



Creating A Single Global Electronic Market

1

2 **ebXML Registry Services**

3 **ebXML Registry Project Team**

4 **Working Draft 1/20/2001**

5 **This version: Version 0.84**

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/RegistryServicesSpecificationv0.84.pdf

19

20 ***Latest version:***

21 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.84.pdf

22

23 ***Previous version:***

24 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.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 Document7

68 3.2 General Conventions7

69 3.3 Audience.....7

70 3.4 Related Documents7

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

72 4.1 Goals8

73 4.2 Caveats and Assumptions8

74 **5 System Overview 8**

75 5.1 What The ebXML Registry Does8

76 5.2 How The ebXML Registry Works9

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 Submitted9

80 5.6 Buyer Discovers The Seller9

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	8	Object Query Management Service	21
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	8.2.2.6	Semantic Constraints On Query Syntax	26
116	8.2.3	Simple Metadata Based Queries	26
117	8.2.4	Classification Queries	27
118	8.2.4.1	Identifying ClassificationNodes	27
119	8.2.4.2	Getting Root Classification Nodes	27
120	8.2.4.3	Getting Children of Specified ClassificationNode	27
121	8.2.4.4	Getting Objects Classified By a ClassificationNode	27
122	8.2.4.5	Getting ClassificationNodes That Classify an Object... ..	28
123	8.2.5	Association Queries	28
124	8.2.5.1	Getting All Association With Specified Object As Its	
125		Source	28
126	8.2.5.2	Getting All Association With Specified Object As Its	
127		Target	28
128	8.2.5.3	Getting Associated Objects Based On Association	
129		Attributes	28
130	8.2.5.4	Complex Association Queries	29
131	8.2.6	Package Queries	29
132	8.2.6.1	Complex Package Queries	29
133	8.2.7	ExternalLink Queries	29
134	8.2.7.1	Complex ExternalLink Queries	30
135	8.2.8	Audit Trail Queries	30
136	8.2.9	Content Based Ad Hoc Queries	30
137	8.2.9.1	Automatic Classification of XML Content	30
138	8.2.9.2	Index Definition	31
139	8.2.9.3	Example Of Index Definition	31
140	8.2.9.4	Example of Automatic Classification	31
141	8.2.10	Ad Hoc Query Request/Response	32
142	8.3	Content Retrieval	33

