. The ContactFilter returns a set of identifiers for Contact
2004 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2005 9. For every ClassificationNodeFilter XML element, the leftargument attribute of
2006 any containing SimpleClause shall identify a public attribute of the
2007 ClassificationNode UML class defined in [RIM]. If not, raise exception:
2008 *classification node attribute error*. The ClassificationNodeFilter returns a set of
2009 identifiers for ClassificationNode instances whose attribute values evaluate to
2010 *True* for the Clause predicate.
- 2011 10. For every AssociationFilter XML element, the leftargument attribute of any
2012 containing SimpleClause shall identify a public attribute of the Association
2013 UML class defined in [RIM]. If not, raise exception: *association attribute error*.
2014 The AssociationFilter returns a set of identifiers for Association instances
2015 whose attribute values evaluate to *True* for the Clause predicate.

- 2016 11. For every ClassificationFilter XML element, the leftargument attribute of any
2017 containing SimpleClause shall identify a public attribute of the Classification
2018 UML class defined in [RIM]. If not, raise exception: *classification attribute*
2019 *error*. The ClassificationFilter returns a set of identifiers for Classification
2020 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2021 12. For every ExternalLinkFilter XML element, the leftargument attribute of any
2022 containing SimpleClause shall identify a public attribute of the ExternalLink
2023 UML class defined in [RIM]. If not, raise exception: *external link attribute*
2024 *error*. The ExternalLinkFilter returns a set of identifiers for ExternalLink instances
2025 whose attribute values evaluate to *True* for the Clause predicate.
- 2026 13. For every AuditableEventFilter XML element, the leftargument attribute of any
2027 containing SimpleClause shall identify a public attribute of the AuditableEvent
2028 UML class defined in [RIM]. If not, raise exception: *auditable event attribute*
2029 *error*. The AuditableEventFilter returns a set of identifiers for AuditableEvent
2030 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2031 14. For every UserFilter XML element, the leftargument attribute of any
2032 containing SimpleClause shall identify a public attribute of the User UML
2033 class defined in [RIM]. If not, raise exception: *auditable identity attribute*
2034 *error*. The UserFilter returns a set of identifiers for User instances whose attribute
2035 values evaluate to *True* for the Clause predicate.
- 2036
2037

2037 **8.2.10 XML Clause Constraint Representation**

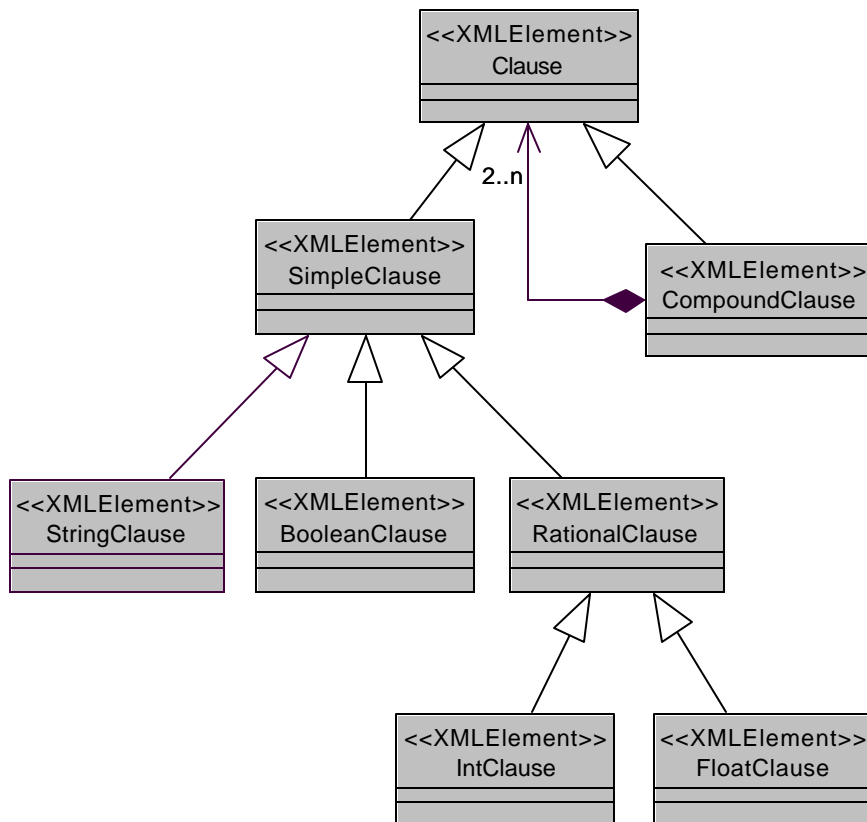
2038 **Purpose**

2039 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate*
 2040 *Clauses*. Predicate Clauses are utilized to formally define the constraint
 2041 mechanism, and are referred to simply as **Clauses** in this specification.

2042 **Conceptual UML Diagram**

2043 The following is a conceptual diagram outlining the Clause base structure. It is
 2044 expressed in UML for visual depiction.

2045



2046

2047

2048

2049

2050

2051 **Semantic Rules**

2052 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate -
 2053 RightArgument" format to form a *Clause*. There are two types of Clauses:
 2054 *SimpleClauses* and *CompoundClauses*.

2055 SimpleClauses

2056 A SimpleClause always defines the left argument as a text string, sometimes
 2057 referred to as the *Subject* of the Clause. SimpleClause itself is incomplete
 2058 (abstract) and must be extended. SimpleClause is extended to support
 2059 BooleanClause, StringClause, and RationalClause (abstract).

2060 BooleanClause implicitly defines the predicate as 'equal to', with the right
 2061 argument as a boolean. StringClause defines the predicate as an enumerated
 2062 attribute of appropriate string-compare operations and a right argument as the
 2063 element's text data. Rational number support is provided through a common
 2064 RationalClause providing an enumeration of appropriate rational number
 2065 compare operations, which is further extended to IntClause and FloatClause,
 2066 each with appropriate signatures for the right argument.

2067 CompoundClauses

2068 A CompoundClause contains two or more Clauses (Simple or Compound) and a
 2069 connective predicate. This provides for arbitrarily complex Clauses to be formed.

2070

2071 **Definition**

2072

```

2073 <!ELEMENT Clause ( SimpleClause | CompoundClause )>
2074
2075 <!ELEMENT Simpleclause
2076 ( BooleanClause | RationalClause | StringClause )>
2077 <!ATTLIST SimpleClause leftargument CDATA #REQUIRED >
2078
2079 <!ELEMENT CompoundClause ( Clause, Clause+ )>
2080 <!ATTLIST CompoundClause connectivepredicate ( And | Or ) #REQUIRED>
2081
2082 <!ELEMENT BooleanClause EMPTY >
2083 <!ATTLIST BooleanClause booleanpredicate ( True | False ) #REQUIRED>
2084
2085 <!ELEMENT RationalClause ( IntClause | FloatClause )>
2086 <!ATTLIST RationalClause logicalpredicate
2087 ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
2088
2089 <!ELEMENT IntClause ( #PCDATA )>
2090 <!ATTLIST IntClause e-dtype NMTOKEN #FIXED 'int' >
2091
2092 <!ELEMENT FloatClause ( #PCDATA )>
2093 <!ATTLIST FloatClause e-dtype NMTOKEN #FIXED 'float' >
2094
2095 <!ELEMENT StringClause ( #PCDATA )>

```

```
2096 <!ATTLIST StringClause stringpredicate
2097 ( contains | -contains | startswith |
2098 -startswith | endswith | -endswith ) #REQUIRED >
```

2099

2100 Examples

2101 Simple BooleanClause: "Smoker" = True

```
2102
2103 <?xml version="1.0" encoding="UTF-8"?>
2104 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2105 <Clause>
2106 <SimpleClause leftargument="Smoker">
2107 <BooleanClause booleanpredicate="True"/>
2108 </SimpleClause>
2109 </Clause>
2110
```

2111 Simple StringClause: "Smoker" contains "mo"

```
2112
2113 <?xml version="1.0" encoding="UTF-8"?>
2114 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2115 <Clause>
2116 <SimpleClause leftargument="Smoker">
2117 <StringClause stringcomparepredicate="contains">
2118 mo
2119 </StringClause>
2120 </SimpleClause>
2121 </Clause>
```

2122

2123 Simple IntClause: "Age" >= 7

```
2124
2125 <?xml version="1.0" encoding="UTF-8"?>
2126 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2127 <Clause>
2128 <SimpleClause leftargument="Age">
2129 <RationalClause logicalpredicate="GE">
2130 <IntClause e-dtype="int">7</IntClause>
2131 </RationalClause>
2132 </SimpleClause>
2133 </Clause>
2134
```

2135 Simple FloatClause: "Size" = 4.3

```
2136
2137 <?xml version="1.0" encoding="UTF-8"?>
2138 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2139 <Clause>
2140 <SimpleClause leftargument="Size">
2141 <RationalClause logicalpredicate="E">
```

```

2142         <FloatClause e-dtype="float">4.3</FloatClause>
2143     </RationalClause>
2144 </SimpleClause>
2145 </Clause>

```

2146

2147 **Compound with two Simples ("Smoker" = False)AND("Age" =< 45))**

```

2148
2149 <?xml version="1.0" encoding="UTF-8"?>
2150 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2151 <Clause>
2152     <CompoundClause connectivepredicate="And">
2153         <Clause>
2154             <SimpleClause leftargument="Smoker">
2155                 <BooleanClause booleanpredicate="False"/>
2156             </SimpleClause>
2157         </Clause>
2158         <Clause>
2159             <SimpleClause leftargument="Age">
2160                 <RationalClause logicalpredicate="EL">
2161                     <IntClause e-dtype="int">45</IntClause>
2162                 </RationalClause>
2163             </SimpleClause>
2164         </Clause>
2165     </CompoundClause>
2166 </Clause>

```

2167

2168 **Coumpound with one Simple and one Compound**2169 **(("Smoker" = False)And(("Age" =< 45)Or("American"=True)))**

```

2170
2171 <?xml version="1.0" encoding="UTF-8"?>
2172 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2173 <Clause>
2174     <CompoundClause connectivepredicate="And">
2175         <Clause>
2176             <SimpleClause leftargument="Smoker">
2177                 <BooleanClause booleanpredicate="False"/>
2178             </SimpleClause>
2179         </Clause>
2180         <Clause>
2181             <CompoundClause connectivepredicate="Or">
2182                 <Clause>
2183                     <SimpleClause leftargument="Age">
2184                         <RationalClause logicalpredicate="EL">
2185                             <IntClause e-dtype="int">45</IntClause>
2186                         </RationalClause>
2187                     </SimpleClause>
2188                 </Clause>
2189                 <Clause>
2190                     <SimpleClause leftargument="American">

```

```
2191         <BooleanClause booleanpredicate="True" />
2192     </SimpleClause>
2193 </Clause>
2194 </CompoundClause>
2195 </Clause>
2196 </CompoundClause>
2197 </Clause>
```

2198 **8.3 SQL Query Support**

2199 The Registry may optionally support an SQL based query capability that is
2200 designed for Registry clients that demand more complex query capability. The
2201 optional SQLQuery element in the AdhocQueryRequest allows a client to submit
2202 complex SQL queries using a declarative query language.

2203 The syntax for the SQLQuery of the Registry is defined by a stylized use of a
2204 proper subset of the "SELECT" statement of Entry level SQL defined by ISO/IEC
2205 9075:1992, Database Language SQL [SQL], extended to include `<sql`
2206 `invoked routines`> (also known as stored procedures) as specified in
2207 ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined in template form in
2208 appendix C.3. The exact syntax of the Registry query language is defined by the
2209 BNF grammar in C.1.

2210 Note that the use of a subset of SQL syntax for SQLQuery does not imply a
2211 requirement to use relational databases in a Registry implementation.

2212 **8.3.1 SQL Query Syntax Binding To [RIM]**

2213 SQL Queries are defined based upon the query syntax in in appendix C.1 and a
2214 fixed relational schema defined in appendix C.3. The relational schema is an
2215 algorithmic binding to [RIM] as described in the following sections.

