



Creating A Single Global Electronic Market

1

## 2 **ebXML Registry Services**

### 3 **ebXML Registry Project Team**

4 **Working Draft 1/16/2001**

5 **This version: Version 0.83**

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/RegistryServicesSpecificationv0.83.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf)

19

### 20 ***Latest version:***

21 [http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpecificationv0.83.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf)

22

### 23 ***Previous version:***

24 [http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpecificationv0.82.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.82.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 Joseph Baran - Extol  
33 Lisa Carnahan – NIST  
34 Joe Dalman - Tie  
35 Philippe DeSmedt - Viquity  
36 Sally Fuger - AIAG  
37 Steve Hanna - Sun Microsystems  
38 Scott Hinkelman - IBM  
39 Michael Kass, NIST  
40 Jong.L Kim – Innodigital  
41 Bob Miller - GXS  
42 Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan  
43 Dale Moberg – Sterling Commerce  
44 Ron Monzillo – Sun Microsystems  
45 JP Morgenthal – XML Solutions  
46 Joel Munter - Intel  
47 Farrukh Najmi - Sun Microsystems  
48 Scott Nieman - Norstan Consulting  
49 Frank Olken – Lawrence Berkeley National Laboratory  
50 Michael Park - eSum Technologies  
51 Bruce Peat - eProcess Solutions  
52 Mike Rowley – Excelon Corporation  
53 Waqar Sadiq - Vitria  
54 Krishna Sankar - CISCO  
55 Kim Tae Soo - Government of Korea  
56 Nikola Stojanovic - Columbine JDS Systems  
57 David Webber - XML Global  
58 Yutaka Yoshida - Sun Microsystems  
59 Prasad Yendluri - webmethods  
60 Peter Z. Zhoo - Knowledge For the new Millennium  
61

61 **Table of Contents**

62 **1 Status of this Document ..... 1**

63 **2 ebXML participants ..... 2**

64 **Table of Contents..... 3**

65 **Table of Tables ..... 6**

66 **3 Introduction ..... 7**

67 3.1 Summary of Contents of Document .....7

68 3.2 General Conventions .....7

69 3.3 Audience.....7

70 3.4 Related Documents .....7

71 **4 Design Objectives..... 8**

72 4.1 Goals .....8

73 4.2 Caveats and Assumptions .....8

74 **5 System Overview ..... 8**

75 5.1 What The ebXML Registry Does .....8

76 5.2 How The ebXML Registry Works .....9

77 5.3 Schema Documents Are Submitted.....9

78 5.4 Business Process Documents Are Submitted.....9

79 5.5 Seller’s Collaboration Protocol Profile Is Submitted .....9

80 5.6 Buyer Discovers The Seller .....9

81 5.7 CPA Is Established ..... 10

82 5.8 Where the Registry Services May Be Implemented..... 10

83 **6 Registry Architecture..... 10**

84 6.1 Implicit CPA Between Clients And Registry..... 10

85 6.2 Client To Registry Communication Bootstrapping ..... 11

86 6.3 Interfaces Exposed By The Registry..... 12

87 6.3.1 Interface *RegistryService*..... 12

88 6.3.2 Interface *ObjectManager* ..... 13

89 6.3.3 Interface *ObjectQueryManager* ..... 13

90 6.4 Interfaces Exposed By Registry Clients ..... 15

91 6.4.1 Interface *RegistryClient*..... 15

92 6.4.2 Interface *ObjectManagerClient*..... 15

93 6.4.3 Interface *ObjectQueryManagerClient*..... 16

94 **7 Object Management Service ..... 17**

95 7.1 Life Cycle of a Managed Object..... 17

96 7.2 Object Attributes ..... 18

97 7.3 The Submit Objects Protocol ..... 19

98 7.4 The Approve Objects Request ..... 19

99	7.5	The Deprecate Objects Request .....	20
100	7.6	The Remove Objects Request .....	21
101	<b>8</b>	<b>Object Query Management Service .....</b>	<b>21</b>
102	8.1	Browse and Drill Down Query Support .....	22
103	8.1.1	Get Root Classification Nodes Request .....	22
104	8.1.2	Get Classification Tree Request .....	23
105	8.1.3	Get Classified Objects Request .....	24
106	8.1.3.1	Get Classified Objects Request Example .....	24
107	8.2	Ad Hoc Query Support .....	25
108	8.2.1	Query Language Syntax .....	25
109	8.2.2	Query Syntax Binding To [RIM] .....	25
110	8.2.2.1	Interface and Class Binding .....	25
111	8.2.2.2	Accessor Method To Attribute Binding .....	25
112	8.2.2.3	Primitive Attributes Binding .....	25
113	8.2.2.4	Reference Attribute Binding .....	26
114	8.2.2.5	Collection Attribute Binding .....	26
115	8.2.3	Simple Metadata Based Queries .....	26
116	8.2.4	Classification Queries .....	26
117	8.2.4.1	Identifying ClassificationNodes .....	27
118	8.2.4.2	Getting Root Classification Nodes .....	27
119	8.2.4.3	Getting Children of Specified ClassificationNode .....	27
120	8.2.4.4	Getting Objects Classified By a ClassificationNode .....	27
121	8.2.4.5	Getting ClassificationNodes That Classify an Object .....	27
122	8.2.5	Association Queries .....	28
123	8.2.5.1	Getting All Association With Specified Object As Its	
124		Source	28
125	8.2.5.2	Getting All Association With Specified Object As Its	
126		Target	28
127	8.2.5.3	Getting Associated Objects Based On Association	
128		Attributes .....	28
129	8.2.5.4	Complex Association Queries .....	29
130	8.2.6	Package Queries .....	29
131	8.2.6.1	Complex Package Queries .....	29
132	8.2.7	ExternalLink Queries .....	29
133	8.2.7.1	Complex ExternalLink Queries .....	30
134	8.2.8	Audit Trail Queries .....	30
135	8.2.9	Content Based Ad Hoc Queries .....	30
136	8.2.9.1	Automatic Classification of XML Content .....	30
137	8.2.9.2	Index Definition .....	31
138	8.2.9.3	Example Of Index Definition .....	31
139	8.2.9.4	Example of Automatic Classification .....	31
140	8.2.10	Ad Hoc Query Request/Response .....	32
141	8.3	Content Retrieval .....	33
142	8.3.1	Identification Of Content Payloads .....	33

143 8.3.2 GetContentResponse Message Structure ..... 33

144 8.4 Query And Retrieval: Typical Sequence ..... 34

145 **9 Registry Security ..... 35**

146 9.1 Integrity of Registry Content ..... 36

147 9.1.1 Message Payload Signature ..... 36

148 9.2 Authentication ..... 36

149 9.2.1 Message Header Signature ..... 36

150 9.3 Confidentiality ..... 37

151 9.3.1 On-the-wire Message Confidentiality ..... 37

152 9.3.2 Confidentiality of Registry Content ..... 37

153 9.4 Authorization ..... 37

154 9.4.1 Pre-defined Roles For Registry Users ..... 37

155 9.4.2 Default Access Control Policies ..... 37

156 **Appendix A Schemas and DTD Definitions ..... 38**

157 A.1 ebXMLError Message DTD ..... 38

158 A.2 ebXML Registry DTD ..... 39

159 **Appendix B Interpretation of UML Diagrams ..... 46**

160 B.1 UML Class Diagram ..... 46

161 B.2 UML Sequence Diagram ..... 47

162 **Appendix C BNF for Query Syntax Grammar ..... 47**

163 **Appendix D Security Implementation Guideline ..... 49**

164 D.1 Authentication ..... 49

165 D.2 Authorization ..... 49

166 D.3 Registry Bootstrap ..... 50

167 D.4 Content Submission – Client Responsibility ..... 50

168 D.5 Content Submission – Registry Responsibility ..... 50

169 D.6 Content Delete/Deprecate – Client Responsibility ..... 50

170 D.7 Content Delete/Deprecate – Registry Responsibility ..... 50

171 **Appendix E Terminology Mapping ..... 51**

172 **10 References ..... 51**

173 **11 Disclaimer ..... 52**

174 **12 Contact Information ..... 53**

175 **Copyright Statement ..... 54**

176 **Table of Figures**

177 Figure 1: ebXML Registry Interfaces ..... 12

178 Figure 3: Life Cycle of a Managed Object ..... 18

179 Figure 4: Submit Objects Sequence Diagram ..... 19

180 Figure 5: Approve Objects Sequence Diagram..... 20

181 Figure 6: Deprecate Objects Sequence Diagram ..... 20

182 Figure 7: Remove Objects Sequence Diagram..... 21

183 Figure 8: Get Root Classification Nodes Sequence Diagram..... 22

184 Figure 9: Get Root Classification Nodes Asynchronous Sequence Diagram ..... 23

185 Figure 10: Get Classification Tree Sequence Diagram ..... 23

186 Figure 11: Get Classification Tree Asynchronous Sequence Diagram ..... 23

187 Figure 12: A Sample Geography Classification ..... 24

188 Figure 13: Submit Ad Hoc Query Sequence Diagram ..... 32

189 Figure 14: Submit Ad Hoc Query Asynchronous Sequence Diagram..... 32

190 Figure 15: Typical Query and Retrieval Sequence..... 35

191 **Table of Tables**

192 Table 1: Terminology Mapping Table ..... 51

193

194

## 194 **3 Introduction**

### 195 **3.1 Summary of Contents of Document**

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

198 A separate document, *ebXML Registry Information Model* [RIM], provides  
199 information on the type of metadata that is stored in the Registry as well as the  
200 relationships among metadata classes.

### 201 **3.2 General Conventions**

202 o UML diagrams are used as a way to concisely describe concepts. They are  
203 not intended to convey any specific implementation or methodology  
204 requirements.

205 o The term "*managed object content*" is used to refer to actual Registry content  
206 (e.g. a DTD, as opposed to metadata about the DTD).

207 o The term "*ManagedObject*" is used to refer to an object that provides  
208 metadata about a content instance (*managed object content*).

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

### 212 **3.3 Audience**

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

215 o Implementers of ebXML Registry Services

216 o Implementers of ebXML Registry Clients

### 217 **3.4 Related Documents**

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

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

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

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

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

- 226 e) *Collaboration Protocol Specification* [CPA] (under development) - defines  
227 how profiles can be defined for a party and how two parties' profiles may  
228 be used to define a party agreement

229

## 230 **4 Design Objectives**

### 231 **4.1 Goals**

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

- 233 o Communicate functionality of Registry services to software developers
- 234 o Specify the interface for Registry clients and the Registry
- 235 o Provide a basis for future support of more complete ebXML Registry  
236 requirements
- 237 o Be compatible with other ebXML specifications

### 238 **4.2 Caveats and Assumptions**

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

242 It is assumed that:

- 243 1. All interactions between the clients of the ebXML Registry and the ebXML  
244 Registry will be conducted using ebXML Messaging Service.
- 245 2. All access to the Registry content is exposed via the interfaces defined for  
246 the Registry Services.
- 247 3. The Registry makes use of a Repository for storing and retrieving  
248 persistent information required by the Registry Services. This is an  
249 implementation detail that will not be discussed further in this specification.