143 8.3.1 Identification Of Content Payloads 33

144 8.3.2 GetContentResponse Message Structure 33

145 8.4 Query And Retrieval: Typical Sequence 34

146 **9 Registry Security 35**

147 9.1 Integrity of Registry Content 36

148 9.1.1 Message Payload Signature 36

149 9.2 Authentication 36

150 9.2.1 Message Header Signature 36

151 9.3 Confidentiality 37

152 9.3.1 On-the-wire Message Confidentiality 37

153 9.3.2 Confidentiality of Registry Content 37

154 9.4 Authorization 37

155 9.4.1 Pre-defined Roles For Registry Users 37

156 9.4.2 Default Access Control Policies 37

157 **Appendix A Schemas and DTD Definitions 38**

158 A.1 ebXMLError Message DTD 38

159 A.2 ebXML Registry DTD 39

160 **Appendix B Interpretation of UML Diagrams 46**

161 B.1 UML Class Diagram 46

162 B.2 UML Sequence Diagram 47

163 **Appendix C BNF for Query Syntax Grammar 47**

164 **Appendix D Security Implementation Guideline 49**

165 D.1 Authentication 49

166 D.2 Authorization 50

167 D.3 Registry Bootstrap 50

168 D.4 Content Submission – Client Responsibility 50

169 D.5 Content Submission – Registry Responsibility 50

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

171 D.7 Content Delete/Deprecate – Registry Responsibility 51

172 **Appendix E Terminology Mapping 51**

173 **10 References 52**

174 **11 Disclaimer 53**

175 **12 Contact Information 54**

176 **Copyright Statement 55**

177 **Table of Figures**

178 Figure 1: ebXML Registry Interfaces 12

179 Figure 2: Life Cycle of a Managed Object 18

180 Figure 3: Submit Objects Sequence Diagram 19

181 Figure 4: Approve Objects Sequence Diagram..... 20

182 Figure 5: Deprecate Objects Sequence Diagram 20

183 Figure 6: Remove Objects Sequence Diagram..... 21

184 Figure 7: Get Root Classification Nodes Sequence Diagram..... 22

185 Figure 8: Get Root Classification Nodes Asynchronous Sequence Diagram 23

186 Figure 9: Get Classification Tree Sequence Diagram..... 23

187 Figure 10: Get Classification Tree Asynchronous Sequence Diagram 23

188 Figure 11: A Sample Geography Classification 24

189 Figure 12: Submit Ad Hoc Query Sequence Diagram 32

190 Figure 13: Submit Ad Hoc Query Asynchronous Sequence Diagram..... 32

191 Figure 14: Typical Query and Retrieval Sequence..... 35

192 **Table of Tables**

193 Table 1: Terminology Mapping Table 51

194

195

195 **3 Introduction**

196 **3.1 Summary of Contents of Document**

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

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

202 **3.2 General Conventions**

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

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

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

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

213 **3.3 Audience**

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

216 o Implementers of ebXML Registry Services

217 o Implementers of ebXML Registry Clients

218 **3.4 Related Documents**

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

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

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

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

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

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

230

231 **4 Design Objectives**

232 **4.1 Goals**

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

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

239 **4.2 Caveats and Assumptions**

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

243 It is assumed that:

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

251 **5 System Overview**

252 **5.1 What The ebXML Registry Does**

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

258 **5.2 How The ebXML Registry Works**

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

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

270 **5.3 Schema Documents Are Submitted**

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

275 **5.4 Business Process Documents Are Submitted**

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

280 **5.5 Seller's Collaboration Protocol Profile Is Submitted**

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

286 **5.6 Buyer Discovers The Seller**

287 The buyer browses the Registry using classification schemes defined within the
288 Registry using a Registry Browser GUI tool to discover a suitable seller. For
289 example the buyer may look for all parties that are in the Automotive Industry,
290 play a seller role, support the 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. Although these interaction protocols are specific to the Registry, they
312 are identical in nature to the interactions between two parties conducting B2B
313 message communication using the ebXML Messaging Service as defined by
314 [MS] and [CPA].

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

317 **6.1 Implicit CPA Between Clients And Registry**

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

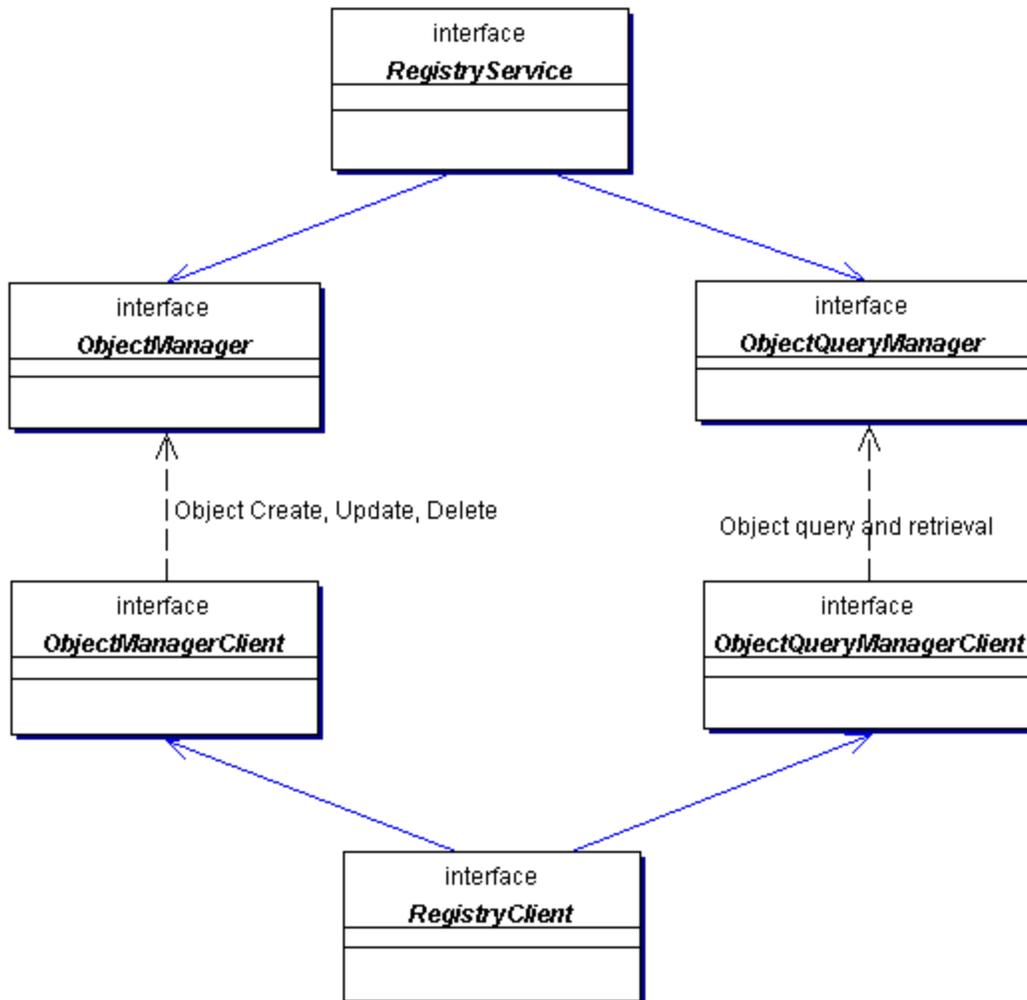
320 Similarly, this specification defines a CPA between a Registry client and the
321 Registry. Typical B2B interactions in ebXML require an explicit CPA to be
322 negotiated between parties. However, the CPA between clients and the Registry
323 is an implicit CPA that describes the interfaces that the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