2216 **8.3.1.1 Interface and Class Binding**

2217 A subset of the Interface and class names defined in [RIM] map to table names
2218 that may be queried by an SQL query. Appendix C.3 defines the names of the
2219 RIM interfaces and classes that may be queries by an SQL query.

2220 The algorithm used to define the binding of [RIM] classes to table definitions in
2221 appendix C.3 is as follows:

2222 ?? Only those classes and interfaces that have concrete instances are
2223 mapped to relational tables. This results in intermediate interfaces in the
2224 inheritance hierarchy, such as Object and IntrinsicObject to not map to
2225 SQL tables. An exception to this rule is RegistryEntry as defined next.

- 2226 ?? A special view called RegistryEntry is defined to allow SQL queries to be
2227 made against RegistryEntry instances. This is the only interface defined in
2228 [RIM] that does not have concrete instances but is queryable by SQL
2229 queries.
- 2230 ?? The names of relational tables are the same as the corresponding [RIM]
2231 class or interface name. However, the name binding is case insensitive.
- 2232 ?? Each [RIM] class or interface that maps to a table in appendix C.3
2233 includes column definitions in appendix C.3 where the column definitions
2234 are based on a subset of attributes defined for that class or interface in
2235 [RIM]. The attributes that map to columns include the inherited attributes
2236 for the [RIM] class or interface. Comments in appendix C.3 indicate which
2237 ancestor class or interface contributed which column definitions.
- 2238 An SQLQuery against a table not defined in appendix C.3 may result in an
2239 ebXMLError message with an InvalidQueryException.
- 2240 The algorithm for mapping of attributes to column definitions will be described in
2241 following sections.
- 2242 **8.3.1.2 Accessor Method To Attribute Binding**
- 2243 Most of the [RIM] interfaces methods are simple get methods that map directly to
2244 attributes. For example the getName method on Object maps to a *name* attribute
2245 of type String. Each get method in [RIM] defines the exact attribute name that it
2246 maps to in the interface definitions in [RIM].
- 2247 **8.3.1.3 Primitive Attributes Binding**
- 2248 Attributes defined by [RIM] that are of primitive types (e.g. String) may be used in
2249 the same way as column names in SQL. Again the exact attribute names are
2250 defined in the interface definitions in [RIM]. Note that while names are in mixed
2251 case, SQL-92 is case insensitive. It is therefore valid for a query to contain
2252 attribute names that do not exactly match the case defined in [RIM].
- 2253 **8.3.1.4 Reference Attribute Binding**
- 2254 A few of the [RIM] interface methods return references to instances of interfaces
2255 or classes defined by [RIM]. For example, the getAccessControlPolicy method of
2256 the Object class returns a reference to an instance of an AccessControlPolicy
2257 object.
- 2258 In such cases the reference maps to the *id* attribute for the referenced object.
2259 The name of the resulting column is the same as the attribute name in [RIM] as
2260 defined by 8.3.1.3. The data type for the column is UUID as defined in appendix
2261 C.3.

2262 When a reference attribute value holds a null reference it maps to a null value in
2263 the SQL binding which may be tested with the <null specification> as defined by
2264 [SQL].

2265 Reference attribute binding is a special case of a primitive attribute mapping.

2266 **8.3.1.5 Complex Attribute Binding**

2267 A few of the [RIM] interfaces define attributes that are not primitive types. Instead
2268 they are of a complex type as defined by an entity class in [RIM]. Examples
2269 include attributes of type TelephoneNumber, Contact, PersonName etc. in
2270 interface Organization and class Contact.

2271 The SQL query schema algorithmically maps such complex attributes as multiple
2272 primitive attributes within the parent table. The mapping simply flattens out the
2273 entity class attributes within the parent table. The attribute name for the flattened
2274 attributes are composed of a concatenation of attribute names in the reference
2275 chain. For example Organization has a contact attribute of type Contact. Contact
2276 has an address attribute of type PostalAddress. PostalAddress has a String
2277 attribute named city. This city attribute will be named contact_address_city.

2278 **8.3.1.6 Collection Attribute Binding**

2279 A few of the [RIM] interface methods return collection of references to instances
2280 of interfaces or classes defined by [RIM]. For example, the getPackages method
2281 of the ManagedObject class returns a Collection of references to instances of
2282 Packages that the object is a member of.

2283 Such collection attributes in [RIM] classes have been mapped to stored
2284 procedures in appendix C.3 such that these stored procedures return a collection
2285 of `id` attribute values. The returned value of these stored procedures can be
2286 treated as the result of a table sub-query in SQL.

2287 These stored procedures may be used SQL IN clause to test for membership of
2288 an object in such collections of references.

2289 **8.3.2 Semantic Constraints On Query Syntax**

2290 This section defines simplifying constraints on the query syntax that cannot be
2291 expressed in the BNF for the query syntax. These constraints must be applied in
2292 the semantic analysis of the query.

2293 1. Class names and attribute names must be processed in a case insensitive
2294 manner.

2295

2296 2. The syntax used for stored procedure invocation must be consistent with
2297 the syntax of an SQL procedure invocation as specified by ISO/IEC 9075-
2298 4 [SQL/PSM].

2299

2300 The SQL select column specified must always be `t.id` for this version of the
2301 specification, where `t` is a table reference in the FROM clause.

2302 **8.3.3 SQL Query Results**

2303 The results of an SQL query is always an `ObjectRefList` as defined by the
2304 `AdHocQueryResponse` in 8.3.12. This means the result of an SQL query is
2305 always a collection of references to instances of a sub-class of the `Object`
2306 interface in [RIM]. This is reflected in a semantic constraint that requires that the
2307 SQL select column specified must always be an `id` column in a table in appendix
2308 C.3 for this version of the specification.

2309 **8.3.4 Simple Metadata Based Queries**

2310 The simplest form of an SQL query is based upon metadata attributes specified
2311 for a single class within [RIM]. This section gives some examples of simple
2312 metadata based queries.

2313 For example, to get the collection of `ExtrinsicObjects` whose name contains the
2314 word 'Acme' and that have a version greater than 1.3, the following query
2315 predicates must be supported:

2316
2317
2318
2319

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

2320 Note that the query syntax allows for conjugation of simpler predicates into more
2321 complex queries as shown in the simple example above.

2322 **8.3.5 RegistryEntry Queries**

2323 Given the central role played by the `RegistryEntry` interface in RIM, the schema
2324 for the SQL query defines a special view called `RegistryEntry` that allows doing a
2325 polymorphic query against all `RegistryEntry` instances regardless of their actual
2326 concrete type or table name.

2327 The following example is the same as section 8.3.1.2 except that it is applied
2328 against all `RegistryEntry` instances rather than just `ExtrinsicObject` instances. The
2329 result set will include `id` for all qualifying `RegistryEntry` instances whose name
2330 contains the word 'Acme' and that have a version greater than 1.3.

2331
2332
2333

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

2334 **8.3.6 Classification Queries**

2335 This section describes the various classification related queries that must be
2336 supported.

2337 **8.3.6.1 Identifying ClassificationNodes**

2338 Like all objects in [RIM], ClassificationNodes are identified by their ID. However,
2339 they may also be identified as a path attribute that specifies an xpath expression
2340 from a root classification node to the specified classification node in the XML
2341 document that would represent the ClassificationNode tree including the said
2342 ClassificationNode.

2343 **8.3.6.2 Getting Root Classification Nodes**

2344 To get the collection of root ClassificationNodes the following query predicate
2345 must be supported:

```
2346 SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

2347 The above query returns all ClassificationNodes that have their parent attribute
2348 set to null. Note that the above query may also specify a predicate on the name if
2349 a specific root ClassificationNode is desired.

2350 **8.3.6.3 Getting Children of Specified ClassificationNode**

2351 To get the children of a ClassificationNode given the ID of that node the following
2352 style of query must be supported:

```
2353 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

2354 The above query returns all ClassificationNodes that have the node specified by
2355 ID as their parent attribute.

2356 **8.3.6.4 Getting Objects Classified By a ClassificationNode**

2357 To get the collection of ExtrinsicObjects classified by specified
2358 ClassificationNodes the following style of query must be supported:

```
2359 SELECT id FROM ExtrinsicObject
2360 WHERE
2361     id IN (SELECT classifiedObject FROM Classification
2362           WHERE
2363             classificationNode IN (SELECT id FROM ClassificationNode
2364                                   WHERE path = '/Geography/Asia/Japan'))
2365 AND
2366     id IN (SELECT classifiedObject FROM Classification
2367           WHERE
2368             classificationNode IN (SELECT id FROM ClassificationNode
2369                                   WHERE path = '/Industry/Automotive'))
```

2370 The above query gets the collection of ExtrinsicObjects that are classified by the
2371 Automotive Industry and the Japan Geography. Note that according to the
2372 semantics defined for GetClassifiedObjectsRequest, the query will also contain
2373 any objects that are classified by descendents of the specified
2374 ClassificationNodes.

2375 8.3.6.5 Getting ClassificationNodes That Classify an Object

2376 To get the collection of ClassificationNodes that classify a specified Object the
2377 following style of query must be supported:

```
2378 SELECT id FROM ClassificationNode  
2379 WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

2380 8.3.7 Association Queries

2381 This section describes the various Association related queries that must be
2382 supported.

2383 8.3.7.1 Getting All Association With Specified Object As Its Source

2384 To get the collection of Associations that have the specified Object as its source,
2385 the following query must be supported:

```
2386 SELECT id FROM Association WHERE sourceObject = <id>
```

2387 8.3.7.2 Getting All Association With Specified Object As Its Target

2388 To get the collection of Associations that have the specified Object as its target,
2389 the following query must be supported:

```
2390 SELECT id FROM Association WHERE targetObject = <id>
```

2391 8.3.7.3 Getting Associated Objects Based On Association Attributes

2392 To get the collection of Associations that have specified Association attributes,
2393 the following queries must be supported:

2394 Select Associations that have the specified name.

```
2395 SELECT id FROM Association WHERE name = <name>
```

2396 Select Associations that have the specified source role name.

```
2397 SELECT id FROM Association WHERE sourceRole = <roleName>
```

2398 Select Associations that have the specified target role name.

```
2399 SELECT id FROM Association WHERE targetRole = <roleName>
```

2400 Select Associations that have the specified association type, where association
2401 type is a string containing the corresponding field name described in [RIM].

```
2402 SELECT id FROM Association WHERE  
2403 associationType = <associationType>
```

2404 8.3.7.4 Complex Association Queries

2405 The various forms of Association queries may be combined into complex
2406 predicates. The following query selects Associations from an object with a
2407 specified id, that have the sourceRole "buysFrom" and targetRole "sellsTo":

```
2408 SELECT id FROM Association WHERE  
2409 sourceObject = <id> AND  
2410 sourceRole = 'buysFrom' AND  
2411 sourceRole = 'sellsTo'
```

2412 8.3.8 Package Queries

2413 To find all Packages that a specified ExtrinsicObject belongs to, the following
2414 query is specified:

```
2415 SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

2416 8.3.8.1 Complex Package Queries

2417 The following query gets all Packages that a specified object belongs to, that are
2418 not deprecated and where name contains "RosettaNet."

```
2419 SELECT id FROM Package WHERE  
2420 id IN (RegistryEntry_packages(<id>)) AND  
2421 name LIKE '%RosettaNet%' AND  
2422 status <> 'DEPRECATED'
```

2423 8.3.9 ExternalLink Queries

2424 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following
2425 query is specified:

```
2426 SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

2427 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the
2428 following query is specified:

```
2429 SELECT id From ExtrinsicObject WHERE id IN (RegistryEntry_linkedObjects(<id>))
```

2430 8.3.9.1 Complex ExternalLink Queries

2431 The following query gets all ExternalLinks that a specified ExtrinsicObject
2432 belongs to, that contain the word 'legal' in their description and have a URL for
2433 their externalURI.