## 250 **5 System Overview**

### 251 **5.1 What The ebXML Registry Does**

252 The ebXML Registry provides a set of services that enable sharing of information  
253 between interested parties for the purpose of enabling business process  
254 integration between such parties based on the ebXML specifications. The shared  
255 information is maintained as objects in a repository and managed by the ebXML  
256 Registry Services defined in this document and its future versions.

## 257 **5.2 How The ebXML Registry Works**

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

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

## 269 **5.3 Schema Documents Are Submitted**

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

## 274 **5.4 Business Process Documents Are Submitted**

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

## 279 **5.5 Seller's Collaboration Protocol Profile Is Submitted**

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

## 285 **5.6 Buyer Discovers The Seller**

286 The buyer browses the Registry using a Registry Browser GUI tool. The buyer  
287 searches the Registry for suitable sellers according to the flexible classification  
288 schemes supported by the Registry. For example the buyer may search for all  
289 parties that are in the Automotive Industry, play a seller role, support the  
290 RosettaNet PIP3A4 process and sell Car Stereos.

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

## 293 **5.7 CPA Is Established**

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

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

## 300 **5.8 Where the Registry Services May Be Implemented**

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

## 304 **6 Registry Architecture**

305 The ebXML Registry architecture consists of an ebXML Registry and ebXML  
306 Registry clients. Clients communicate with the Registry using the ebXML  
307 Messaging Service in the same manner as any two ebXML applications  
308 communicating with each other. Future versions of this specification may extend  
309 the Registry architecture to support distributed Registries.

310 This specification defines the interaction between a Registry client and the  
311 Registry as a set of business processes. Although these interaction protocols  
312 and business processes are specific to the Registry, they are identical in nature  
313 to the interactions between two parties conducting B2B message communication  
314 using the ebXML Messaging Service as defined by [MS] and [CPA].

315 As such, these Registry specific interaction protocols and business processes  
316 are a special case of business process interactions between two parties using  
317 the ebXML Messaging Service.

### 318 **6.1 Implicit CPA Between Clients And Registry**

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

321 Similarly, this specification defines a CPA between a Registry client and the  
322 Registry. Typical B2B interactions in ebXML require an explicit CPA to be  
323 negotiated between parties. However, the CPA between clients and the Registry  
324 is an implicit CPA that describes the interfaces that the Registry and the client  
325 expose to each other for Registry specific interactions. These interfaces are  
326 described in Figure 1 and subsequent sections.

## 327 **6.2 Client To Registry Communication Bootstrapping**

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

333 For example, if the communication used by the Registry is HTTP then the  
334 communication address is a URL. In this example, the client uses the Registry's  
335 public URL to create an implicit CPA with the Registry. When the client sends a  
336 request to the Registry, it provides a URL to itself. The Registry uses the client's  
337 URL to form its version of an Implicit CPA with the client. At this point a session  
338 is established within the Registry.

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

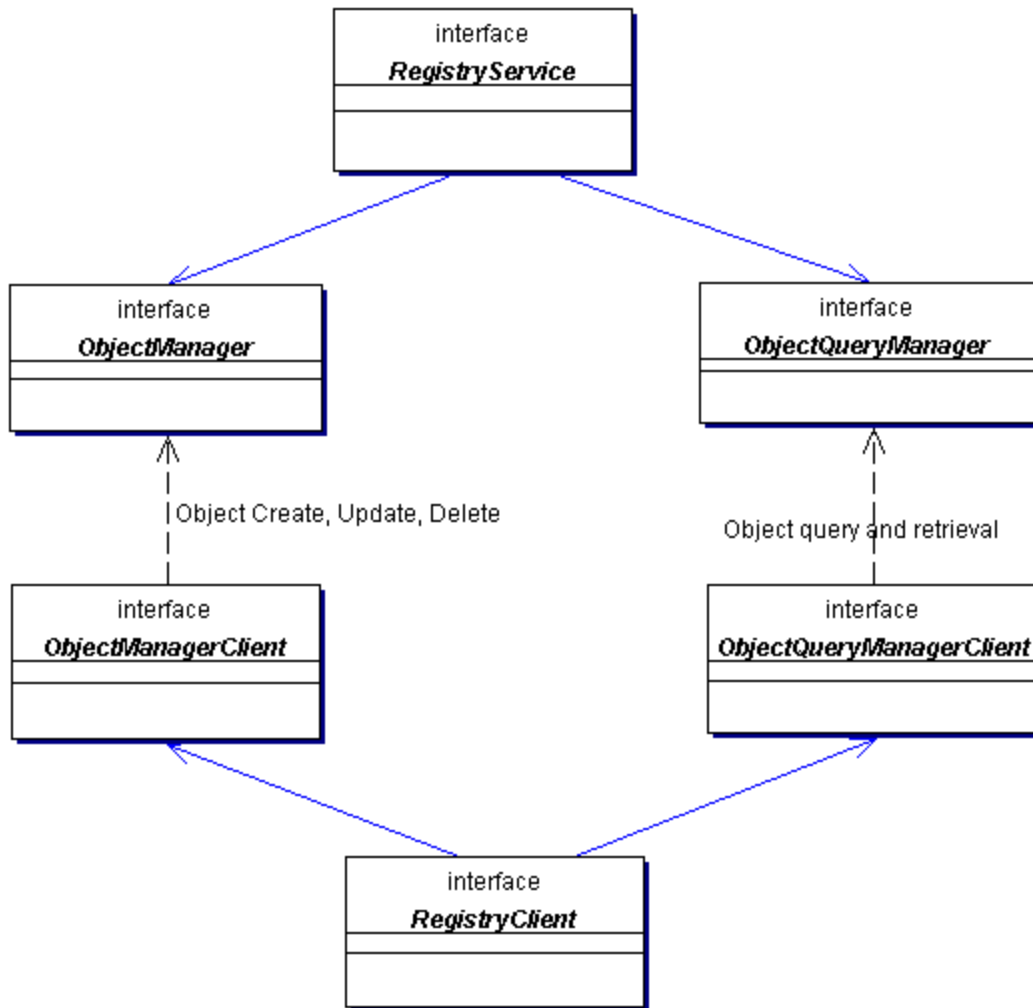


Figure 1: ebXML Registry Interfaces

342  
343

### 6.3 Interfaces Exposed By The Registry

344

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

#### 6.3.1 Interface *RegistryService*

347

348

---

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

352

---

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

353

354 **6.3.2 Interface *ObjectManager***

355

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

360

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

361 **6.3.3 Interface *ObjectQueryManager***

362

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

367

<b>Method Summary</b>	
<a href="#">GetClassificationTreeResponse</a>	<p><a href="#">getClassificationTree</a> (  <a href="#">GetClassificationTreeRequest</a> req)                      Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.</p>
void	<p><a href="#">getClassificationTreeAsync</a> (  <a href="#">GetClassificationTreeRequest</a> req)                      Asynchronous version of getClassificationTree.</p>
<a href="#">GetClassifiedObjectsResponse</a>	<p><a href="#">getClassifiedObjects</a> (  <a href="#">GetClassifiedObjectsRequest</a> req)                      Returns a collection of references to ManagedObjects classified under specified ClassificationItem.</p>
void	<p><a href="#">getClassifiedObjectsAsync</a> (  <a href="#">GetClassifiedObjectsRequest</a> req)                      Asynchronous version of getClassifiedObjects.</p>
<a href="#">GetContentResponse</a>	<p><a href="#">getContent</a> (                      Returns the specified content. The response includes all the content specified in the request as additional payloads within the response message.</p>
void	<p><a href="#">getContentAsync</a> (                      Async version of getContent.</p>
<a href="#">GetRootClassificationNodesResponse</a>	<p><a href="#">getRootClassificationNodes</a> (  <a href="#">GetRootClassificationNodesRequest</a> req)                      Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.</p>
void	<p><a href="#">getRootClassificationNodesAsync</a> (  <a href="#">GetRootClassificationNodesRequest</a> req)                      Async version of getRootClassificationNodes.</p>
<a href="#">AdhocQueryResponse</a>	<p><a href="#">submitAdhocQuery</a> (<a href="#">AdhocQueryRequest</a> req)                      Submit an ad hoc query request.</p>
void	<p><a href="#">submitAdhocQueryAsync</a> (<a href="#">AdhocQueryRequest</a> req)                      Async version of submitAdhocQuery.</p>

368 **6.4 Interfaces Exposed By Registry Clients**

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

370 **6.4.1 Interface *RegistryClient***

371 \_\_\_\_\_  
 372 This is the principal interface implemented by a Registry client. The client  
 373 provides this interface when creating a connection to the Registry. It provides the  
 374 methods that are used by the Registry to discover service specific interfaces  
 375 implemented by the client.

376

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

377

378 **6.4.2 Interface *ObjectManagerClient***

379 \_\_\_\_\_  
 380 This is the client callback interface for the ObjectManager service of the Registry.  
 381 The ObjectManager invokes its methods to notify the client about the results of a  
 382 previously submitted request from the client to the ObjectManager service.

383

<b>Method Summary</b>	
void	<a href="#"><u>approveObjectsAccepted(RequestAcceptedResponse resp)</u></a> Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.
void	<a href="#"><u>approveObjectsError(ebXMLError error)</u></a> Notifies client that a previously submitted ApproveObjectsRequest was not accepted by the Registry due to an error.

void	<a href="#">deprecateObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted DeprecateObjectsRequest was accepted by the Registry.
void	<a href="#">deprecateObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted DeprecateObjectsRequest was not accepted by the Registry due to an error.
void	<a href="#">removeObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted RemoveObjectsRequest was accepted by the Registry.
void	<a href="#">removeObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted RemoveObjectsRequest was not accepted by the Registry due to an error.
void	<a href="#">submitObjectsAccepted</a> ( <a href="#">RequestAcceptedResponse</a> resp) Notifies client that a previously submitted SubmitObjectsRequest was accepted by the Registry.
void	<a href="#">submitObjectsError</a> ( <a href="#">ebXMLError</a> error) Notifies client that a previously submitted SubmitObjectsRequest was not accepted by the Registry due to an error.

384

385 **6.4.3 Interface *ObjectQueryManagerClient***

386

387 This is the callback interface for the ObjectQueryManager service of the Registry.  
 388 The ObjectQueryManager invokes its methods to notify the client about the  
 389 results of a previously submitted query request from client to the  
 390 ObjectQueryManager service.

391

<b>Method Summary</b>	
void	<a href="#">getClassificationTreeAsyncResponse</a> ( <a href="#">GetClassificationTreeResponse</a> resp) Async response for getClassificationTreeAsync request.
void	<a href="#">getClassifiedObjectsAsyncResponse</a> ( <a href="#">GetClassifiedObjectsResponse</a> resp) Async response for getClassifiedObjectsAsync request.

void	<a href="#">getContentAsyncResponse</a> ( <a href="#">GetContentResponse</a> resp) Async response for getContent request.
void	<a href="#">getRootClassificationNodesAsyncResponse</a> ( <a href="#">GetRootClassificationNodesResponse</a> resp) Async response for getRootClassificationNodesAsync request.
void	<a href="#">submitAdhocQueryAsyncResponse</a> ( <a href="#">AdhocQueryResponse</a> resp) Async response for submitAdhocQueryAsync request.

## 392 **7 Object Management Service**

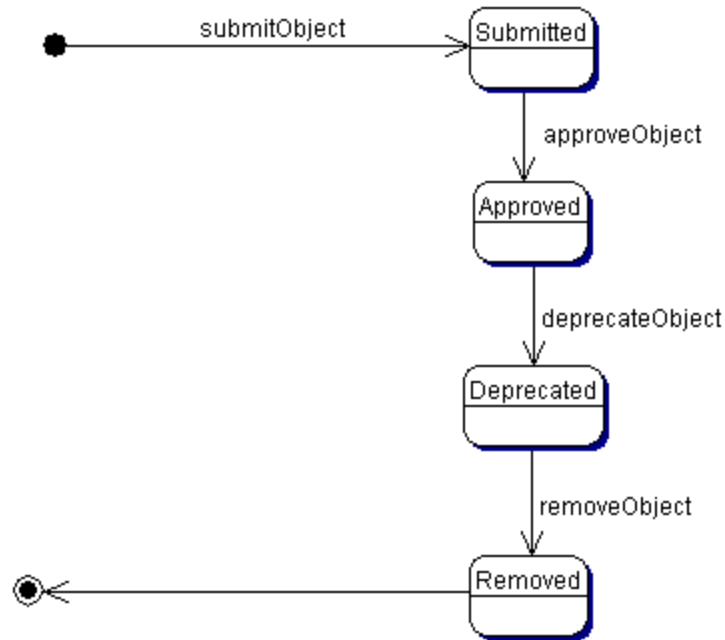
393 This section defines the Object Management service of the Registry. The Object  
394 Management Service is a sub-service of the Registry service. It provides the  
395 functionality required by RegistryClients to manage the life cycle of managed  
396 object contents (e.g. XML documents required for ebXML business processes).  
397 The Object Management Service can be used with all types of managed object  
398 contents as well as the metadata objects specified in [RIM] such as Classification  
399 and Association.

400 In the current version of this specification, any client may submit content as long  
401 as the content is digitally signed by an approved Certification Authority.  
402 Submitting Organizations do not have to register prior to submitting content.

### 403 **7.1 Life Cycle of a Managed Object**

404 The main purpose of the Object Management service is to manage the life cycle  
405 of managed object contents in the Registry.

406 Figure 2 shows the typical life cycle of a managed object content. Note that the  
407 current version of this specification does not support Object versioning. Object  
408 versioning will be added in a future version of this specification.



409  
410

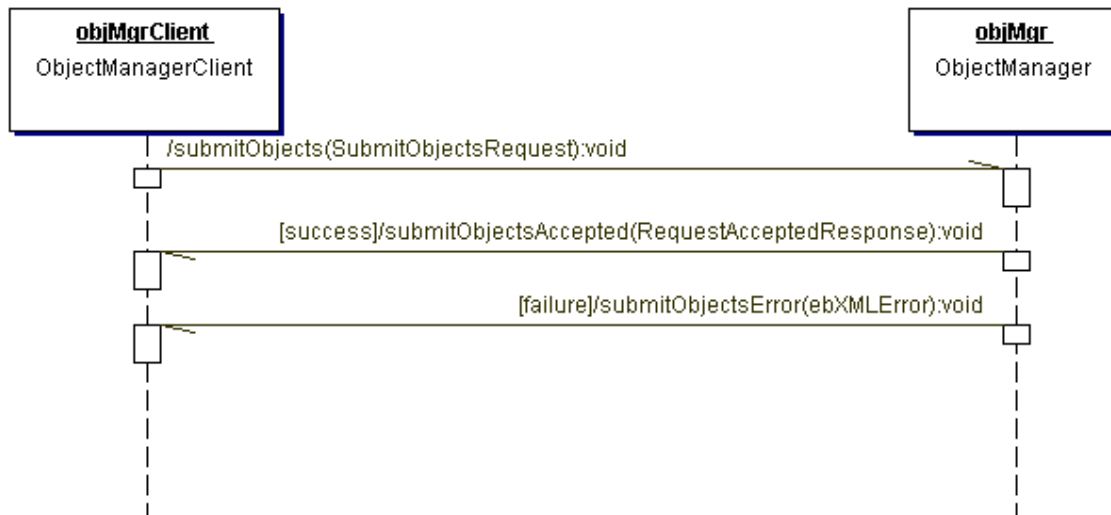
**Figure 3: Life Cycle of a Managed Object**

411 **7.2 Object Attributes**

412 A managed object content is associated with a set of standard metadata defined  
 413 as attributes of the Object class and its sub-classes as described in [RIM]. These  
 414 attributes reside outside of the actual managed object content and catalog  
 415 descriptive information about the managed object content. XML DTD elements  
 416 called ExtrinsicObject and IntrinsicObject (See Appendix A.1 for details.) are  
 417 defined that encapsulates all object metadata attributes defined in [RIM] as  
 418 attributes of the DTD elements.

419 **7.3 The Submit Objects Protocol**

420 This section describes the protocol of the Registry Service that allows a  
 421 RegistryClient to submit one or more managed object contents in the repository  
 422 using the *ObjectManager* on behalf of a Submitting Organization. It is expressed  
 423 in UML notation as described in Appendix B.



424

425

**Figure 4: Submit Objects Sequence Diagram**

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

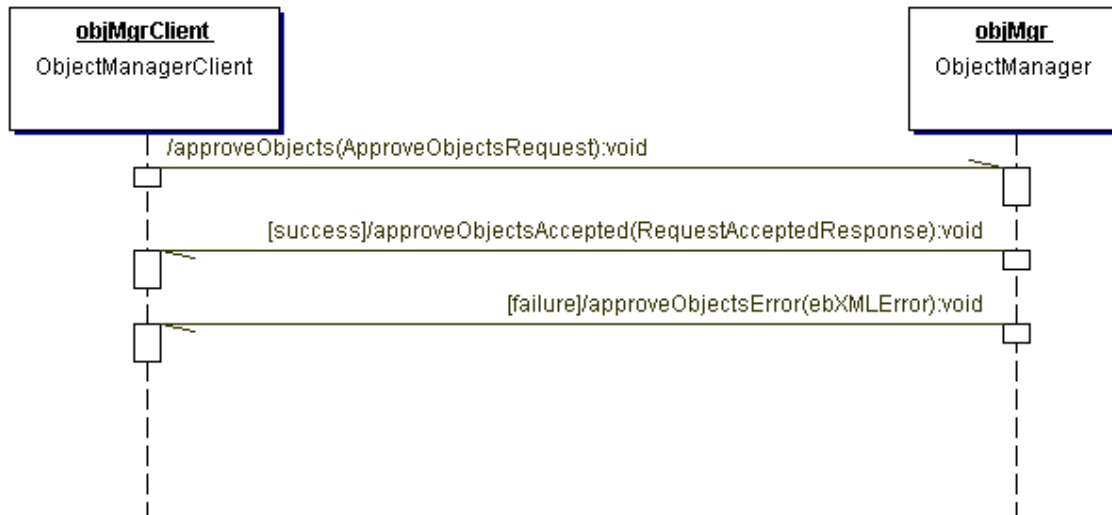
428 The SubmitObjectRequest message includes 1 or more SubmittedObject  
 429 elements.

430 Each SubmittedObject element specifies an ExtrinsicObject along with any  
 431 Classifications, Associations, ExternalLinks, or Packages related to the object  
 432 being submitted.

433 An ExtrinsicObject element provides required metadata about the content being  
 434 submitted to the Registry as defined by [RIM]. Note that these standard  
 435 ExtrinsicObject attributes are separate from the managed object content itself,  
 436 thus allowing the ebXML Registry to catalog arbitrary objects. In addition each  
 437 SubmittedObject in the request may optionally specify any number of  
 438 Classifications, Associations and ExternalLinks for the SubmittedObject.

439 **7.4 The Approve Objects Request**

440 This section describes the protocol of the Registry Service that allows a client to  
 441 approve one or more previously submitted managed object contents using the  
 442 Object Manager. Once a managed object content is approved it will become  
 443 available for use by business parties (e.g. during the assembly of new CPAs and  
 444 Collaboration Protocol Profiles).



445

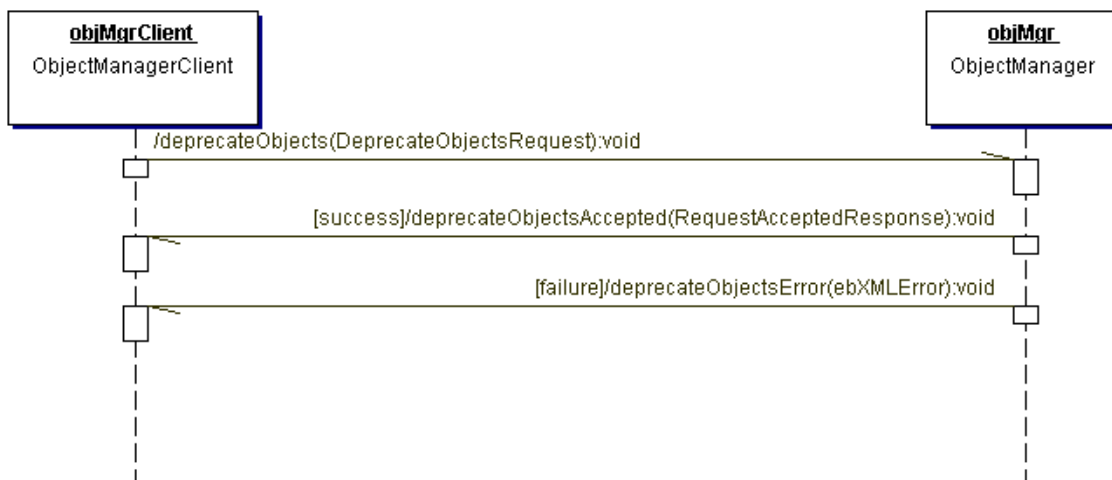
446

**Figure 5: Approve Objects Sequence Diagram**

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

449 **7.5 The Deprecate Objects Request**

450 This section describes the protocol of the Registry Service that allows a client to  
 451 deprecate one or more previously submitted managed object contents using the  
 452 Object Manager. Once an object is deprecated, no new references (e.g. *new*  
 453 Associations, Classifications and ExternalLinks) to that object can be submitted.  
 454 However, existing references to a deprecated object continue to function  
 455 normally.



456

457

**Figure 6: Deprecate Objects Sequence Diagram**

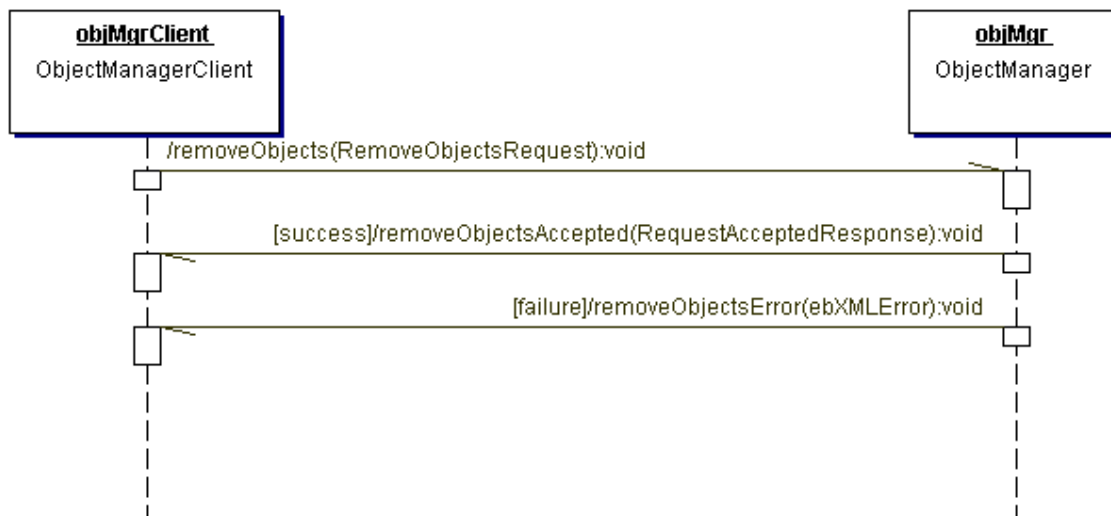
458 For details on the schema for the business documents shown in this process  
 459 refer to Appendix A.2.