```
2434 SELECT id FROM ExternalLink WHERE  
2435 id IN (RegistryEntry_externalLinks(<id>)) AND  
2436 description LIKE '%legal%' AND  
2437 externalURI LIKE '%http://%'
```

2438 8.3.10 Audit Trail Queries

2439 To get the complete collection of AuditableEvent objects for a specified
2440 ManagedObject, the following query is specified:

```
2441 SELECT id FROM AuditableEvent WHERE registryEntry = <id>
```

2442 8.3.11 Content Based Ad Hoc Queries

2443

2444 The Registry SQL query capability supports the ability to search for content
2445 based not only on metadata that catalogs the content but also the data contained
2446 within the content itself. For example it is possible for a client to submit a query
2447 that searches for all Collaboration Party Profiles that define a role named "seller"
2448 within a RoleName element in the CPP document itself.

2449 Currently content-based query capability is restricted to XML content.

2450 **8.3.11.1 Automatic Classification of XML Content**

2451 Content-based queries are indirectly supported through the existing classification
2452 mechanism supported by the Registry.

2453 A submitting organization may define logical indexes on any XML schema or
2454 DTD when it is submitted. An instance of such a logical index defines a link
2455 between a specific attribute or element node in an XML document tree and a
2456 ClassificationNode in a classification scheme within the registry.

2457 The registry utilizes this index to automatically classify documents that are
2458 instances of the schema at the time the document instance is submitted. Such
2459 documents are classified according to the data contained within the document
2460 itself.

2461 Such automatically classified content may subsequently be discovered by clients
2462 using the existing classification-based discovery mechanism of the Registry and
2463 the query facilities of the ObjectQueryManager.

2464 [Note]This approach is conceptually similar to the
2465 way databases support indexed retrieval. DBAs
2466 define indexes on tables in the schema. When
2467 data is added to the table, the data gets
2468 automatically indexed.

2469 **8.3.11.2 Index Definition**

2470 This section describes how the logical indexes are defined in the
2471 SubmittedObject element defined in the Registry DTD. The complete Registry
2472 DTD is specified in Appendix A.2.

2473 A SubmittedObject element for a schema or DTD may define a collection of
2474 ClassificationIndexes in a ClassificationIndexList optional element. The
2475 ClassificationIndexList is ignored if the content being submitted is not of the
2476 SCHEMA objectType.

2477 The ClassificationIndex element inherits the attributes of the base class Object in
2478 [RIM]. It then defines specialized attributes as follows:

- 2479 1. classificationNode: This attribute references a specific ClassificationNode
2480 by its ID.
- 2481 2. contentIdentifier: This attribute identifies a specific data element within the
2482 document instances of the schema using an XPATH path expression as
2483 defined by [XPT].

2484 **8.3.11.3 Example Of Index Definition**

2485 To define an index that automatically classifies a CPP based upon the roles
2486 defined within its RoleName elements, the following index must be defined on the
2487 CPP schema or DTD:

```

2488 <ClassificationIndex
2489     classificationNode='id-for-role-classification-scheme'
2490     contentIdentifier='/Role//RoleName'
2491 />
    
```

2492 **8.3.11.4 Example of Automatic Classification**

2493 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.”
 2494 When the CPP is submitted it will automatically be classified by two
 2495 ClassificationNodes named “buyer” and “seller” that are both children of the
 2496 ClassificationNode (e.g. a node named Role) specified in the classificationNode
 2497 attribute of the ClassificationIndex. Note that if either of the two
 2498 ClassificationNodes named “buyer” and “seller” did not previously exist, the
 2499 ObjectManager would automatically create these ClassificationNodes.

2500

2501 **8.3.12 Ad Hoc Query Request/Response**

2502 A client submits an ad hoc query to the ObjectQueryManager by sending an
 2503 AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that
 2504 defines a query in one of the supported Registry query mechanisms.

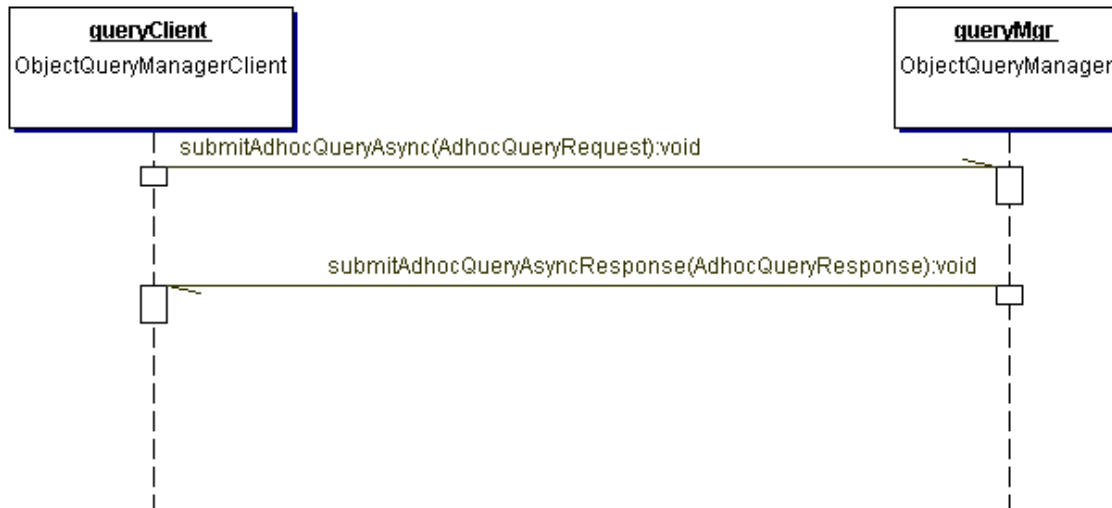
2505 The ObjectQueryManager sends an AdhocQueryResponse either synchronously
 2506 or asynchronously back to the client. The AdhocQueryResponse return a
 2507 collection of objects whose element type is in the set of element types
 2508 represented by the leaf nodes of the RegistryEntry hierarchy in [RIM].



2509

2510

Figure 15: Submit Ad Hoc Query Sequence Diagram



2511

2512

Figure 16: Submit Ad Hoc Query Asynchronous Sequence Diagram

2513

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

2514

2515

8.4 Content Retrieval

2516

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

2517

2518

2519

2520

2521

2522

2523

2524

2525

2526

2527

8.4.1 Retrieval of Registry Profile

2528

A special case of content retrieval is the retrieval of the RegistryProfile XML document. The RegistryProfile XML document is retrieved by specifying

2529

2530

A special id named "RegistryProfileID" as the value of the id attribute for the ObjectRef element in GetContentRequest.

2531

2532 **8.4.2 Identification Of Content Payloads**

2533 Since the GetContentResponse message may include several repository items
 2534 as additional payloads, it is necessary to have a way to identify each payload in
 2535 the message. To facilitate this identification, the Registry must do the following:

- 2536 ?? Use the ID for each RegistryEntry instance that describes the repository
- 2537 item as the DocumentLabel element in the DocumentReference for that
- 2538 object in the Manifest element of the ebXMLHeader.

2539 **8.4.3 GetContentResponse Message Structure**

2540 The following message fragment illustrates the structure of the
 2541 GetContentResponse Message that is returning a Collection of CPPs as a result
 2542 of a GetContentRequest that specified the IDs for the requested objects. Note
 2543 that the ID for each object retrieved in the message as additional payloads is
 2544 used as its DocumentLabel in the Manifest of the ebXMLHeader.

```

2545 ...
2546 --7250537.978150567601.JavaMail.najmi.irian
2547 ...
2548 ...
2549 <ebXMLHeader MessageType="Normal" Version="1.0">
2550   <Manifest>
2551     <DocumentReference>
2552       <DocumentLabel>GetContentsResponse</DocumentLabel>
2553       <DocumentId>6835fb:e3be512ac8:-8000</DocumentId>
2554     </DocumentReference>
2555     <DocumentReference>
2556       <DocumentLabel> ID for CPP content #1 </DocumentLabel>
2557       <DocumentId>...</DocumentId>
2558     </DocumentReference>
2559     <DocumentReference>
2560       <DocumentLabel> ID for CPP content #2 </DocumentLabel>
2561       <DocumentId>... </DocumentId>
2562     </DocumentReference>
2563   </Manifest>
2564   <Header>
2565     ...
2566   </Header>
2567 --7250537.978150567601.JavaMail.najmi.irian
2568 Content-Type: application/xml
2569 Content-Description: GetContentsResponse
2570 Content-ID: 6835fb:e3be512ac8:-7ffc
2571 Content-Length: 97
2572
2573 <?xml version="1.0" encoding="UTF-8"?>
2574 <GetContentsResponse />
2575
2576 --7250537.978150567601.JavaMail.najmi.irian
2577 Content-Type: application/xml
2578 Content-Description: ID for CPP content #1
2579 Content-ID: ...
2580 ...
2581 <CPP>
2582 ...
2583 </CPP>
2584 --7250537.978150567601.JavaMail.najmi.irian
2585 Content-Type: application/xml
2586 Content-Description: ID for CPP content #2
    