### 460 7.6 The Remove Objects Request

461 This section describes the protocol of the Registry Service that allows a client to  
 462 remove one or more previously deprecated managed object contents using the  
 463 Object Manager.

464 Only if all references (e.g. Associations, Classifications, ExternalLinks) to an  
 465 object have been removed, can that object then be removed using a  
 466 RemoveObjectsRequest. Attempts to remove an object while it still has  
 467 references results in an InvalidRequestError that is returned within an  
 468 ebXMLERror message sent to the ObjectManagerClient by the ObjectManager.

469 Once an object is removed it will be not be present at all in the Registry. The  
 470 remove object protocol is expressed in UML notation as described in Appendix B.



471

472 **Figure 7: Remove Objects Sequence Diagram**

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

### 475 8 Object Query Management Service

476 This section describes the capabilities of the Registry Service that allow a client  
 477 (ObjectQueryManagerClient) to search for or query ManagedObjects in the  
 478 ebXML Registry using the ObjectQueryManager interface of the Registry.

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

482 **8.1 Browse and Drill Down Query Support**

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

486 **8.1.1 Get Root Classification Nodes Request**

487 An ObjectQueryManagerClient sends this request to get a list of root  
 488 ClassificationNodes defined in the repository. Root classification nodes are  
 489 defined as nodes that have no parent. Note that it is possible to specify a  
 490 namePattern attribute that can filter on the name attribute of the root  
 491 ClassificationNodes using a wildcard pattern defined by SQL-92 LIKE clause as  
 492 defined by [SQL].



493  
 494

**Figure 8: Get Root Classification Nodes Sequence Diagram**



495

496 **Figure 9: Get Root Classification Nodes Asynchronous Sequence Diagram**

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

499 **8.1.2 Get Classification Tree Request**

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



507  
 508 **Figure 10: Get Classification Tree Sequence Diagram**



509  
 510 **Figure 11: Get Classification Tree Asynchronous Sequence Diagram**

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

### 513 8.1.3 Get Classified Objects Request

514 An ObjectQueryManagerClient sends this request to get a list of  
515 ManagedObjects that are classified by all of the specified ClassificationNodes (or  
516 any of their descendants), as specified by the ObjectRefList in the request.

517 It is possible to get ManagedObjects based on matches with multiple  
518 classifications. Note that specifying a ClassificationNode is implicitly specifying a  
519 logical OR with all descendants of the specified ClassificationNode.

520 When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it  
521 should return Objects that are:

- 522 1. Either directly classified by the specified ClassificationNode
- 523 2. Or are directly classified by a descendant of the specified  
524 ClassificationNode

#### 525 8.1.3.1 Get Classified Objects Request Example



526

527 Figure 12: A Sample Geography Classification

528 Let us say a classification tree has the structure shown in Figure 12:

529 ?? If the Geography node is specified in the GetClassifiedObjectsRequest then  
530 the GetClassifiedObjectsResponse should include all ManagedObjects that  
531 are directly classified by Geography *or* North America *or* US *or* Asia *or* Japan  
532 *or* Korea *or* Europe *or* Germany.

533 ?? If the Asia node is specified in the GetClassifiedObjectsRequest then the  
534 GetClassifiedObjectsResponse should include all ManagedObjects that are  
535 directly classified by Asia *or* Japan *or* Korea.

536 ?? If the Japan *and* Korea nodes are specified in the  
537 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should  
538 include all ManagedObjects that are directly classified by both Japan *and*  
539 Korea.

540 ?? If the North America *and* Asia node is specified in the  
541 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should  
542 include all ManagedObjects that are directly classified by (North America *or*  
543 US) *and* (Asia *or* Japan *or* Korea).

544

## 545 **8.2 Ad Hoc Query Support**

546 The Registry supports an Ad hoc query capability that is designed for Registry  
547 clients that demand more complex query capability. The ad hoc query interface  
548 allows a client to submit complex queries using a declarative query language.

### 549 **8.2.1 Query Language Syntax**

550 [Note] The query syntax may change in a future version  
551 of this document due to a lack of consensus  
552 within the Registry team on the choice of query  
553 syntax.

554 The ad hoc query language syntax of the Registry is defined by a stylized use of  
555 a proper subset of the "SELECT" statement of SQL-92 query language as  
556 defined by [SQL]. The exact syntax of the Registry query language is defined by  
557 the BNF grammar in Appendix C.

558 Note that the use of a subset of SQL syntax for ad hoc queries does not imply a  
559 requirement to use relational databases in a Registry implementation. Its purpose  
560 is to declaratively define a query on metadata in the Registry, based on classes  
561 and attributes defined by [RIM].

562 In a future version of this specification, the W3C XML Query Language may be  
563 considered as an alternate query syntax when it reaches the recommendation  
564 stage.

### 565 **8.2.2 Query Syntax Binding To [RIM]**

566 Registry queries are defined based upon the query syntax in in Appendix C and a  
567 fixed logical schema defined by [RIM]. The following section define this binding.

#### 568 **8.2.2.1 Interface and Class Binding**

569 Interface and class names in [RIM] map to table references in the query syntax.  
570 Interface and class names may be used in the same way as table names in SQL.

#### 571 **8.2.2.2 Accessor Method To Attribute Binding**

572 Most of the [RIM] interfaces methods are simple get methods that map directly to  
573 attributes. For example the getName method on Object maps to a *name* attribute  
574 of type String.

#### 575 **8.2.2.3 Primitive Attributes Binding**

576 Attributes defined by [RIM] that are of primitive types (e.g. String) may be used in  
577 the same way as column names in SQL.

#### 578 **8.2.2.4 Reference Attribute Binding**

579 A few of the [RIM] interface methods return references to instances of interfaces  
580 or classes defined by [RIM]. For example, the `getAccessControlPolicy` method of  
581 the `Object` class returns a reference to an instance of an `AccessControlPolicy`.

582 In such cases the reference maps to the ID attribute for the referenced object.  
583 This is a special case of a primitive attribute mapping.

#### 584 **8.2.2.5 Collection Attribute Binding**

585 A few of the [RIM] interface methods return Collections of references to instances  
586 of interfaces or classes defined by [RIM]. For example, the `getPackages` method  
587 of the `ManagedObject` class returns a Collection of references to instances of  
588 Packages that the object is a member of.

589 The SQL IN clause may be used to test for membership of an object in such  
590 collections of references.

### 591 **8.2.3 Simple Metadata Based Queries**

592 The simplest form of an ad hoc query is based upon metadata attributes  
593 specified for a single class within [RIM]. This section gives some examples of  
594 simple metadata based queries.

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

```
598  
599 SELECT DISTINCT obj FROM Object WHERE  
600     obj.name LIKE '%bicycle%' AND  
601     obj.majorVersion >= 1 AND  
602     (obj.majorVersion >= 2 OR obj.minorVersion > 3);
```

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

### 605 **8.2.4 Classification Queries**

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

#### 608 **8.2.4.1 Identifying ClassificationNodes**

609 Like all objects in [RIM], ClassificationNodes are identified by their ID. However,  
610 they may also be identified as a path attribute that specifies an absolute path  
611 from a root classification node to the specified classification node where each  
612 path element is the name attribute of a ClassificationNode and is separated by '.'  
613 as a delimiter.

#### 614 **8.2.4.2 Getting Root Classification Nodes**

615 To get the collection of root ClassificationNodes the following query predicate  
616 must be supported:

```
617 SELECT FROM ClassificationNode WHERE parent IS NULL
```

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

#### 621 **8.2.4.3 Getting Children of Specified ClassificationNode**

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

```
624 SELECT FROM ClassificationNode WHERE parent = <id>
```

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

#### 627 **8.2.4.4 Getting Objects Classified By a ClassificationNode**

628 To get the collection of Objects classified by specified ClassificationNodes the  
629 following style of query must be supported:

```
630 SELECT DISTINCT eo  
631 FROM ExtrinsicObject eo, ClassificationNode auto, ClassificationNode geo  
632 WHERE  
633     (geo IN eo.classificationNodes AND geo.path = "Geography.Asia.Japan")  
634 AND  
635     (auto IN eo.classificationNodes AND auto.path = "Industry.Automotive")
```

636 The above query gets the collection of ExtrinsicObjects that are classified by the  
637 Automotive Industry and the Japan Geography. Note that according to the  
638 semantics defined for GetClassifiedObjectsRequest, the query will also contain  
639 any objects that are classified by descendents of the specified  
640 ClassificationNodes.

#### 641 **8.2.4.5 Getting ClassificationNodes That Classify an Object**

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

```
644 SELECT cn FROM ClassificationNode cn, ExtrinsicObject o WHERE
645     o.ID = <id> AND
646     cn IN o.classificationNodes
```

## 647 **8.2.5 Association Queries**

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

### 650 **8.2.5.1 Getting All Association With Specified Object As Its Source**

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

```
653 SELECT assoc FROM Association WHERE assoc.sourceObject = <id>;
```

### 654 **8.2.5.2 Getting All Association With Specified Object As Its Target**

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

```
657 SELECT assoc FROM Association WHERE assoc.targetObject = <id>;
```

### 658 **8.2.5.3 Getting Associated Objects Based On Association Attributes**

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

661 Select Associations that have the specified name.

```
662 SELECT assoc FROM Association WHERE
663     assoc.name = <name>;
```

664 Select Associations that have the specified source role name.

```
665 SELECT assoc FROM Association WHERE
666     assoc.sourceRole = <roleName>;
```

667 Select Associations that have the specified target role name.

```
668 SELECT assoc FROM Association WHERE
669     assoc.targetRole = <roleName>;
```

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

```
672 SELECT DISTINCT assoc FROM Association WHERE
673     assoc.associationType = <associationType>;
```

#### 674 **8.2.5.4 Complex Association Queries**

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

```
678 SELECT DISTINCT assoc FROM Association WHERE  
679     Assoc.sourceObject = <id>  
680     assoc.sourceRole = "buysFrom" AND  
681     assoc.targetRole = "sellsTo";
```

#### 682 **8.2.6 Package Queries**

683 To find all packages that a specified ExtrinsicObject belongs to, the following  
684 query is specified:

```
685 SELECT p FROM Package p, ExtrinsicObject obj WHERE  
686     obj.ID = <id> AND p IN obj.packages
```