```



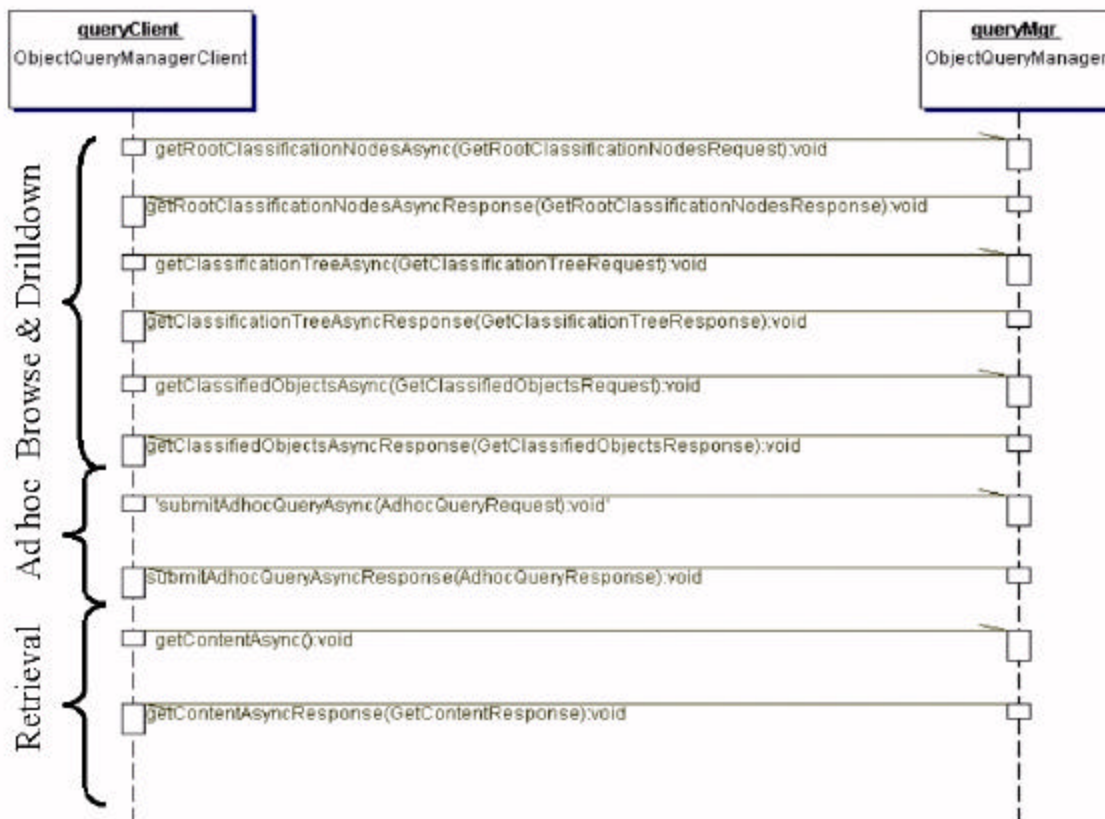
```

2587 Content-ID: ...
2588 ..
2589 <CPP>
2590 ..
2591 </CPP>
2592 --7250537.978150567601.JavaMail.najmi.irian--
    
```

2593

2594 8.5 Query And Retrieval: Typical Sequence

2595 The following diagram illustrates the use of both browse/drilldown and ad hoc
 2596 queries followed by a retrieval of content that was selected by the queries.



2597

2598

Figure 17: Typical Query and Retrieval Sequence

2599 9 Registry Security

2600 This chapter describes the security features of the ebXML Registry. It is assumed
 2601 that the reader is familiar with the security related classes in the Registry
 2602 information model as described in [RIM].

2603 In the current version of this specification, a minimalist approach has been
2604 specified for Registry security. The philosophy is that “Any *known* entity can
2605 publish content and *anyone* can view published content.” The Registry
2606 information model has been designed to allow more sophisticated security
2607 policies in future versions of this specification.

2608 **9.1 Integrity of Registry Content**

2609 It is assumed that most business registries do not have the resources to validate
2610 the veracity of the content submitted to them. The minimal integrity that the
2611 Registry must provide is to ensure that content submitted by a Submitting
2612 Organization (SO) is maintained in the Registry without any tampering either *en-*
2613 *route* or *within* the Registry. Furthermore, the Registry must make it possible to
2614 identify the SO for any Registry content unambiguously.

2615 **9.1.1 Message Payload Signature**

2616 Integrity of Registry content requires that all submitted content must be signed by
2617 the Registry client as defined by [SEC]. The signature on the submitted content
2618 ensures that:

2619 ?? The content has not been tampered with en-route or within the Registry.

2620 ?? The content’s veracity can be ascertained by its association with a
2621 specific submitting organization

2622 **9.2 Authentication**

2623 The Registry must be able to authenticate the identity of the Principal associated
2624 with client requests. Authentication is required to identify the ownership of
2625 content as well as to identify what “privileges” a Principal can be assigned with
2626 respect to the specific objects in the Registry.

2627 The Registry must perform Authentication on a per request basis. From a
2628 security point of view, all messages are independent and there is no concept of a
2629 session encompassing multiple messages or conversations. Session support
2630 may be added as an optimization feature in future versions of this specification.

2631 The Registry must implement a credential-based authentication mechanism
2632 based on digital certificates and signatures. The Registry uses the certificate DN
2633 from the signature to authenticate the user.

2634 **9.2.1 Message Header Signature**

2635 Message headers may be signed by the sending ebXML Messaging Service as
 2636 defined by [SEC]. Since this specification is not yet finalized, this version does
 2637 not require that the message header be signed. In the absence of a message
 2638 header signature, the payload signature is used to authenticate the identity of the
 2639 requesting client.

2640 **9.3 Confidentiality**

2641 **9.3.1 On-the-wire Message Confidentiality**

2642 It is suggested but not required that message payloads exchanged between
 2643 clients and the Registry be encrypted during transmission. Payload encryption
 2644 must abide by any restrictions set forth in [SEC].

2645 **9.3.2 Confidentiality of Registry Content**

2646 In the current version of this specification, there are no provisions for
 2647 confidentiality of Registry content. All content submitted to the Registry may be
 2648 discovered and read by *any* client. Therefore, the Registry must be able to
 2649 decrypt any submitted content after it has been received and prior to storing it in
 2650 its repository. This implies that the Registry and the client have an a priori
 2651 agreement regarding encryption algorithm, key exchange agreements, etc. This
 2652 service is not addressed in this specification.

2653 **9.4 Authorization**

2654 The Registry must provide an authorization mechanism based on the information
 2655 model defined in [RIM]. In this version of the specification the authorization
 2656 mechanism is based on a default Access Control Policy defined for a pre-defined
 2657 set of roles for Registry users. Future versions of this specification will allow for
 2658 custom Access Control Policies to be defined by the Submitting Organization.

2659 **9.4.1 Pre-defined Roles For Registry Users**

2660 The following roles must be pre-defined in the Registry:

Role	Description
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.

2661 **9.4.2 Default Access Control Policies**

2662 The Registry must create a default AccessControlPolicy object that grants the
 2663 default permissions to Registry users based upon their assigned role.

2664 The following table defines the Permissions granted by the Registry to the
 2665 various pre-defined roles for Registry users based upon the default
 2666 AccessControlPolicy.

2667

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

2668

2669 The following list summarizes the default role-based AccessControlPolicy:

- 2670 ?? The Registry must implement the default AccessControlPolicy and
- 2671 associate it with all Objects in the Registry
- 2672 ?? Anyone can publish content, but needs to be authenticated
- 2673 ?? Anyone can access the content without requiring authentication
- 2674 ?? The ContentOwner has access to all methods for Registry Objects owned
- 2675 by them
- 2676 ?? The RegistryAdministrator has access to all methods on all Registry
- 2677 Objects
- 2678 ?? Unauthenticated clients can access all read-only (getXXX) methods
- 2679 ?? At the time of content submission, the Registry must assign the default
- 2680 ContentOwner role to the Submitting Organization (SO) as authenticated
- 2681 by the credentials in the submission message. In the current version of
- 2682 this specification, it will be the DN as identified by the certificate
- 2683 ?? Clients that browse the Registry need not use certificates. The Registry
- 2684 must assign the default RegistryGuest role to such clients.

2685 Appendix A Schemas and DTD Definitions

2686 The following are definitions for the various ebXML Message payloads described
2687 in this document.

2688 A.1 ebXMLError Message DTD

2689 See [ERR] for ebXMLError Message DTD.

2690 A.2 ebXML Registry DTD

```

2691 <?xml version="1.0" encoding="UTF-8"?>
2692 <!-- Begin information model mapping. -->
2693 <!ENTITY % errorSchema SYSTEM "ebXMLError.dtd">
2694 %errorSchema;
2695
2696 <!--
2697 ObjectAttributes are attributes from the Object interface in RIM.
2698
2699 id may be empty. If specified it may be in urn:uuid format or be in some arbitrary format.
2700 If id is empty registry must generate globally unique id.
2701 If id is provided and in proper UUID syntax (starts with urn:uuid:) registry will honour
2702 it
2703 If id is provided and is not in proper UUID syntax then it is used for linkage within
2704 document
2705 and is ignored by the registry. In this case the registry generates a UUID for id
2706 attribute.
2707
2708 id must not be null when object is being retrieved from the registry.
2709 -->
2710 <!ENTITY % ObjectAttributes "
2711     id          ID          #IMPLIED
2712     name        CDATA      #IMPLIED
2713     description CDATA      #IMPLIED
2714 ">
2715
2716 <!--
2717 Use as a proxy for an Object that is in the registry already.
2718 Specifies the id attribute of the object in the registry as its id attribute.
2719 id attribute in ObjectAttributes is exactly the same syntax and semantics as id attribute
2720 in Object.
2721 -->
2722 <!ELEMENT ObjectRef EMPTY>
2723 <!ATTLIST ObjectRef
2724     id ID #IMPLIED
2725 >
2726 <!ELEMENT ObjectRefList (ObjectRef)*>
2727
2728 <!--
2729 RegistryEntryAttributes are attributes from the RegistryEntry interface in RIM.
2730 It inherits ObjectAttributes
2731 -->
2732 <!ENTITY % RegistryEntryAttributes "%ObjectAttributes;
2733     majorVersion CDATA '1'
2734     minorVersion CDATA '0'
2735
2736     status        CDATA #IMPLIED
2737     userVersion   CDATA #IMPLIED
2738     stability     CDATA 'Dynamic'
2739     expirationDate CDATA #IMPLIED">
2740

```

```

2741
2742 <!ELEMENT RegistryEntry (SlotList?)>
2743 <!ATTLIST RegistryEntry
2744     %RegistryEntryAttributes;
2745 >
2746 <!ELEMENT Value (#PCDATA)>
2747 <!ELEMENT ValueList (Value*)>
2748 <!ELEMENT Slot (ValueList?)>
2749 <!ATTLIST Slot
2750     name CDATA #REQUIRED
2751     slotType CDATA #IMPLIED
2752 >
2753 <!ELEMENT SlotList (Slot*)>
2754
2755 <!--
2756 ExtrinsicObject are attributes from the ExtrinsicObject interface in RIM.
2757 It inherits RegistryEntryAttributes
2758 -->
2759
2760 <!ELEMENT ExtrinsicObject (ClassificationIndexList?)>
2761 <!ATTLIST ExtrinsicObject
2762     %RegistryEntryAttributes;
2763     contentURI CDATA #REQUIRED
2764     mimeType CDATA #IMPLIED
2765     objectType CDATA #REQUIRED
2766     opaque (true | false) "false"
2767 >
2768
2769 <!--
2770 A ClassificationIndexList is specified on ExtrinsicObjects of objectType 'Schema' to
2771 define
2772 an automatic Classification of instance objects of the schema using
2773 the specified classificationNode as parent and a ClassificationNode
2774 created or selected by the object content as selected by the contentIdentifier
2775 -->
2776 <!ELEMENT ClassificationIndex EMPTY>
2777 <!ATTLIST ClassificationIndex
2778     %ObjectAttributes;
2779     classificationNode IDREF #REQUIRED
2780     contentIdentifier CDATA #REQUIRED
2781 >
2782
2783 <!-- ClassificationIndexList contains new ClassificationIndexes -->
2784 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
2785
2786 <!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
2787
2788 <!-- Leaf classes that reflect the concrete classes in RIM -->
2789 <!ELEMENT RegistryEntryList (Association | Classification | ClassificationNode | Package
2790 | ExternalLink | ExternalIdentifier | Organization | ExtrinsicObject | ObjectRef)*>
2791
2792
2793 <!--
2794 An ExternalLink specifies a link from a RegistryEntry and an external URI
2795 -->
2796 <!ELEMENT ExternalLink EMPTY>
2797 <!ATTLIST ExternalLink
2798     %IntrinsicObjectAttributes;
2799     externalURI CDATA #IMPLIED
2800 >
2801
2802 <!--
2803 An ExternalIdentifier provides an identifier for a RegistryEntry
2804
2805 The value is the value of the identifier (e.g. the social security number)
2806 -->
2807 <!ELEMENT ExternalIdentifier EMPTY>
2808 <!ATTLIST ExternalIdentifier

```

```

2809     %IntrinsicObjectAttributes;
2810     value CDATA #REQUIRED
2811 >
2812
2813 <!--
2814 An Association specifies references to two previously submitted
2815 registry entrys.
2816
2817 The sourceObject is id of the sourceObject in association
2818 The targetObject is id of the targetObject in association
2819 -->
2820 <!ELEMENT Association EMPTY>
2821 <!ATTLIST Association
2822     %IntrinsicObjectAttributes;
2823     fromLabel CDATA #IMPLIED
2824     toLabel CDATA #IMPLIED
2825     associationType CDATA #REQUIRED
2826     bidirection (true | false) "false"
2827     sourceObject IDREF #REQUIRED
2828     targetObject IDREF #REQUIRED
2829 >
2830
2831 <!--
2832 A Classification specifies references to two registry entrys.
2833
2834 The classifiedObject is id of the Object being classified.
2835 The classificationNode is id of the ClassificationNode classying the object
2836 -->
2837 <!ELEMENT Classification EMPTY>
2838 <!ATTLIST Classification
2839     %IntrinsicObjectAttributes;
2840     classifiedObject IDREF #REQUIRED
2841     classificationNode IDREF #REQUIRED
2842 >
2843
2844 <!--
2845 A Package is a named collection of objects.
2846 -->
2847 <!ELEMENT Package EMPTY>
2848 <!ATTLIST Package
2849     %IntrinsicObjectAttributes;
2850 >
2851
2852 <!-- Attributes inherited by various types of telephone number elements -->
2853 <!ENTITY % TelephoneNumberAttributes " areaCode CDATA #REQUIRED
2854     contryCode CDATA #REQUIRED
2855     extension CDATA #IMPLIED
2856     number CDATA #REQUIRED
2857     url CDATA #IMPLIED">
2858 <!ELEMENT TelephoneNumber EMPTY>
2859 <!ATTLIST TelephoneNumber
2860     %TelephoneNumberAttributes;
2861 >
2862 <!ELEMENT FaxNumber EMPTY>
2863 <!ATTLIST FaxNumber
2864     %TelephoneNumberAttributes;
2865 >
2866 <!ELEMENT PagerNumber EMPTY>
2867 <!ATTLIST PagerNumber
2868     %TelephoneNumberAttributes;
2869 >
2870 <!--
2871 <!ELEMENT MobileTelephoneNumber EMPTY>
2872 <!ATTLIST MobileTelephoneNumber
2873     %TelephoneNumberAttributes;
2874 >
2875 <!--
2876 PostalAddress -->

```