687 To find all Association objects in a specified package, the following query is  
688 specified:

```
689 SELECT a FROM Association, Package p WHERE  
690     p.ID = <id> AND a IN p.memberObjects;
```

#### 691 **8.2.6.1 Complex Package Queries**

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

```
694 SELECT p FROM Package p, ExtrinsicObject obj WHERE  
695     obj.ID = <id> AND p IN obj.packages AND  
696     p.name LIKE '%RosettaNet%' AND  
697     p.status != 'DEPRECATED';
```

#### 698 **8.2.7 ExternalLink Queries**

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

```
701 SELECT l FROM ExternalLink, ExtrinsicObject obj WHERE  
702     obj.ID = <id> AND l IN obj.externalLinks
```

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

```
705 SELECT obj FROM ExtrinsicObject, ExternalLink l WHERE  
706     l.ID = <id> AND obj IN l.linkedObjects
```

### 707 **8.2.7.1 Complex ExternalLink Queries**

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

```
711 SELECT I FROM ExternalLink, ExtrinsicObject obj WHERE  
712     obj.ID = <id> AND I IN obj.externalLinks AND  
713     I.description LIKE '%legal%' AND  
714     I.externalURI LIKE '%http://%'
```

### 715 **8.2.8 Audit Trail Queries**

716 To get the complete collection of AuditableEvent objects for a specified  
717 ManagedObject, the following style query is specified:

```
718 SELECT ev FROM AuditableEvent, ExtrinsicObject obj WHERE  
719     obj.ID = <id> AND ev IN obj.auditTrail;
```

720

### 721 **8.2.9 Content Based Ad Hoc Queries**

722 The ad hoc query interface of the Registry supports the ability to search for  
723 content based not only on metadata that catalogs the content but also the data  
724 contained within the content itself. For example it is possible for a client to submit  
725 a query that searches for all Collaboration Party Profiles that define a role named  
726 "seller" within a RoleName element in the CPP document itself.

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

#### 728 **8.2.9.1 Automatic Classification of XML Content**

729 Content-based queries are indirectly supported through the existing classification  
730 mechanism supported by the Registry.

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

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

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

742 [Note] This approach is conceptually similar to the

743 way databases support indexed retrieval. DBAs  
744 define indexes on tables in the schema. When  
745 data is added to the table, the data gets  
746 automatically indexed.

#### 747 **8.2.9.2 Index Definition**

748 This section describes how the logical indexes are defined in the  
749 SubmittedObject element defined in the Registry DTD. The complete Registry  
750 DTD is specified in Appendix A.2.

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

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

- 757 1. classificationNode: This attribute references a specific ClassificationNode  
758 by its ID.
- 759 2. contentIdentifier: This attribute identifies a specific data element within the  
760 document instances of the schema using an XPATH path expression as  
761 defined by [XPT].

#### 762 **8.2.9.3 Example Of Index Definition**

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

```
766 <ClassificationIndex  
767     classificationNode='id-for-role-classification-scheme'  
768     contentIdentifier='/Role//RoleName'  
769 />
```

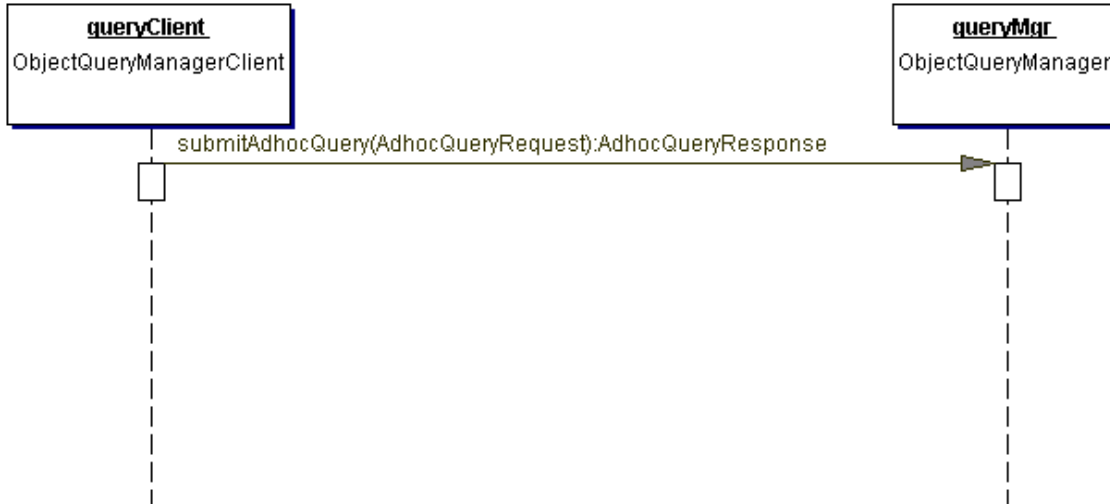
#### 770 **8.2.9.4 Example of Automatic Classification**

771 Assume that a CPP is submitted that defines two roles as "seller" and "buyer."  
772 When the CPP is submitted it will automatically be classified by two  
773 ClassificationNodes named "buyer" and "seller" that are both children of the  
774 ClassificationNode (e.g. a node named Role) specified in the classificationNode  
775 attribute of the ClassificationIndex. Note that if either of the two  
776 ClassificationNodes named "buyer" and "seller" did not previously exist, the  
777 ObjectManager would automatically create these ClassificationNodes.

778 **8.2.10 Ad Hoc Query Request/Response**

779 A client submits an ad hoc query to the ObjectQueryManager by sending an  
 780 AdhocQueryRequest. The AdhocQueryRequest contains the query string in the  
 781 queryString attribute.

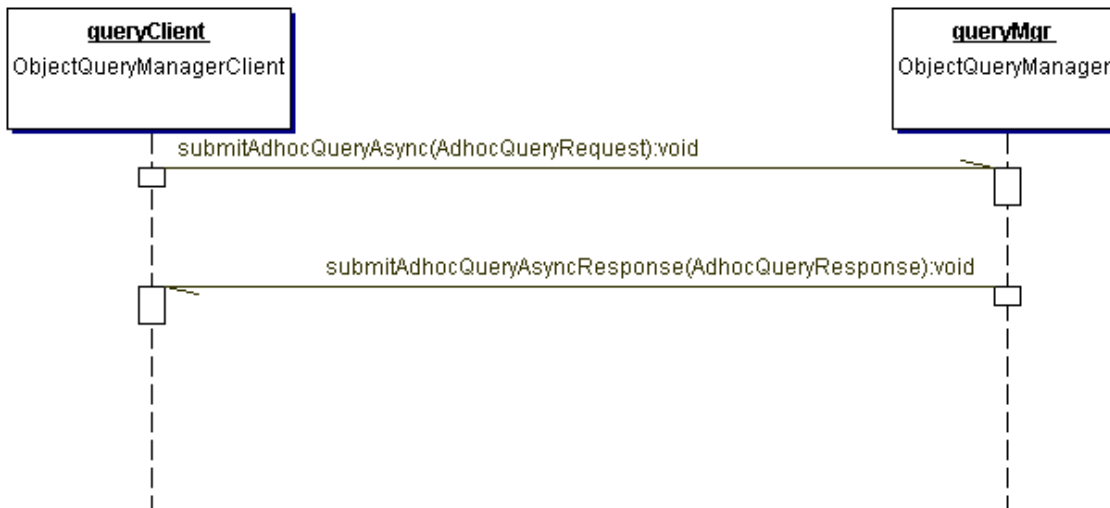
782 The ObjectQueryManager sends an AdhocQueryResponse either synchronously  
 783 or asynchronously back to the client. The AdhocQueryResponse return a  
 784 collection of objects whose element type is in the set of element types  
 785 represented by the leaf nodes of the ManagedObject hierarchy in [RIM].



786

787

**Figure 13: Submit Ad Hoc Query Sequence Diagram**



788

789

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

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

## 792 8.3 Content Retrieval

793 A client retrieves content from the Registry by sending the GetContentRequest to  
794 the ObjectQueryManager. The GetContentRequest specifies a list of Object  
795 references for Objects that need to be retrieved. The ObjectQueryManager  
796 returns the specified content by sending a GetContentResponse message to the  
797 ObjectQueryManagerClient interface of the client. If there are no errors  
798 encountered, the GetContentResponse message includes the specified content  
799 as additional payloads within the message. In addition to the  
800 GetContentResponse payload, there is one additional payload for each content  
801 that was requested. If there are errors encountered, the GetContentResponse  
802 payload includes an ebXMLError and there are no additional content specific  
803 payloads.

### 804 8.3.1 Identification Of Content Payloads

805 Since the GetContentResponse message may include several managed object  
806 contents as additional payloads, it is necessary to have a way to identify each  
807 payload in the message. To facilitate this identification, the Registry must do the  
808 following:

809     ?? Use the ID for each managed object content as the DocumentLabel  
810         element in the DocumentReference for that object in the Manifest element  
811         of the ebXMLHeader.