```

2877 <!ELEMENT PostalAddress EMPTY>
2878 <!ATTLIST PostalAddress
2879     city CDATA #REQUIRED
2880     country CDATA #REQUIRED
2881     postalCode CDATA #REQUIRED
2882     state CDATA #REQUIRED
2883     street CDATA #REQUIRED
2884 >
2885 <!-- PersonName -->
2886 <!ELEMENT PersonName EMPTY>
2887 <!ATTLIST PersonName
2888     firstName CDATA #REQUIRED
2889     middleName CDATA #REQUIRED
2890     lastName CDATA #REQUIRED
2891 >
2892
2893 <!-- Organization -->
2894 <!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
2895 <!ATTLIST Organization
2896     %IntrinsicObjectAttributes;
2897     parent IDREF #IMPLIED
2898     primaryContact IDREF #REQUIRED
2899 >
2900
2901 <!ELEMENT User (PersonName, PostalAddress, TelephoneNumber, MobileTelephoneNumber?,
2902 FaxNumber?, PagerNumber?)>
2903 <!ATTLIST User
2904     %ObjectAttributes;
2905     organization IDREF #IMPLIED
2906     email CDATA #IMPLIED
2907     url CDATA #IMPLIED
2908 >
2909
2910 <!ELEMENT AuditableEvent EMPTY>
2911 <!ATTLIST AuditableEvent
2912     %ObjectAttributes;
2913     eventType CDATA #REQUIRED
2914     registryEntry IDREF #REQUIRED
2915     timestamp CDATA #REQUIRED
2916     user IDREF #REQUIRED
2917 >
2918 <!--
2919 ClassificationNode is used to submit a Classification tree to the Registry.
2920
2921 parent is the id to the parent node. code is an optional code value for a
2922 ClassificationNode
2923 often defined by an external taxonomy (e.g. NAICS)
2924 -->
2925 <!ELEMENT ClassificationNode EMPTY>
2926 <!ATTLIST ClassificationNode
2927     %IntrinsicObjectAttributes;
2928     parent IDREF #IMPLIED
2929     code CDATA #IMPLIED
2930 >
2931
2932 <!--
2933 End information model mapping.
2934
2935 Begin Registry Services Interface
2936 -->
2937 <!ELEMENT RequestAcceptedResponse EMPTY>
2938 <!ATTLIST RequestAcceptedResponse
2939     xml:lang NMTOKEN #REQUIRED
2940     interfaceId CDATA #REQUIRED
2941     requestMessage CDATA #REQUIRED
2942     actionId CDATA #REQUIRED
2943 >
2944 <!--

```



```

2945 The SubmittedObject provides meta data for submitted object
2946 Note object being submitted is in a separate document that is not
2947 in this DTD.
2948 -->
2949 <!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
2950 <!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
2951 <!-- Only need name in Slot within SlotList -->
2952 <!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
2953 <!--
2954 The ObjectRefList is the list of
2955 refs to the registry entrys being approved.
2956 -->
2957 <!ELEMENT ApproveObjectsRequest (ObjectRefList)>
2958 <!--
2959 The ObjectRefList is the list of
2960 refs to the registry entrys being deprecated.
2961 -->
2962 <!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
2963 <!--
2964 The ObjectRefList is the list of
2965 refs to the registry entrys being removed
2966 -->
2967 <!ELEMENT RemoveObjectsRequest (ObjectRefList)>
2968 <!ATTLIST RemoveObjectsRequest
2969     deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
2970 >
2971 <!ELEMENT GetRootClassificationNodesRequest EMPTY>
2972 <!--
2973 The namePattern follows SQL-92 syntax for the pattern specified in
2974 LIKE clause. It allows for selecting only those root nodes that match
2975 the namePattern. The default value of '*' matches all root nodes.
2976 -->
2977 <!ATTLIST GetRootClassificationNodesRequest
2978     namePattern CDATA "*"
2979 >
2980 <!--
2981 The response includes one or more ClassificationNodes
2982 -->
2983 <!ELEMENT GetRootClassificationNodesResponse ((ClassificationNode+) | ebXMLError)>
2984 <!--
2985 Get the classification tree under the ClassificationNode specified parentRef.
2986
2987 If depth is 1 just fetch immediate child
2988 nodes, otherwise fetch the descendant tree upto the specified depth level.
2989 If depth is 0 that implies fetch entire sub-tree
2990 -->
2991 <!ELEMENT GetClassificationTreeRequest EMPTY>
2992 <!ATTLIST GetClassificationTreeRequest
2993     parent CDATA #REQUIRED
2994     depth CDATA "1"
2995 >
2996 <!--
2997 The response includes one or more ClassificationNodes which includes only
2998 immediate ClassificationNode children nodes if depth attribute in
2999 GetClassificationTreeRequest was 1, otherwise the decendent nodes
3000 upto specified depth level are returned.
3001 -->
3002 <!ELEMENT GetClassificationTreeResponse ((ClassificationNode+) | ebXMLError)>
3003 <!--
3004 Get refs to all registry entrys that are classified by all the
3005 ClassificationNodes specified by ObjectRefList.
3006 Note this is an implicit logical AND operation
3007 -->
3008 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
3009 <!--
3010 objectType attribute can specify the type of objects that the registry
3011 client is interested in, that is classified by this ClassificationNode.
3012 It is a String that matches a choice in the type attribute of ExtrinsicObject.

```

```

3013 The default value of '*' implies that client is interested in all types
3014 of registry entries that are classified by the specified ClassificationNode.
3015 -->
3016 <!--
3017 The response includes a RegistryEntryList which has zero or more
3018 RegistryEntries that are classified by the ClassificationNodes
3019 specified in the ObjectRefList in GetClassifiedObjectsRequest.
3020 -->
3021 <!ELEMENT GetClassifiedObjectsResponse (RegistryEntryList | ebXMLError)>
3022 <!--
3023 An Ad hoc query request specifies a query string as defined by [RS] in the queryString
3024 attribute
3025 -->
3026 <!ELEMENT AdhocQueryRequest (FilterQuery | GetRegistryEntry | GetRepositoryItem |
3027 SQLQuery)>
3028 <!ELEMENT SQLQuery (#PCDATA)>
3029 <!--
3030 The response includes a RegistryEntryList which has zero or more
3031 RegistryEntries that match the query specified in AdhocQueryRequest.
3032 -->
3033 <!ELEMENT AdhocQueryResponse (RegistryEntryList | FilterQueryResult |
3034 GetRegistryEntryResult | GetRepositoryItemResult | ebXMLError)>
3035 <!--
3036 Gets the actual content (not metadata) specified by the ObjectRefList
3037 -->
3038 <!ELEMENT GetContentRequest (ObjectRefList)>
3039 <!--
3040 The GetObjectsResponse will have no sub-elements if there were no errors.
3041 The actual contents will be in the other payloads of the message.
3042 If any errors were encountered the message will contain the ebXMLError and
3043 the content payloads will be empty.
3044 -->
3045 <!ELEMENT GetContentResponse (ebXMLError?)>
3046 <!--
3047 Describes the capability profile for the registry and what optional features
3048 are supported
3049 -->
3050 <!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
3051 <!ATTLIST RegistryProfile
3052     version CDATA #REQUIRED
3053 >
3054
3055 <!ELEMENT OptionalFeaturesSupported EMPTY>
3056 <!ATTLIST OptionalFeaturesSupported
3057     sqlQuery (true | false) "false"
3058     xQuery (true | false) "false"
3059 >
3060 <!-- Begin FilterQuery DTD -->
3061 <!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery | ClassificationNodeQuery
3062 | RegistryPackageQuery | OrganizationQuery)>
3063 <!ELEMENT FilterQueryResult (RegistryEntryQueryResult | AuditableEventQueryResult |
3064 ClassificationNodeQueryResult | RegistryPackageQueryResult | OrganizationQueryResult)>
3065 <!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
3066 <!ELEMENT RegistryEntryView EMPTY>
3067 <!ATTLIST RegistryEntryView
3068     objectURN CDATA #REQUIRED
3069     contentURL CDATA #IMPLIED
3070     objectID CDATA #IMPLIED
3071 >
3072 <!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
3073 <!ELEMENT AuditableEventView EMPTY>
3074 <!ATTLIST AuditableEventView
3075     objectID CDATA #REQUIRED
3076     timestamp CDATA #REQUIRED
3077 >
3078 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
3079 <!ELEMENT ClassificationNodeView EMPTY>
3080 <!ATTLIST ClassificationNodeView

```

```

3081     objectURN CDATA #REQUIRED
3082     contentURL CDATA #IMPLIED
3083     objectID CDATA #IMPLIED
3084 >
3085 <!--ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
3086 <!--ELEMENT RegistryPackageView EMPTY>
3087 <!--ATTLIST RegistryPackageView
3088     objectURN CDATA #REQUIRED
3089     contentURL CDATA #IMPLIED
3090     objectID CDATA #IMPLIED
3091 >
3092 <!--ELEMENT OrganizationQueryResult (OrganizationView*)>
3093 <!--ELEMENT OrganizationView EMPTY>
3094 <!--ATTLIST OrganizationView
3095     orgURN CDATA #REQUIRED
3096     contactURL CDATA #IMPLIED
3097     objectID CDATA #IMPLIED
3098 >
3099 <!--ELEMENT StatusResult (Success | (Exception | Warning)+)>
3100 <!--ELEMENT Success EMPTY>
3101 <!--ELEMENT Exception (#PCDATA)>
3102 <!--ATTLIST Exception
3103     code CDATA #REQUIRED
3104 >
3105 <!--ELEMENT Warning (#PCDATA)>
3106 <!--ATTLIST Warning
3107     code CDATA #REQUIRED
3108 >
3109 <!--ELEMENT RegistryEntryQuery (RegistryEntryFilter?, AsSourceAssociation*,
3110 AsTargetAssociation*, RegistryEntryClassification*, SubmittingOrgFilter?,
3111 ResponsibleOrgFilter?, ExternalLinkFilter*, RegistryEntryAuditableEvent*)>
3112 <!--ELEMENT AsSourceAssociation (AssociationFilter?, RegistryEntryFilter?)>
3113 <!--ELEMENT AsTargetAssociation (AssociationFilter?, RegistryEntryFilter?)>
3114 <!--ELEMENT RegistryEntryClassification (ClassificationFilter?, ClassificationNodeFilter?)>
3115 <!--ELEMENT SubmittingOrgFilter (OrganizationFilter?, ContactFilter?)>
3116 <!--ELEMENT ResponsibleOrgFilter (OrganizationFilter?, ContactFilter?)>
3117 <!--ELEMENT RegistryEntryAuditableEvent (AuditableEventFilter?, UserFilter?,
3118 OrganizationFilter?)>
3119 <!--ELEMENT AuditableEventQuery (AuditableEventFilter?, RegistryEntryQuery*, UserFilter?,
3120 OrganizationQuery?)>
3121 <!--ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?, ClassifiesRegistryEntry*,
3122 HasParentNode?, HasSubnode*)>
3123 <!--ELEMENT ClassifiesRegistryEntry (ClassificationFilter?, RegistryEntryQuery?)>
3124 <!--ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
3125 <!--ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
3126 <!--ELEMENT RegistryPackageQuery (PackageFilter?, PackageHasMember*)>
3127 <!--ELEMENT PackageHasMember (RegistryEntryQuery?)>
3128 <!--ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsEntry*, HasParentOrganization?,
3129 InvokesEvent*, ContactFilter*)>
3130 <!--ELEMENT SubmitsEntry (RegistryEntryQuery?)>
3131 <!--ELEMENT HasParentOrganization (OrganizationFilter?, HasParentOrganization?)>
3132 <!--ELEMENT InvokesEvent (UserFilter?, AuditableEventFilter?, RegistryEntryQuery?)>
3133 <!--ELEMENT GetRegistryEntry (RegistryEntryQuery, WithClassifications?,
3134 WithAsSourceAssociations?, WithAsTargetAssociations?, WithAuditableEvents?,
3135 WithExternalLinks?)>
3136 <!--ELEMENT WithClassifications (ClassificationFilter?)>
3137 <!--ELEMENT WithAsSourceAssociations (AssociationFilter?)>
3138 <!--ELEMENT WithAsTargetAssociations (AssociationFilter?)>
3139 <!--ELEMENT WithAuditableEvents (AuditableEventFilter?)>
3140 <!--ELEMENT WithExternalLinks (ExternalLinkFilter?)>
3141 <!--ELEMENT GetRegistryEntryResult (RegistryEntryMetadata*, StatusResult)>
3142 <!--ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*, AsSourceAssociations?,
3143 AsTargetAssociations?, AuditableEvent*, ExternalLink*)>
3144 <!--ELEMENT AsSourceAssociations (Association*)>
3145 <!--ELEMENT AsTargetAssociations (Association*)>
3146 <!--ELEMENT GetRepositoryItem (RegistryEntryQuery, RecursiveAssociationOption?,
3147 WithShortDescription?)>
3148 <!--ELEMENT RecursiveAssociationOption (AssociationRole+)>

```