### 812 8.3.2 GetContentResponse Message Structure

813 The following message fragment illustrates the structure of the  
814 GetContentResponse Message that is returning a Collection of CPPs as a result  
815 of a GetContentRequest that specified the IDs for the requested objects. Note  
816 that the ID for each object retrieved in the message as additional payloads is  
817 used as its DocumentLabel in the Manifest of the ebXMLHeader.

```
818 ...  
819 --7250537.978150567601.JavaMail.najmi.irian  
820 ...  
821 <ebXMLHeader MessageType="Normal" Version="1.0">  
822   <Manifest>  
823     <DocumentReference>  
824       <DocumentLabel>GetContentsResponse</DocumentLabel>  
825       <DocumentId>6835fb:e3be512ac8:-8000</DocumentId>  
826     </DocumentReference>  
827     <DocumentReference>  
828       <DocumentLabel> ID for CPP content #1 </DocumentLabel>  
829       <DocumentId>....</DocumentId>
```

```

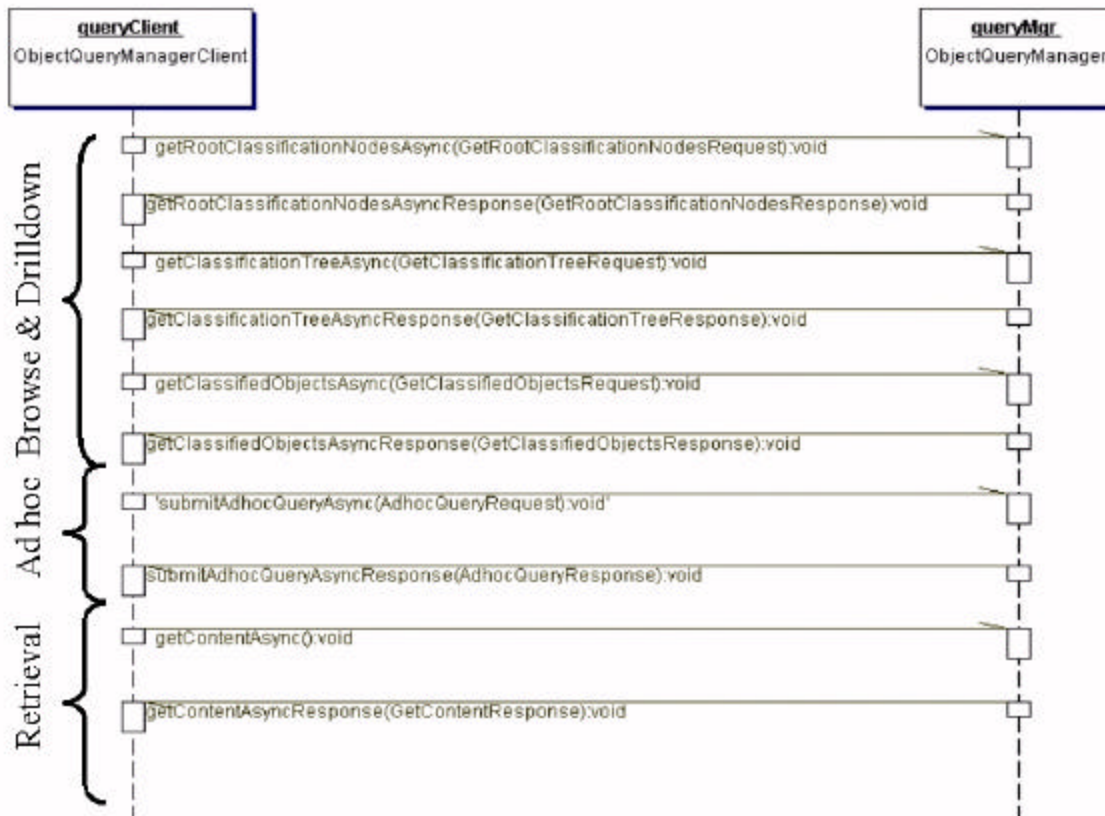
831     </DocumentReference>
832     <DocumentReference>
833         <DocumentLabel> ID for CPP content #2 </DocumentLabel>
834         <DocumentId>... </DocumentId>
835     </DocumentReference>
836 </Manifest>
837 <Header>
838 ...
839 </Header>
840 --7250537.978150567601.JavaMail.najmi.irian
841 Content-Type: application/xml
842 Content-Description: GetContentsResponse
843 Content-ID: 6835fb:e3be512ac8:-7ffc
844 Content-Length: 97
845
846 <?xml version="1.0" encoding="UTF-8"?>
847 <GetContentsResponse />
848
849 --7250537.978150567601.JavaMail.najmi.irian
850 Content-Type: application/xml
851 Content-Description: ID for CPP content #1
852 Content-ID: ....
853 ...
854 <CPP>
855 ...
856 </CPP>
857 --7250537.978150567601.JavaMail.najmi.irian
858 Content-Type: application/xml
859 Content-Description: ID for CPP content #2
860 Content-ID: ....
861 ...
862 <CPP>
863 ...
864 </CPP>
865 --7250537.978150567601.JavaMail.najmi.irian--
866
867

```

868

## 869 **8.4 Query And Retrieval: Typical Sequence**

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



872

873

Figure 15: Typical Query and Retrieval Sequence

874

## 9 Registry Security

875

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

876

877

878

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

879

880

881

882

## 883 **9.1 Integrity of Registry Content**

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

### 890 **9.1.1 Message Payload Signature**

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

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

895 ?? The content's veracity can be ascertained by its association with a  
896 specific submitting organization

## 897 **9.2 Authentication**

898 The Registry must be able to authenticate the identity of the Principal associated  
899 with client requests. Authentication is required to identify the ownership of  
900 content as well as to identify what "privileges" a Principal can be assigned with  
901 respect to the specific objects in the Registry.

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

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

### 909 **9.2.1 Message Header Signature**

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

915 **9.3 Confidentiality**

916 **9.3.1 On-the-wire Message Confidentiality**

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

920 **9.3.2 Confidentiality of Registry Content**

921 In the current version of this specification, there are no provisions for  
 922 confidentiality of Registry content. All content submitted to the Registry may be  
 923 discovered and read by *any* client. The Registry must decrypt any submitted  
 924 content after it has been received and prior to storing it in its repository.

925 **9.4 Authorization**

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

931 **9.4.1 Pre-defined Roles For Registry Users**

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

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

933 **9.4.2 Default Access Control Policies**

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

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

939

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).

940

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

- 942 ?? The Registry must implement the default AccessControlPolicy and
- 943 associate it with all Objects in the Registry
- 944 ?? Anyone can publish content, but needs to be authenticated
- 945 ?? Anyone can access the content without requiring authentication
- 946 ?? The ContentOwner has access to all methods for Registry Objects owned
- 947 by them
- 948 ?? The RegistryAdministrator has access to all methods on all Registry
- 949 Objects
- 950 ?? Unauthenticated clients can access all read-only (getXXX) methods
- 951 ?? At the time of content submission, the Registry must assign the default
- 952 ContentOwner role to the Submitting Organization (SO) as authenticated
- 953 by the credentials in the submission message. In the current version of
- 954 this specification, it will be the DN as identified by the certificate
- 955 ?? Clients that browse the Registry need not use certificates. The Registry
- 956 must assign the default RegistryGuest role to such clients.

957

## 958 **Appendix A Schemas and DTD Definitions**

959 The following are definitions for the various ebXML Message payloads described  
960 in this document.

### 961 **A.1 ebXMLError Message DTD**

962 See [ERR] for ebXMLError Message DTD.

## 963 A.2 ebXML Registry DTD

```

964 <?xml version='1.0' encoding='UTF-8' ?>
965
966 <!--Generated by XML Authority-->
967 <!-- $Header: /jse/jaxr/schema/Registry.dtd,v 1.7 2001/01/10 17:56:28 najmi Exp
968 $ -->
969 <!ENTITY % errorSchema SYSTEM "ebXMLError.dtd">
970
971 %errorSchema;
972
973 <!ENTITY % VersionAttribute " version CDATA #REQUIRED">
974
975 <!ENTITY % ObjectAttributes " description CDATA #IMPLIED
976 ID CDATA #REQUIRED
977 name CDATA #REQUIRED">
978
979 <!ENTITY % ManagedObjectAttributes " %ObjectAttributes;
980 status (SUBMITTED | APPROVED | DEPRECATED )
981 'SUBMITTED'
982 majorVersion CDATA '1'
983 minorVersion CDATA '0'">
984
985 <!ELEMENT ManagedObject EMPTY>
986 <!ATTLIST ManagedObject %ManagedObjectAttributes; >
987 <!ELEMENT ExtrinsicObject EMPTY>
988 <!ATTLIST ExtrinsicObject %ManagedObjectAttributes;
989 contentURN CDATA #IMPLIED
990 mimeType CDATA #IMPLIED
991 objectType (PARTY_AGREEMENT |
992 PARTY_PROFILE |
993 PROCESS |
994 ROLE |
995 SERVICE_INTERFACE |
996 SOFTWARE_COMPONENT |
997 TRANSPORT |
998 UML_MODEL |
999 UNKNOWN |
1000 XML_SCHEMA ) #REQUIRED
1001 opaque CDATA 'false'
1002 a-dtype NMTOKENS 'opaque boolean' >
1003 <!--
1004 A ClassificationIndex is specified on SCHEMA ExtrinsicObjects to define
1005 an automatic Classification of instance objects of the schema using
1006 the specified classificationNode as parent and a ClassificationNode

```

```
1007 created or selected by the object content as selected by the contentIdentifier
1008 -->
1009 <!ELEMENT ClassificationIndex EMPTY>
1010 <!ATTLIST ClassificationIndex %ObjectAttributes;
1011             classificationNode CDATA #REQUIRED
1012             contentIdentifier CDATA #REQUIRED >
1013 <!-- ClassificationIndexList contains new ClassificationIndexes -->
1014 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
1015
1016 <!ENTITY % IntrinsicObjectAttributes " %ManagedObjectAttributes;">
1017
1018 <!ELEMENT IntrinsicObject EMPTY>
1019 <!ATTLIST IntrinsicObject %ManagedObjectAttributes; >
1020 <!-- Leaf classes that reflect the concrete classes in RIM -->
1021 <!ELEMENT ManagedObjectList (Association | Classification |
1022 ClassificationNode | ExternalLink | Organization | ExtrinsicObject)*>
1023
1024 <!-- Reference to an Object via its URN specified by it ID attribute -->
1025 <!ELEMENT ObjectRef EMPTY>
1026 <!ATTLIST ObjectRef uuid CDATA #REQUIRED >
1027 <!ELEMENT ObjectRefList (ObjectRef)*>
1028
1029 <!--
1030 An ExternalLink specifies a link from a ManagedObject and an external URI
1031
1032 The sourceObjectRef is ref to the ManagedObject
1033
1034 The sourceObjectRef is optional when Association is defined as part of
1035 a SubmittedObject.
1036 -->
1037 <!ELEMENT ExternalLink EMPTY>
1038 <!ATTLIST ExternalLink %IntrinsicObjectAttributes;
1039             sourceObjectRef CDATA #IMPLIED
1040             uri CDATA #IMPLIED >
1041 <!-- ExternalLinkList contains new ExternalLinks or refs to pre-existing
1042 ExternalLinks -->
1043 <!ELEMENT ExternalLinkList (ExternalLink | ObjectRef)*>
1044
1045 <!--
1046 An Association specifies references to two previously submitted
1047 managed objects.
1048
1049 The sourceObjectRef is ref to the sourceObject in association
1050 The targetObjectRef is ref to the targetObject in association
1051
```

1052 The sourceObjectRef is optional when Association is defined part of  
 1053 a SubmittedObject.

1054 -->  
 1055 <!ELEMENT Association EMPTY>  
 1056 <!ATTLIST Association %IntrinsicObjectAttributes;  
 1057       fromLabel        CDATA   #IMPLIED  
 1058       toLabel         CDATA   #IMPLIED  
 1059       associationType   (CLASSIFIED\_BY |  
 1060                          CONTAINED\_BY |  
 1061                          CONTAINS |  
 1062                          EXTENDS |  
 1063                          IMPLEMENTS |  
 1064                          INSTANCE\_OF |  
 1065                          RELATED\_TO |  
 1066                          SUPERSEDED\_BY |  
 1067                          SUPERSEDES |  
 1068                          USED\_BY |  
 1069                          USES ) #FIXED 'RELATED\_TO'  
 1070       bidirection       CDATA   'false'  
 1071       sourceObjectRef   CDATA   #REQUIRED  
 1072       targetObjectRef   CDATA   #REQUIRED  
 1073       a-dtype           NMTOKENS 'bidirection boolean' >  
 1074 <!ELEMENT AssociationList (Association)\*>  
 1075  
 1076 <!--  
 1077 A Classification specifies references to two previously submitted  
 1078 managed objects.  
 1079  
 1080 The sourceObjectRef is ref to the sourceObject in Classification  
 1081 The targetObjectRef is ref to the targetObject in Classification  
 1082  
 1083 The sourceObjectRef is optional when Classification is defined as part of  
 1084 a SubmittedObject.

1085 -->  
 1086 <!ELEMENT Classification EMPTY>  
 1087 <!ATTLIST Classification %IntrinsicObjectAttributes;  
 1088       sourceObjectRef   CDATA   #REQUIRED  
 1089       targetObjectRef   CDATA   #REQUIRED >  
 1090 <!ELEMENT ClassificationList (Classification)\*>  
 1091  
 1092 <!ELEMENT Package EMPTY>  
 1093 <!ATTLIST Package %IntrinsicObjectAttributes; >  
 1094 <!-- PackageList contains new Packages or refs to pre-existing Packages -->  
 1095 <!ELEMENT PackageList (Package | ObjectRef)\*>  
 1096

```
1097 <!ENTITY % TelephoneNumberAttributes " areaCode CDATA #REQUIRED
1098         contryCode CDATA #REQUIRED
1099         extension CDATA #IMPLIED
1100         number CDATA #REQUIRED
1101         url CDATA #IMPLIED">
1102
1103 <!ELEMENT TelephoneNumber EMPTY>
1104 <!ATTLIST TelephoneNumber %TelephoneNumberAttributes; >
1105 <!ELEMENT FaxNumber EMPTY>
1106 <!ATTLIST FaxNumber %TelephoneNumberAttributes; >
1107 <!ELEMENT MobileTelephoneNumber EMPTY>
1108 <!ATTLIST MobileTelephoneNumber %TelephoneNumberAttributes; >
1109 <!-- PostalAddress -->
1110 <!ELEMENT PostalAddress EMPTY>
1111 <!ATTLIST PostalAddress city CDATA #REQUIRED
1112         country CDATA #REQUIRED
1113         postalCode CDATA #REQUIRED
1114         state CDATA #REQUIRED
1115         street CDATA #REQUIRED >
1116 <!-- PersonName -->
1117 <!ELEMENT PersonName EMPTY>
1118 <!ATTLIST PersonName firstName CDATA #REQUIRED
1119         middleName CDATA #REQUIRED
1120         lastName CDATA #REQUIRED >
1121 <!-- Contact -->
1122 <!ELEMENT Contact (PostalAddress , PersonName , FaxNumber? ,
1123 TelephoneNumber , MobileTelephoneNumber? )>
1124 <!ATTLIST Contact email CDATA #REQUIRED >
1125 <!-- Organization -->
1126 <!ELEMENT Organization (PostalAddress , Contact , FaxNumber? ,
1127 TelephoneNumber )>
1128 <!ATTLIST Organization %IntrinsicObjectAttributes;
1129         parent CDATA #IMPLIED >
1130 <!--
1131 ClassificationNode is used to submit a Classification tree to the Registry.
1132 Note that this is a recursive schema definition.
1133
1134 The parent attribute of a node in tree is implied by the enclosing
1135 ClassificationNode
1136 The children nodes of a node are implied by enclosing immediate child elements
1137 of type ClassificationNode.
1138 -->
1139 <!ELEMENT ClassificationNode EMPTY>
1140 <!ATTLIST ClassificationNode %IntrinsicObjectAttributes;>
1141
```

```
1142 <!--
1143 parent is the URN to the parent node. parent is optional if ClassificationNode is
1144 enclosed
1145 in a parent ClassificationNode or if it is a root ClassificationNode
1146 -->
1147 <!ATTLIST ClassificationNode parent          CDATA #IMPLIED>
1148
1149 <!ELEMENT ClassificationNodeList (ClassificationNode)*>
1150
1151 <!--
1152 End information model mapping.
1153
1154 Begin Registry Services Interface
1155 -->
1156 <!ELEMENT RequestAcceptedResponse EMPTY>
1157 <!ATTLIST RequestAcceptedResponse %VersionAttribute;
1158          xml:lang      NMTOKEN #REQUIRED
1159          interfacedId  CDATA #REQUIRED
1160          requestMessage CDATA #REQUIRED
1161          actionId      CDATA #REQUIRED >
1162 <!--
1163 The SubmittedObject provides meta data for submitted object
1164 Note object being submitted is in a separate document that is not
1165 in this DTD.
1166 -->
1167 <!ELEMENT SubmitObjectsRequest (SubmittedObject+ )>
1168 <!ATTLIST SubmitObjectsRequest %VersionAttribute; >
1169 <!--
1170 The ExtrinsicObject provides meta data about the object being submitted
1171 ClassificationList can be optionally specified to define Classifications
1172 for the SubmittedObject
1173
1174 AssociationList can be optionally specified to define Associations
1175 for the SubmittedObject
1176
1177 The ExternalLinkList provides zero or more external objects related to
1178 the object being submitted.
1179 -->
1180 <!ELEMENT SubmittedObject (ExtrinsicObject? , ClassificationIndexList? ,
1181 ClassificationList? , AssociationList? , ExternalLinkList? , PackageList? )>
1182
1183 <!--
1184 The ObjectRefList is the list of
1185 refs to the managed objects being approved.
1186 -->
```

```

1187 <!ELEMENT ApproveObjectsRequest (ObjectRefList )>
1188 <!ATTLIST ApproveObjectsRequest %VersionAttribute; >
1189 <!--
1190 The ObjectRefList is the list of
1191 refs to the managed objects being deprecated.
1192 -->
1193 <!ELEMENT DeprecateObjectsRequest (ObjectRefList )>
1194 <!ATTLIST DeprecateObjectsRequest %VersionAttribute; >
1195 <!--
1196 The ObjectRefList is the list of
1197 refs to the managed objects being removed
1198 -->
1199 <!ELEMENT RemoveObjectsRequest (ObjectRefList )>
1200 <!ATTLIST RemoveObjectsRequest %VersionAttribute; >
1201 <!ELEMENT GetRootClassificationNodesRequest EMPTY>
1202 <!ATTLIST GetRootClassificationNodesRequest %VersionAttribute;>
1203
1204 <!--
1205 The namePattern follows SQL-92 syntax for the pattern specified in
1206 LIKE clause. It allows for selecting only those root nodes that match
1207 the namePattern. The default value of '*' matches all root nodes.
1208 -->
1209 <!ATTLIST GetRootClassificationNodesRequest namePattern CDATA "*">
1210
1211 <!--
1212 The response includes a ClassificationNodeList which has zero or more
1213 ClassificationNodes
1214 -->
1215 <!ELEMENT GetRootClassificationNodesResponse (ClassificationNodeList |
1216 ebXMLError )>
1217 <!ATTLIST GetRootClassificationNodesResponse %VersionAttribute; >
1218 <!--
1219 Get the classification tree under the ClassificationNode specified parentRef.
1220
1221 If depth is 1 just fetch immediate child
1222 nodes, otherwise fetch the descendant tree upto the specified depth level.
1223 If depth is 0 that implies fetch entire sub-tree
1224 -->
1225 <!ELEMENT GetClassificationTreeRequest EMPTY>
1226 <!ATTLIST GetClassificationTreeRequest %VersionAttribute;
1227             parent CDATA #REQUIRED
1228             depth CDATA '1' >
1229 <!--
1230 The response includes a ClassificationNodeList which includes only
1231 immediate ClassificationNode children nodes if depth attribute in

```

1232 GetClassificationTreeRequest was 1, otherwise the decendent nodes  
1233 upto specified depth level are returned.

```
1234 -->  
1235 <!ELEMENT GetClassificationTreeResponse (ClassificationNodeList |  
1236 ebXMLError )>  
1237 <!ATTLIST GetClassificationTreeResponse %VersionAttribute; >  
1238 <!--
```

1239 Get refs to all managed objects that are classified by all the  
1240 ClassificationNodes specified by ObjectRefList.  
1241 Note this is an implicit logical AND operation

```
1242 -->  
1243 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList )>  
1244  
1245 <!--
```

1246 objectType attribute can specify the type of objects that the registry  
1247 client is interested in, that is classified by this ClassificationNode.  
1248 It is a String that matches a choice in the type attribute of ExtrinsicObject.  
1249 The default value of '\*' implies that client is interested in all types  
1250 of managed objects that are classified by the specified ClassificationNode.

```
1251 -->  
1252 <!--
```

1253 The response includes a ManagedObjectList which has zero or more  
1254 ManagedObjects that are classified by the ClassificationNodes  
1255 specified in the ObjectRefList in GetClassifiedObjectsRequest.

```
1256 -->  
1257 <!ELEMENT GetClassifiedObjectsResponse (ManagedObjectList | ebXMLError  
1258 )>  
1259 <!ATTLIST GetClassifiedObjectsResponse %VersionAttribute; >  
1260 <!--
```

1261 An Ad hoc query request specifies a query string as defined by [RS] in the  
1262 queryString attribute

```
1263 -->  
1264 <!ELEMENT AdhocQueryRequest EMPTY>  
1265 <!ATTLIST AdhocQueryRequest %VersionAttribute;  
1266           queryString CDATA #REQUIRED >  
1267 <!--
```

1268 The response includes a ManagedObjectList which has zero or more  
1269 ManagedObjects that match the query specified in AdhocQueryRequest.

```
1270 -->  
1271 <!ELEMENT AdhocQueryResponse (ManagedObjectList | ebXMLError )>  
1272 <!ATTLIST AdhocQueryResponse %VersionAttribute; >  
1273 <!--
```

1274 Gets the actual content (not metadata) specified by the ObjectRefList

```
1275 -->  
1276 <!ELEMENT GetContentRequest (ObjectRefList )>
```

```
1277 <!ATTLIST GetContentRequest %VersionAttribute; >
1278 <!--
1279 The GetObjectsResponse will have no sub-elements if there were no errors.
1280 The actual contents will be in the other payloads of the message.
1281 If any errors were encountered the message will contain the ebXMLError and
1282 the content payloads will be empty.
1283 -->
1284 <!ELEMENT GetContentResponse (ebXMLError? )>
1285 <!ATTLIST GetContentResponse %VersionAttribute; >
1286 <!--
1287 The contrived root node
1288 -->
1289 <!ELEMENT RootElement (RequestAcceptedResponse | ebXMLError |
1290 SubmitObjectsRequest | ApproveObjectsRequest | DeprecateObjectsRequest |
1291 RemoveObjectsRequest | GetRootClassificationNodesRequest |
1292 GetRootClassificationNodesResponse | GetClassificationTreeRequest |
1293 GetClassificationTreeResponse | GetClassifiedObjectsRequest |
1294 GetClassifiedObjectsResponse | AdhocQueryRequest | AdhocQueryResponse |
1295 GetContentRequest | GetContentResponse )>
```

## 1296 **Appendix B Interpretation of UML Diagrams**

1297 This section describes in *abstract terms* the conventions used to define ebXML  
1298 business process description in UML.

### 1299 **B.1 UML Class Diagram**

1300 A UML class diagram is used to describe the Service Interfaces (as defined by  
1301 [CPA]) required to implement an ebXML Registry Services and clients. See  
1302 Figure 1 on page 12 for an example. The UML class diagram contains:

1303

- 1304 1. A collection of UML interfaces where each interface represents a Service  
1305 Interface for a Registry service.
- 1306 2. Tabular description of methods on each interface where each method  
1307 represents an Action (as defined by [CPA]) within the Service Interface  
1308 representing the UML interface.
- 1309 3. Each method within a UML interface specifies one or more parameters,  
1310 where the type of each method argument represents the ebXML message  
1311 type that is exchanged as part of the Action corresponding to the method.  
1312 Multiple arguments imply multiple payload documents within the body of  
1313 the corresponding ebXML message.

1314 **B.2 UML Sequence Diagram**

1315 A UML sequence diagram is used to specify the business protocol representing  
 1316 the interactions between the UML interfaces for a Registry specific ebXML  
 1317 business process. A UML sequence diagram provides the necessary information  
 1318 to determine the sequencing of messages, request to response association as  
 1319 well as request to error response association as described by [CPA].