```

3149 <!-- ATTLIST RecursiveAssociationOption
3150     depthLimit CDATA #IMPLIED
3151 >
3152 <!-- ELEMENT AssociationRole EMPTY>
3153 <!-- ATTLIST AssociationRole
3154     role CDATA #REQUIRED
3155 >
3156 <!-- ELEMENT WithShortDescription EMPTY>
3157 <!-- ELEMENT GetRepositoryItemResult (RepositoryItem*, StatusResult)>
3158 <!-- ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject |
3159 ExternalItem)>
3160 <!-- ATTLIST RepositoryItem
3161     identifier CDATA #REQUIRED
3162     name CDATA #REQUIRED
3163     repositoryURL CDATA #REQUIRED
3164     objectType CDATA #REQUIRED
3165     status CDATA #REQUIRED
3166     stability CDATA #REQUIRED
3167     description CDATA #IMPLIED
3168 >
3169 <!-- ELEMENT RegistryPackage EMPTY>
3170 <!-- ELEMENT WithdrawnObject EMPTY>
3171 <!-- ELEMENT ExternalItem EMPTY>
3172 <!-- ELEMENT ObjectFilter (Clause)>
3173 <!-- ELEMENT RegistryEntryFilter (Clause)>
3174 <!-- ELEMENT IntrinsicObjectFilter (Clause)>
3175 <!-- ELEMENT ExtrinsicObjectFilter (Clause)>
3176 <!-- ELEMENT PackageFilter (Clause)>
3177 <!-- ELEMENT OrganizationFilter (Clause)>
3178 <!-- ELEMENT ContactFilter (Clause)>
3179 <!-- ELEMENT ClassificationNodeFilter (Clause)>
3180 <!-- ELEMENT AssociationFilter (Clause)>
3181 <!-- ELEMENT ClassificationFilter (Clause)>
3182 <!-- ELEMENT ExternalLinkFilter (Clause)>
3183 <!-- ELEMENT AuditableEventFilter (Clause)>
3184 <!-- ELEMENT UserFilter (Clause)>
3185 <!-- ELEMENT Clause (SimpleClause | CompoundClause)>
3186 <!-- ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
3187 <!-- ATTLIST SimpleClause
3188     leftArgument CDATA #REQUIRED
3189 >
3190 <!-- ELEMENT CompoundClause (Clause, Clause+)>
3191 <!-- ATTLIST CompoundClause
3192     connectivePredicate (And | Or) #REQUIRED
3193 >
3194 <!-- ELEMENT BooleanClause EMPTY>
3195 <!-- ATTLIST BooleanClause
3196     booleanPredicate (true | false) #REQUIRED
3197 >
3198 <!-- ELEMENT RationalClause (IntClause | FloatClause)>
3199 <!-- ATTLIST RationalClause
3200     logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
3201 >
3202 <!-- ELEMENT IntClause (#PCDATA)>
3203 <!-- ATTLIST IntClause
3204     e-dtype NMTOKEN #FIXED "int"
3205 >
3206 <!-- ELEMENT FloatClause (#PCDATA)>
3207 <!-- ATTLIST FloatClause
3208     e-dtype NMTOKEN #FIXED "float"
3209 >
3210 <!-- ELEMENT StringClause (#PCDATA)>
3211 <!-- ATTLIST StringClause
3212     stringPredicate (contains | -contains | startswith | -startswith | endswith | -
3213 endswith) #REQUIRED
3214 >
3215 <!-- End FilterQuery DTD -->
3216 <!--

```

```

3217 The contrived root node
3218 -->
3219 <!ELEMENT RootElement (RequestAcceptedResponse | ebXMLError | SubmitObjectsRequest |
3220 ApproveObjectsRequest | DeprecateObjectsRequest | RemoveObjectsRequest |
3221 GetRootClassificationNodesRequest | GetRootClassificationNodesResponse |
3222 GetClassificationTreeRequest | GetClassificationTreeResponse |
3223 GetClassifiedObjectsRequest | GetClassifiedObjectsResponse | AdhocQueryRequest |
3224 AdhocQueryResponse | GetContentRequest | GetContentResponse | AddSlotsRequest |
3225 RemoveSlotsRequest | RegistryProfile)>

```

3226 **Appendix B Interpretation of UML Diagrams**

3227 This section describes in *abstract terms* the conventions used to define ebXML
3228 business process description in UML.

3229 **B.1 UML Class Diagram**

3230 A UML class diagram is used to describe the Service Interfaces (as defined by
3231 [CPA]) required to implement an ebXML Registry Services and clients. See
3232 Figure 2 on page 15 for an example. The UML class diagram contains:

3233

- 3234 1. A collection of UML interfaces where each interface represents a Service
3235 Interface for a Registry service.
- 3236 2. Tabular description of methods on each interface where each method
3237 represents an Action (as defined by [CPA]) within the Service Interface
3238 representing the UML interface.
- 3239 3. Each method within a UML interface specifies one or more parameters,
3240 where the type of each method argument represents the ebXML message
3241 type that is exchanged as part of the Action corresponding to the method.
3242 Multiple arguments imply multiple payload documents within the body of
3243 the corresponding ebXML message.

3244 **B.2 UML Sequence Diagram**

3245 A UML sequence diagram is used to specify the business protocol representing
3246 the interactions between the UML interfaces for a Registry specific ebXML
3247 business process. A UML sequence diagram provides the necessary information
3248 to determine the sequencing of messages, request to response association as
3249 well as request to error response association as described by [CPA].

3250 Each sequence diagram shows the sequence for a specific conversation protocol
3251 as method calls from the requestor to the responder. Method invocation may be
3252 synchronous or asynchronous based on the UML notation used on the arrow-
3253 head for the link. A half arrow-head represents asynchronous communication. A
3254 full arrow-head represents synchronous communication.

3255 Each method invocation may be followed by a response method invocation from
3256 the responder to the requestor to indicate the ResponseName for the previous
3257 Request. Possible error response is indicated by a conditional response method
3258 invocation from the responder to the requestor. See Figure 4 on page 22 for an
3259 example.

3260 **Appendix C SQL Query**

3261 **C.1 SQL Query Syntax Specification**

3262 This section specifies the rules that define the SQL Query syntax as a subset of
3263 SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in
3264 [SQL/PSM].

3265

- 3266 1. The SQL query syntax conforms to the <query specification>, modulo the
3267 restrictions identified below
- 3268 2. A <select list> may contain at most one <select sublist>
- 3269 3. In a <select list> must be is a single column whose data type is UUID,
3270 from the table in the <from clause>,
- 3271 4. A <derived column> may not have an <as clause>
- 3272 5. <table expression> does not contain the optional <group by clause> and
3273 <having clause> clauses.
- 3274 6. A <table reference> can only consist of <table name> and <correlation
3275 name>
- 3276 7. A <table reference> does not have the optional AS between <table name>
3277 and <correlation name>
- 3278 8. There can only be one <table reference> in the <from clause>
- 3279 9. Restricted use of sub-queries is allowed by the syntax as follows. The <in
3280 predicate> allows for the right hand side of the <in predicate> to be limited
3281 to a restricted <query specification> as defined above.
- 3282 10. A <search condition> within the <where clause> may not include a <query
3283 expression>.
- 3284
- 3285 11. The SQL query syntax allows for the use of <sql invoked routines>
3286 invocation from [SQL/PSM] as the RHS of the <in predicate>.

3287

C.2 Non-Normative BNF for Query Syntax Grammar

3288

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

3289

3290

3291

3292

3293

3294

3295

3296

3297

3298

3299

3300

3301

3302

3303

3304

3305

3306

3307

3308

3309

3310

3311

3312

3313

3314

3315

3316

3317

3318

3319

3320

3321

3322

3323

3324

3325

3326

3327

3328

3329

3330

3331

3332

3333

3334

3335

3336

3337

3338

3339

3340

3341

3342

3343

3344

3345

3346

3347

3348

3349

```

/*****
 * The Registry Query (Subset of SQL-92) grammar starts here
 *****/
RegistryQuery = SQLSelect [ ";" ]

SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]

SQLSelectCols = ID

SQLTableList = SQLTableRef

SQLTableRef = ID

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *

SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *

SQLNotExpr = [ "NOT" ] SQLCompareExpr

SQLCompareExpr =
    ( SQLColRef "IS" ) SQLIsClause
  | SQLSumExpr [ SQLCompareExprRight ]

SQLCompareExprRight =
    SQLLikeClause
  | SQLInClause
  | SQLCompareOp SQLSumExpr

SQLCompareOp =
    "="
  | "<>"
  | ">"
  | ">="
  | "<"
  | "<="

SQLInClause = [ "NOT" ] "IN" "(" SQLValueList ")"

SQLValueList = SQLValueElement ( "," SQLValueElement ) *

SQLValueElement = "NULL" | SQLSelect

SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"

SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern

SQLPattern = STRING_LITERAL

SQLLiteral =
    STRING_LITERAL
  | INTEGER_LITERAL
  | FLOATING_POINT_LITERAL

```

```

3350
3351 SQLColRef = SQLLvalue
3352
3353 SQLLvalue = SQLLvalueTerm
3354
3355 SQLLvalueTerm = ID ( "." ID ) *
3356
3357 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr ) *
3358
3359 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr ) *
3360
3361 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
3362
3363 SQLTerm = "(" SQLOrExpr ")"
3364 | SQLColRef
3365 | SQLLiteral
3366
3367 INTEGER_LITERAL = ([ "0"-"9" ] ) +
3368
3369 FLOATING_POINT_LITERAL =
3370 ([ "0"-"9" ] ) + "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3371 | "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3372 | ([ "0"-"9" ] ) + EXPONENT
3373 | ([ "0"-"9" ] ) + ( EXPONENT ) ?
3374
3375 EXPONENT = [ "e", "E" ] ( [ "+" , "-" ] ) ? ( [ "0"-"9" ] ) +
3376
3377 STRING_LITERAL: "'" (~[ "'" ] ) * ( '"' (~[ '"' ] ) * ) * "'"
3378
3379 ID = ( <LETTER> ) + ( "_" | "$" | "#" | <DIGIT> | <LETTER> ) *
3380 LETTER = [ "A"-"Z", "a"-"z" ]
3381 DIGIT = [ "0"-"9" ]

```

C.3 Relational Schema For SQL Queries

```

3382
3383 --SQL Load file for creating the ebXML Registry tables
3384
3385
3386
3387 --Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-
3388 92
3389
3390
3391 CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
3392 CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
3393 CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
3394
3395 CREATE TYPE UUID UNDER ShortName FINAL;
3396 CREATE TYPE URI UNDER LongName FINAL;
3397
3398 CREATE TABLE ExtrinsicObject (
3399
3400 --Object Attributes
3401 id UUID PRIMARY KEY NOT NULL,
3402 name LongName,
3403 description FreeFormText,
3404 accessControlPolicy UUID NOT NULL,
3405
3406 --Versionable attributes
3407 majorVersion INT DEFAULT 0 NOT NULL,
3408 minorVersion INT DEFAULT 1 NOT NULL,
3409
3410 --RegistryEntry attributes
3411 status INT DEFAULT 0 NOT NULL,
3412 userVersion ShortName,
3413 stability INT DEFAULT 0 NOT NULL,

```