1320 Each sequence diagram shows the sequence for a specific conversation protocol  
 1321 as method calls from the requestor to the responder. Method invocation may be  
 1322 synchronous or asynchronous based on the UML notation used on the arrow-  
 1323 head for the link. A half arrow-head represents asynchronous communication. A  
 1324 full arrow-head represents synchronous communication.

1325 Each method invocation may be followed by a response method invocation from  
 1326 the responder to the requestor to indicate the ResponseName for the previous  
 1327 Request. Possible error response is indicated by a conditional response method  
 1328 invocation from the responder to the requestor. See Figure 4 on page 19 for an  
 1329 example.

1330 **Appendix C BNF for Query Syntax Grammar**

1331 The following BNF defines the grammar for the registry query syntax. This  
 1332 grammer is a proper sub-set of SQL-92 as defined by [SQL].

```

1333 /*****
1334 * The Registry Query (Subset of SQL-92) grammar starts here
1335 *****/
1336
1337
1338 SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
1339
1340 SQLSelectCols = ( "ALL" | "DISTINCT" ) * [ "ID" ]
1341
1342 SQLTableList = SQLTableRef ( "," SQLTableRef ) *
1343 SQLTableRef = "ID"
1344
1345 SQLWhere = "WHERE" SQLOrExpr
1346
1347 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
1348 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *
1349 SQLNotExpr = [ "NOT" ] SQLCompareExpr
1350
1351 SQLCompareExpr =
1352     (SQLColRef "IS") SQLIsClause
1353 | SQLTerm [ SQLCompareExprRight ]
    
```

```
1354
1355
1356 SQLCompareExprRight =
1357     SQLLikeClause
1358     | SQLInClause
1359     | SQLCompareOp SQLTerm
1360
1361 SQLCompareOp =
1362     "="
1363     | "!="
1364     | ">"
1365     | ">="
1366     | "<"
1367     | "<="
1368
1369
1370 SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
1371 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
1372 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
1373 SQLPattern = STRING_LITERAL
1374
1375 SQLLiteral =
1376     STRING_LITERAL
1377     | INTEGER_LITERAL
1378     | FLOATING_POINT_LITERAL
1379
1380 SQLColRef = SQLLvalue
1381 SQLLvalue = SQLLvalueTerm
1382 SQLLvalueTerm = ID ( "." ID )*
1383
1384
1385 SQLTerm = "(" SQLOrExpr ")"
1386     | SQLColRef
1387     | SQLLiteral
1388
1389
1390 SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
1391 SQLLValueElement = "NULL" | SQLSelect
1392
1393 INTEGER_LITERAL = ([ "0"-"9" ])+
1394
1395 FLOATING_POINT_LITERAL =
1396     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ (EXPONENT)?
1397     | "." ([ "0"-"9" ])+ (EXPONENT)?
1398     | ([ "0"-"9" ])+ EXPONENT
```

```
1399 | ([\"0\"-\"9\"])+ (EXPONENT)?
1400
1401 EXPONENT = [\"e\", \"E\"] ([\"+\", \"-\"])? ([\"0\"-\"9\"])+
1402
1403 STRING_LITERAL: \"\" (~[\"'\"])* ( \"\" (~[\"'\"])* ) * \"\"
1404
1405 ID = ( <LETTER> )+ ( \"_\" | \"$\" | \"#\" | <DIGIT> | <LETTER> ) *
1406 LETTER = [\"A\"-\"Z\", \"a\"-\"z\"]
1407 DIGIT = [\"0\"-\"9\"]
1408
```

## 1409 **Appendix D Security Implementation Guideline**

1410 This section provides a suggested blueprint for how security processing may be  
1411 implemented in the Registry. It is meant to be illustrative not prescriptive.  
1412 Registries may choose to have different implementations as long as they support  
1413 the default security roles and authorization rules described in this document.

### 1414 **D.1 Authentication**

- 1415 1. As soon as a message is received, the first work is the authentication. A  
1416 principal object is created.
- 1417 2. If the message is signed, it is verified (including the validity of the certificate)  
1418 and the DN of the certificate becomes the identity of the principal. Then the  
1419 Registry is searched for the principal and if found, the roles and groups are  
1420 filled in.
- 1421 3. If the message is not signed, an empty principal is created with the role  
1422 RegistryGuest. This step is for symmetry and to decouple the rest of the  
1423 processing.
- 1424 4. Then the message is processed for the command and the objects it will act on

### 1425 **D.2 Authorization**

1426 For every object, the access controller will iterate through all the  
1427 AccessControlPolicy objects with the object and see if there is a chain through  
1428 the permission objects to verify that the requested method is permitted for the  
1429 Principal. If any of the permission objects which the object is associated with has  
1430 a common role, or identity, or group with the principal, the action is permitted.

### 1431 **D.3 Registry Bootstrap**

1432 When a Registry is newly created, a default Principal object should be created  
1433 with the identity of the Registry Admin's certificate DN with a role RegistryAdmin.  
1434 This way, any message signed by the Registry Admin will get all the privileges.

1435 When a Registry is newly created, a singleton instance of AccessControlPolicy is  
1436 created as the default AccessControlPolicy. This includes the creation of the  
1437 necessary Permission instances as well as the Privileges and Privilege attributes.

### 1438 **D.4 Content Submission – Client Responsibility**

1439 The Registry client has to sign the contents before submission – otherwise the  
1440 content will be rejected.

### 1441 **D.5 Content Submission – Registry Responsibility**

- 1442 1. Like any other request, the client will be first authenticated. In this case, the  
1443 Principal object will get the DN from the certificate.
- 1444 2. As per the request in the message, the ManagedObject will be created.
- 1445 3. The ManagedObject is assigned the singleton default AccessControlPolicy.
- 1446 4. If a principal with the identity of the SO is not available, an identity object with  
1447 the SO's DN is created
- 1448 5. A principal with this identity is created

### 1449 **D.6 Content Delete/Deprecate – Client Responsibility**

1450 The Registry client has to sign the payload (not entire message) before  
1451 submission, for authentication purposes; otherwise, the request will be  
1452 rejected

### 1453 **D.7 Content Delete/Deprecate – Registry Responsibility**

- 1454 1. Like any other request, the client will be first authenticated. In this case, the  
1455 Principal object will get the DN from the certificate. As there will be a principal  
1456 with this identity in the Registry, the Principal object will get all the roles from  
1457 that object
- 1458 2. As per the request in the message (delete or deprecate), the appropriate  
1459 method in the Object will be accessed.

- 1460 3. The access controller performs the authorization by iterating through the
- 1461 Permission objects associated with this object via the singleton default
- 1462 AccessControlPolicy.
- 1463 4. If authorization succeeds then the action will be permitted. Otherwise an error
- 1464 response is sent back with a suitable AuthorizationException error message.

1465 **Appendix E Terminology Mapping**

1466 While every attempt has been made to use the same terminology used in other  
 1467 works there are some terminology differences.

1468 The following table shows the terminology mapping between this specification  
 1469 and that used in other specifications and working groups.

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

1470 **Table 1: Terminology Mapping Table**

1471 **10 References**

1472 [GLS] ebXML Glossary, [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)

1473 [TA] ebXML Technical Architecture

1474 [http://www.ebxml.org/specdrafts/ebXML\\_TA\\_v1.0.pdf](http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf)

- 1475 [OAS] OASIS Information Model  
1476 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 1477 [ISO] ISO 11179 Information Model  
1478 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>  
1479
- 1480 [BDM] Registry and Repository: Business Domain Model  
1481 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 1482 [RIM] ebXML Registry Information Model  
1483 [http://www.ebxml.org/project\\_teams/registry/private/registryInfoModelv0.54.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf)
- 1484 [BPM] ebXML Business Process Metamodel Specification Schema  
1485 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 1486 [CPA] Trading-Partner Specification  
1487 [http://www.ebxml.org/project\\_teams/trade\\_partner/private/](http://www.ebxml.org/project_teams/trade_partner/private/)
- 1488 [CTB] Context table informal document from Core Components
- 1489 [MS] ebXML Messaging Service Specification, Version 0.21  
1490 [http://ebxml.org/project\\_teams/transport/private/ebXML\\_Messaging\\_Service\\_Specification\\_v0-21.pdf](http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf)
- 1491 [ERR] ebXML TRP Error Handling Specification  
1492 [http://www.ebxml.org/project\\_teams/transport/ebXML\\_Message\\_Service\\_Specification\\_v-0.8\\_001110.pdf](http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf)
- 1493 [SEC] ebXML Security Specification  
1494 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 1495 [XPT] XML Path Language (XPath) Version 1.0  
1496 <http://www.w3.org/TR/xpath>
- 1497 [SQL] Structured Query Language (FIPS PUB 127-2)  
1498 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>

## 1499 **11 Disclaimer**

1500 The views and specification expressed in this document are those of the authors  
1501 and are not necessarily those of their employers. The authors and their  
1502 employers specifically disclaim responsibility for any problems arising from  
1503 correct or incorrect implementation or use of this design.  
1504

**1504 12 Contact Information****1505 Team Leader**

1506 Name: Scott Nieman  
1507 Company: Norstan Consulting  
1508 Street: 5101 Shady Oak Road  
1509 City, State, Postal Code: Minnetonka, MN 55343  
1510 Country: USA  
1511 Phone: 952.352.5889  
1512 Email: Scott.Nieman@Norstan

1513

**1514 Vice Team Lead**

1515 Name: Yutaka Yoshida  
1516 Company: Sun Microsystems  
1517 Street: 901 San Antonio Road, MS UMPK17-102  
1518 City, State, Postal Code: Palo Alto, CA 94303  
1519 Country: USA  
1520 Phone: 650.786.5488  
1521 Email: Yutaka.Yoshida@eng.sun.com

1522

**1523 Editor**

1524 Name: Farrukh S. Najmi  
1525 Company: Sun Microsystems  
1526 Street: 1 Network Dr., MS BUR02-302  
1527 City, State, Postal Code: Burlington, MA, 01803-0902  
1528 Country: USA  
1529 Phone: 781.442.0703  
1530 Email: najmi@east.sun.com

1531

1532

1532 **Copyright Statement**

1533 Copyright © ebXML 2000. All Rights Reserved.

1534

1535 This document and translations of it may be copied and furnished to others, and  
1536 derivative works that comment on or otherwise explain it or assist in its  
1537 implementation may be prepared, copied, published and distributed, in whole or  
1538 in part, without restriction of any kind, provided that the above copyright notice  
1539 and this paragraph are included on all such copies and derivative works.  
1540 However, this document itself may not be modified in any way, such as by  
1541 removing the copyright notice or references to the Internet Society or other  
1542 Internet organizations, except as needed for the purpose of developing Internet  
1543 standards in which case the procedures for copyrights defined in the Internet  
1544 Standards process must be followed, or as required to translate it into languages  
1545 other than English.

1546

1547 The limited permissions granted above are perpetual and will not be revoked by  
1548 ebXML or its successors or assigns.

1549

1550 This document and the information contained herein is provided on an  
1551 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR  
1552 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE  
1553 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
1554 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
1555 PARTICULAR PURPOSE.