```

3414     expirationDate                TIMESTAMP,
3415
3416 --ExtrinsicObject attributes
3417     contentURI                    URI,
3418     mimeType                       ShortName,
3419     objectType                     INT DEFAULT 0 NOT NULL,
3420     opaque                         BOOLEAN DEFAULT false NOT NULL
3421
3422 );
3423
3424 CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
3425 --Must return a collection of UUIDs for related RegistryEntry instances
3426 }
3427
3428 CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
3429 --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
3430 --Collection must be in ascending order by timestamp
3431 }
3432
3433 CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
3434 --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
3435 }
3436
3437 CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
3438 --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
3439 }
3440
3441 CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
3442 --Must return a collection of UUIDs for ClassificationNodes classifying this
3443 RegistryEntry.
3444 }
3445
3446 CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
3447 --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
3448 }
3449
3450 CREATE TABLE Package (
3451
3452 --Object Attributes
3453     id                            UUID PRIMARY KEY NOT NULL,
3454     name                          LongName,
3455     description                    FreeFormText,
3456     accessControlPolicy            UUID NOT NULL,
3457
3458 --Versionable attributes
3459     majorVersion                  INT DEFAULT 0 NOT NULL,
3460     minorVersion                  INT DEFAULT 1 NOT NULL,
3461
3462 --RegistryEntry attributes
3463     status                        INT DEFAULT 0 NOT NULL,
3464     userVersion                   ShortName,
3465     stability                      INT     DEFAULT 0 NOT NULL,
3466     expirationDate                TIMESTAMP,
3467
3468 --Package attributes
3469 );
3470
3471 CREATE PROCEDURE Package_memberbjects(packageId) {
3472 --Must return a collection of UUIDs for RegistryEntrys that are members of this Package.
3473 }
3474
3475 CREATE TABLE ExternalLink (
3476
3477 --Object 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 --ExternalLink attributes
3494     externalURI          URI NOT NULL
3495 );
3496
3497 CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
3498 --Must return a collection of UUIDs for objects in this relationship
3499 }
3500
3501 CREATE TABLE ExternalIdentifier (
3502
3503 --Object Attributes
3504     id                    UUID PRIMARY KEY NOT NULL,
3505     name                  LongName,
3506     description           FreeFormText,
3507     accessControlPolicy  UUID NOT NULL,
3508
3509 --Versionable attributes
3510     majorVersion          INT DEFAULT 0 NOT NULL,
3511     minorVersion         INT DEFAULT 1 NOT NULL,
3512
3513 --RegistryEntry attributes
3514     status                INT DEFAULT 0 NOT NULL,
3515     userVersion           ShortName,
3516     stability             INT     DEFAULT 0 NOT NULL,
3517     expirationDate       TIMESTAMP,
3518
3519 --ExternalIdentifier attributes
3520     value                 ShortName NOT NULL
3521 );
3522
3523
3524
3525 --A SlotValue row represents one value of one slot in some
3526 --RegistryEntry
3527 CREATE TABLE SlotValue (
3528
3529 --Object Attributes
3530     registryEntry        UUID   PRIMARY KEY NOT NULL,
3531
3532 --Slot attributes
3533     name                  LongName NOT NULL PRIMARY KEY NOT NULL,
3534     value                 ShortName NOT NULL
3535 );
3536
3537 CREATE TABLE Association (
3538 --Object Attributes
3539     id                    UUID PRIMARY KEY NOT NULL,
3540     name                  LongName,
3541     description           FreeFormText,
3542     accessControlPolicy  UUID NOT NULL,
3543
3544 --Versionable attributes
3545     majorVersion          INT DEFAULT 0 NOT NULL,
3546     minorVersion         INT DEFAULT 1 NOT NULL,
3547
3548 --RegistryEntry attributes
3549     status                INT DEFAULT 0 NOT NULL,

```

```

3550     userVersion          ShortName,
3551     stability             INT     DEFAULT 0 NOT NULL,
3552     expirationDate       TIMESTAMP,
3553
3554 --Association attributes
3555     associationType       INT NOT NULL,
3556     bidirectional         BOOLEAN DEFAULT false NOT NULL,
3557     sourceObject          UUID NOT NULL,
3558     sourceRole            ShortName,
3559     label                 ShortName,
3560     targetObject          UUID NOT NULL,
3561     targetRole            ShortName
3562 );
3563
3564 --Classification is currently identical to Association
3565 CREATE TABLE Classification (
3566 --Object Attributes
3567     id                     UUID PRIMARY KEY NOT NULL,
3568     name                   LongName,
3569     description            FreeFormText,
3570     accessControlPolicy    UUID NOT NULL,
3571
3572 --Versionable attributes
3573     majorVersion          INT DEFAULT 0 NOT NULL,
3574     minorVersion          INT DEFAULT 1 NOT NULL,
3575
3576 --RegistryEntry attributes
3577     status                 INT DEFAULT 0 NOT NULL,
3578     userVersion            ShortName,
3579     stability              INT     DEFAULT 0 NOT NULL,
3580     expirationDate        TIMESTAMP,
3581
3582 --Classification attributes. Assumes not derived from Association
3583     sourceObject           UUID NOT NULL,
3584     targetObject           UUID NOT NULL,
3585 );
3586
3587
3588 CREATE TABLE ClassificationNode (
3589 --Object Attributes
3590     id                     UUID PRIMARY KEY NOT NULL,
3591     name                   LongName,
3592     description            FreeFormText,
3593     accessControlPolicy    UUID NOT NULL,
3594
3595 --Versionable attributes
3596     majorVersion          INT DEFAULT 0 NOT NULL,
3597     minorVersion          INT DEFAULT 1 NOT NULL,
3598
3599 --RegistryEntry attributes
3600     status                 INT DEFAULT 0 NOT NULL,
3601     userVersion            ShortName,
3602     stability              INT     DEFAULT 0 NOT NULL,
3603     expirationDate        TIMESTAMP,
3604
3605 --ClassificationNode attributes
3606     parent                 UUID,
3607     path                   VARCHAR(512) NOT NULL,
3608     code                   ShortName
3609 );
3610
3611 CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
3612 --Must return a collection of UUIDs for RegistryEntries classified by this
3613 ClassificationNode
3614 }
3615
3616 --Begin Registry Audit Trail tables
3617

```

```

3618 CREATE TABLE AuditableEvent (
3619 --Object Attributes
3620     id                UUID PRIMARY KEY NOT NULL,
3621     name              LongName,
3622     description       FreeFormText,
3623     accessControlPolicy  UUID NOT NULL,
3624
3625 --AuditableEvent attributes
3626     user              UUID,
3627     eventType         INT DEFAULT 0 NOT NULL,
3628     registryEntry    UUID NOT NULL,
3629     timestamp         TIMESTAMP NOT NULL,
3630 );
3631
3632
3633
3634 CREATE TABLE User (
3635 --Object Attributes
3636     id                UUID PRIMARY KEY NOT NULL,
3637     name              LongName,
3638     description       FreeFormText,
3639     accessControlPolicy  UUID NOT NULL,
3640
3641 --User attributes
3642     organization      UUID NOT NULL
3643
3644 --address attributes flattened
3645     address_city      ShortName,
3646     address_country   ShortName,
3647     address_postalCode ShortName,
3648     address_state     ShortName,
3649     address_street    ShortName,
3650
3651     email             ShortName,
3652
3653 --fax attribute flattened
3654     fax_areaCode      VARCHAR(4) NOT NULL,
3655     fax_countryCode   VARCHAR(4),
3656     fax_extension     VARCHAR(8),
3657     fax_umber         VARCHAR(8) NOT NULL,
3658     fax_url           URI
3659
3660 --mobilePhone attribute flattened
3661     mobilePhone_areaCode  VARCHAR(4) NOT NULL,
3662     mobilePhone_countryCode VARCHAR(4),
3663     mobilePhone_extension VARCHAR(8),
3664     mobilePhone_umber     VARCHAR(8) NOT NULL,
3665     mobilePhone_url       URI
3666
3667 --name attribute flattened
3668     name_firstName      ShortName,
3669     name_middleName     ShortName,
3670     name_lastName       ShortName,
3671
3672 --pager attribute flattened
3673     pager_areaCode      VARCHAR(4) NOT NULL,
3674     pager_countryCode   VARCHAR(4),
3675     pager_extension     VARCHAR(8),
3676     pager_umber         VARCHAR(8) NOT NULL,
3677     pager_url           URI
3678
3679 --telephone attribute flattened
3680     telephone_areaCode  VARCHAR(4) NOT NULL,
3681     telephone_countryCode VARCHAR(4),
3682     telephone_extension VARCHAR(8),
3683     telephone_umber     VARCHAR(8) NOT NULL,
3684     telephone_url       URI,
3685

```

```

3686         url                                URI,
3687     );
3688
3689
3690 CREATE TABLE Organization (
3691 --Object Attributes
3692     id                                UUID PRIMARY KEY NOT NULL,
3693     name                                LongName,
3694     description                        FreeFormText,
3695     accessControlPolicy                UUID NOT NULL,
3696
3697 --Versionable attributes
3698     majorVersion                      INT DEFAULT 0 NOT NULL,
3699     minorVersion                      INT DEFAULT 1 NOT NULL,
3700
3701 --RegistryEntry attributes
3702     status                            INT DEFAULT 0 NOT NULL,
3703     userVersion                      ShortName,
3704     stability                          INT     DEFAULT 0 NOT NULL,
3705     expirationDate                   TIMESTAMP,
3706
3707 --Organization attributes
3708
3709 --Organization.address attribute flattened
3710     address_city                      ShortName,
3711     address_country                   ShortName,
3712     address_postalCode                ShortName,
3713     address_state                     ShortName,
3714     address_street                    ShortName,
3715
3716 --primary contact for Organization, points to a User.
3717 --Note many Users may belong to the same Organization
3718     contact                            UUID NOT NULL,
3719
3720 --Organization.fax attribute falttened
3721     fax_areaCode                      VARCHAR(4) NOT NULL,
3722     fax_countryCode                   VARCHAR(4),
3723     fax_extension                      VARCHAR(8),
3724     fax_umber                         VARCHAR(8) NOT NULL,
3725     fax_url                           URI,
3726
3727 --Organization.parent attribute
3728     parent                            UUID,
3729
3730 --Organization.telephone attribute falttened
3731     telephone_areaCode                VARCHAR(4) NOT NULL,
3732     telephone_countryCode             VARCHAR(4),
3733     telephone_extension                VARCHAR(8),
3734     telephone_umber                   VARCHAR(8) NOT NULL,
3735     telephone_url                     URI
3736 );
3737
3738
3739 --Note that the RIM security view is not visible through the public query mechanism
3740 --in the current release
3741
3742
3743 --The RegistryEntry View allows polymorphic queries over all RIM classes derived
3744 --from RegistryEntry
3745
3746 CREATE VIEW RegistryEntry (
3747 --Object Attributes
3748     id,
3749     name,
3750     description,
3751     accessControlPolicy,
3752
3753 --Versionable attributes

```

```
3754     majorVersion,
3755     minorVersion,
3756
3757 --RegistryEntry attributes
3758     status,
3759     userVersion,
3760     stability,
3761     expirationDate
3762
3763 ) AS
3764 SELECT
3765 --Object Attributes
3766     id,
3767     name,
3768     description,
3769     accessControlPolicy,
3770
3771 --Versionable attributes
3772     majorVersion,
3773     minorVersion,
3774
3775 --RegistryEntry attributes
3776     status,
3777     userVersion,
3778     stability,
3779     expirationDate
3780
3781 FROM ExtrinsicObject
3782 UNION
3783
3784 SELECT
3785 --Object Attributes
3786     id,
3787     name,
3788     description,
3789     accessControlPolicy,
3790
3791 --Versionable attributes
3792     majorVersion,
3793     minorVersion,
3794
3795 --RegistryEntry attributes
3796     status,
3797     userVersion,
3798     stability,
3799     expirationDate
3800 FROM (Registry)Package
3801 UNION
3802
3803 SELECT
3804 --Object Attributes
3805     id,
3806     name,
3807     description,
3808     accessControlPolicy,
3809
3810 --Versionable attributes
3811     majorVersion,
3812     minorVersion,
3813
3814 --RegistryEntry attributes
3815     status,
3816     userVersion,
3817     stability,
3818     expirationDate
3819 FROM ClassificationNode;
```

3820

3821 **Appendix D Security Implementation Guideline**

3822 This section provides a suggested blueprint for how security processing may be
3823 implemented in the Registry. It is meant to be illustrative not prescriptive.
3824 Registries may choose to have different implementations as long as they support
3825 the default security roles and authorization rules described in this document.

3826 **D.1 Authentication**

- 3827 1. As soon as a message is received, the first work is the authentication. A
3828 principal object is created.
- 3829 2. If the message is signed, it is verified (including the validity of the certificate)
3830 and the DN of the certificate becomes the identity of the principal. Then the
3831 Registry is searched for the principal and if found, the roles and groups are
3832 filled in.
- 3833 3. If the message is not signed, an empty principal is created with the role
3834 RegistryGuest. This step is for symmetry and to decouple the rest of the
3835 processing.
- 3836 4. Then the message is processed for the command and the objects it will act on

3837 **D.2 Authorization**

3838 For every object, the access controller will iterate through all the
3839 AccessControlPolicy objects with the object and see if there is a chain through
3840 the permission objects to verify that the requested method is permitted for the
3841 Principal. If any of the permission objects which the object is associated with has
3842 a common role, or identity, or group with the principal, the action is permitted.

3843 **D.3 Registry Bootstrap**

3844 When a Registry is newly created, a default Principal object should be created
3845 with the identity of the Registry Admin's certificate DN with a role RegistryAdmin.
3846 This way, any message signed by the Registry Admin will get all the privileges.

3847 When a Registry is newly created, a singleton instance of AccessControlPolicy is
3848 created as the default AccessControlPolicy. This includes the creation of the
3849 necessary Permission instances as well as the Privileges and Privilege attributes.

3850 **D.4 Content Submission – Client Responsibility**

3851 The Registry client has to sign the contents before submission – otherwise the
3852 content will be rejected.

3853 **D.5 Content Submission – Registry Responsibility**

- 3854 1. Like any other request, the client will be first authenticated. In this case, the
3855 Principal object will get the DN from the certificate.
- 3856 2. As per the request in the message, the RegistryEntry will be created.
- 3857 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 3858 4. If a principal with the identity of the SO is not available, an identity object with
3859 the SO's DN is created
- 3860 5. A principal with this identity is created

3861 **D.6 Content Delete/Deprecate – Client Responsibility**

3862 The Registry client has to sign the payload (not entire message) before
3863 submission, for authentication purposes; otherwise, the request will be
3864 rejected

3865 **D.7 Content Delete/Deprecate – Registry Responsibility**

- 3866 1. Like any other request, the client will be first authenticated. In this case, the
3867 Principal object will get the DN from the certificate. As there will be a principal
3868 with this identity in the Registry, the Principal object will get all the roles from
3869 that object
- 3870 2. As per the request in the message (delete or deprecate), the appropriate
3871 method in the Object will be accessed.
- 3872 3. The access controller performs the authorization by iterating through the
3873 Permission objects associated with this object via the singleton default
3874 AccessControlPolicy.
- 3875 4. If authorization succeeds then the action will be permitted. Otherwise an error
3876 response is sent back with a suitable AuthorizationException error message.

3877 **Appendix E Native Language Support (NLS)**

3878 **E.1 Definitions**

3879 Although this section discusses only character set and language, the following
3880 terms have to be defined clearly.

3881

3882 **E.1.1 Coded Character Set (CCS):**

3883 CCS is a mapping from a set of abstract characters to a set of integers. [RFC
3884 2130]. Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.
3885

3886 **E.1.2 Character Encoding Scheme (CES):**

3887 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130].
3888 Examples of CES are ISO-2022, UTF-8.

3889 **E.1.3 Character Set (charset):**

3890 charset is a set of rules for mapping from a sequence of octets to a sequence of
3891 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP,
3892 EUC-KR.
3893

3894 A list of registered character sets can be found at [IANA].

3895 **E.2 NLS And Request / Response Messages**

3896 For the accurate processing of data in both registry client and registry services, it
3897 is essential to know which character set is used. Although the body part of the
3898 transaction may contain the charset in xml encoding declaration, registry client
3899 and registry services shall specify charset parameter in MIME header when they
3900 use text/xml. Because as defined in [RFC 3023], if a text/xml entity is received
3901 with the charset parameter omitted, MIME processors and XML processors
3902 MUST use the default charset value of "us-ascii".

3903

3904 Ex. Content-Type: text/xml; charset=ISO-2022-JP

3905

3906 Also, when an application/xml entity is used, the charset parameter is optional,
3907 and registry client and registry services must follow the requirements in section
3908 4.3.3 of [REC-XML] which directly address this contingency.
3909

3909

3910 If another Content-Type is chosen to be used, usage of charset must follow [RFC
3911 3023].

3912 **E.3 NLS And Storing of RegistryEntry**

3913 This section provides NLS guidelines on how a registry should store
3914 **RegistryEntry** instances.

3915 **E.3.1 Character Set of *RegistryEntry***

3916 This is basically an implementation issue because the actual character set that
3917 the ***RegistryEntry*** is stored with, does not affect the interface. However, it is
3918 highly recommended to use UTF-16 or UTF-8 for covering various languages.

3919 **E.3.2 Language Information of *RegistryEntry***

3920 The language may be specified in `xml:lang` attribute (section 2.12 [REC-XML]). If
3921 the `xml:lang` attribute is specified, then the registry may use that language code
3922 as the value of a special Slot with name ***language*** and `sloType` of ***nls*** in the
3923 ***RegistryEntry***. The value must be compliant to [RFC 1766]. Slots are defined in
3924 [RIM].

3925 **E.4 NLS And Storing of Repository Items**

3926 This section provides NLS guidelines on how a registry should store repository
3927 items.

3928 **E.4.1 Character Set of Repository Items**

3929 Unlike the character set of ***RegistryEntry***, the charset of a repository item must
3930 be preserved as it is originally specified in the transaction. The registry may use
3931 a special Slot with name ***repositoryItemCharset***, and `sloType` of ***nls*** for the
3932 ***RegistryEntry*** for storing the charset of the corresponding repository item. Value
3933 must be the one defined in [RFC 2277], [RFC 2278]. The
3934 ***repositoryItemCharset*** is optional because not all repository items require it.

3935

3936 **E.4.2 Language information of repository item**

3937 Specifying only character set is not enough to tell which language is used in
3938 the repository item. A registry may use a special Slot with name
3939 ***repositoryItemLang***, and `sloType` of ***nls*** to store that information. This
3940 attribute is optional because not all repository items require it. Value must be
3941 compliant to [RFC 1766]

3942

3943 This document currently specifies only the method of sending the information of
3944 character set and language, and how it is stored in a registry. However, the
3945 language information may be used as one of the query criteria, such as retrieving
3946 only DTD written in French. Furthermore, a language negotiation procedure, like
3947 registry client is asking a favorite language for messages from registry services,
3948 could be another functionality for the future revision of this document.

3949 **Appendix F Terminology Mapping**

3950 While every attempt has been made to use the same terminology used in other
 3951 works there are some terminology differences.

3952 The following table shows the terminology mapping between this specification
 3953 and that used in other specifications and working groups.

This Document	OASIS	ISO 11179
“repository item”	Registered Object	
RegistryEntry	Registry Item	Administered Component
ExternalObject	Related Data	N/A
Object.ID	RaltemId	
ExtrinsicObject.uri	ObjectLocation	
ExtrinsicObject.objectType	DefnSource, PrimaryClass, SubClass	
RegistryEntry.name	CommonName	
Object.description	Description	
ExtrinsicObject.mimeType	MimeType	
Versionable.majorVersion	partially to Version	
Versionable.minorVersion	partially to Version	
RegistryEntry.status	RegStatus	

3954 **Table 1: Terminology Mapping Table**

3955 **10 References**

3956 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm

3957 [TA] ebXML Technical Architecture

3958 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf

3959 [OAS] OASIS Information Model

3960 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>

3961 [ISO] ISO 11179 Information Model

3962 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
 3963

- 3964 [BDM] Registry and Repository: Business Domain Model
3965 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 3966 [RIM] ebXML Registry Information Model
3967 http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
- 3968 [BPM] ebXML Business Process Metamodel Specification Schema
3969 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 3970 [CPA] Trading-Partner Specification
3971 http://www.ebxml.org/project_teams/trade_partner/private/
- 3972 [CTB] Context table informal document from Core Components
- 3973 [MS] ebXML Messaging Service Specification, Version 0.21
3974 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
- 3975 [ERR] ebXML TRP Error Handling Specification
3976 http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf
- 3977 [SEC] ebXML Security Specification
3978 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 3979 [XPT] XML Path Language (XPath) Version 1.0
3980 <http://www.w3.org/TR/xpath>
- 3981 [SQL] Structured Query Language (FIPS PUB 127-2)
3982 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 3983 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
3984 (SQL/PSM) [ISO/IEC 9075-4:1996]
- 3985 [IANA] IANA (Internet Assigned Numbers Authority).
3986 Official Names for Character Sets, ed. Keld Simonsen et al.
3987 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
3988
- 3989 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
3990 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
3991 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
3992
- 3993 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
3994 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
3995 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
3996
- 3997 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
3998 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
3999 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
4000
- 4001 [RFC 2130] IETF (Internet Engineering Task Force). RFC 2130:

- 4002 The Report of the IAB Character Set Workshop held 29 February - 1
4003 March, 1996,
4004 C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M.
4005 Crispin, P. Svanberg. 1997.
4006 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2130.html>
4007
4008 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
4009 XML Media Types, ed. M. Murata. 2001.
4010 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
4011
4012 [REC-XML] W3C Recommendation. Extensible Markup
4013 language(XML)1.0(Second Edition)
4014 <http://www.w3.org/TR/REC-xml>
4015

4016 **11 Disclaimer**

- 4017 The views and specification expressed in this document are those of the authors
4018 and are not necessarily those of their employers. The authors and their
4019 employers specifically disclaim responsibility for any problems arising from
4020 correct or incorrect implementation or use of this design.
4021

4021 **12 Contact Information**

4022 Team Leader

4023 Name: Scott Nieman
4024 Company: Norstan Consulting
4025 Street: 5101 Shady Oak Road
4026 City, State, Postal Code: Minnetonka, MN 55343
4027 Country: USA
4028 Phone: 952.352.5889
4029 Email: Scott.Nieman@Norstan

4030

4031 Vice Team Lead

4032 Name: Yutaka Yoshida
4033 Company: Sun Microsystems
4034 Street: 901 San Antonio Road, MS UMPK17-102
4035 City, State, Postal Code: Palo Alto, CA 94303
4036 Country: USA
4037 Phone: 650.786.5488
4038 Email: Yutaka.Yoshida@eng.sun.com

4039

4040 Editor

4041 Name: Farrukh S. Najmi
4042 Company: Sun Microsystems
4043 Street: 1 Network Dr., MS BUR02-302
4044 City, State, Postal Code: Burlington, MA, 01803-0902
4045 Country: USA
4046 Phone: 781.442.0703
4047 Email: najmi@east.sun.com

4048

4049

4049 Copyright Statement

4050 Copyright © ebXML 2000. All Rights Reserved.

4051

4052 This document and translations of it may be copied and furnished to others, and
4053 derivative works that comment on or otherwise explain it or assist in its
4054 implementation may be prepared, copied, published and distributed, in whole or
4055 in part, without restriction of any kind, provided that the above copyright notice
4056 and this paragraph are included on all such copies and derivative works.
4057 However, this document itself may not be modified in any way, such as by
4058 removing the copyright notice or references to the Internet Society or other
4059 Internet organizations, except as needed for the purpose of developing Internet
4060 standards in which case the procedures for copyrights defined in the Internet
4061 Standards process must be followed, or as required to translate it into languages
4062 other than English.

4063

4064 The limited permissions granted above are perpetual and will not be revoked by
4065 ebXML or its successors or assigns.

4066

4067 This document and the information contained herein is provided on an
4068 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR
4069 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE
4070 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
4071 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
4072 PARTICULAR PURPOSE.