326 **6.2 Client To Registry Communication Bootstrapping**

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

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

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



341
342

Figure 1: ebXML Registry Interfaces

343 **6.3 Interfaces Exposed By The Registry**

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

346 **6.3.1 Interface *RegistryService***

347

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

351

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.

352

353 **6.3.2 Interface *ObjectManager***

354

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

359

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.

360 **6.3.3 Interface *ObjectQueryManager***

361

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

366

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

367 **6.4 Interfaces Exposed By Registry Clients**

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

369 **6.4.1 Interface *RegistryClient***

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

375

Method Summary	
<u>ObjectManagerClient</u>	<u>getObjectManagerClient()</u> Returns the ObjectManagerClient interface implemented by the client.
<u>ObjectQueryManagerClient</u>	<u>getObjectQueryManagerClient()</u> Returns the ObjectQueryManagerClient interface implemented by the client.

376

377 **6.4.2 Interface *ObjectManagerClient***

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

382

Method Summary	
void	<u>approveObjectsAccepted(RequestAcceptedResponse resp)</u> Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.
void	<u>approveObjectsError(ebXMLError error)</u> 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	removeObjectsError (ebXMLError error) Notifies client that a previously submitted RemoveObjectsRequest 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.

383

384 **6.4.3 Interface *ObjectQueryManagerClient***

385

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

390

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.

391 **7 Object Management Service**

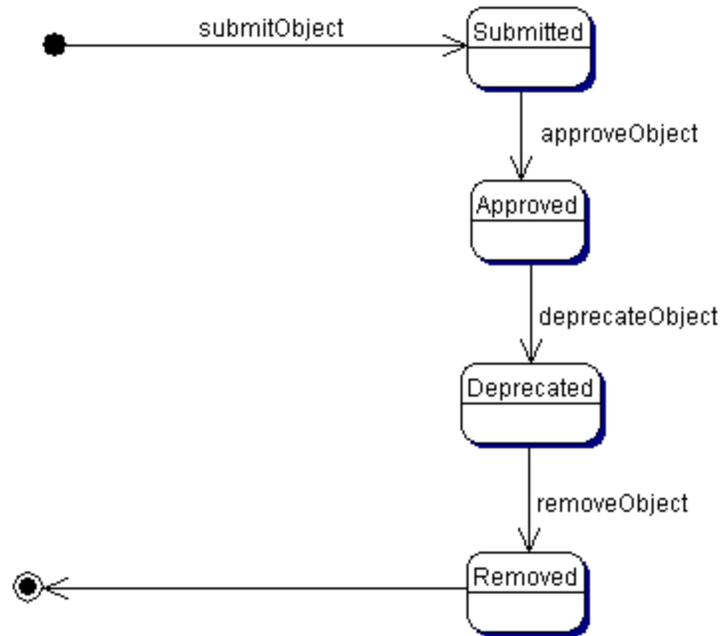
392 This section defines the Object Management service of the Registry. The Object
 393 Management Service is a sub-service of the Registry service. It provides the
 394 functionality required by RegistryClients to manage the life cycle of managed
 395 object contents (e.g. XML documents required for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

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

402 **7.1 Life Cycle of a Managed Object**

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

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



408

409

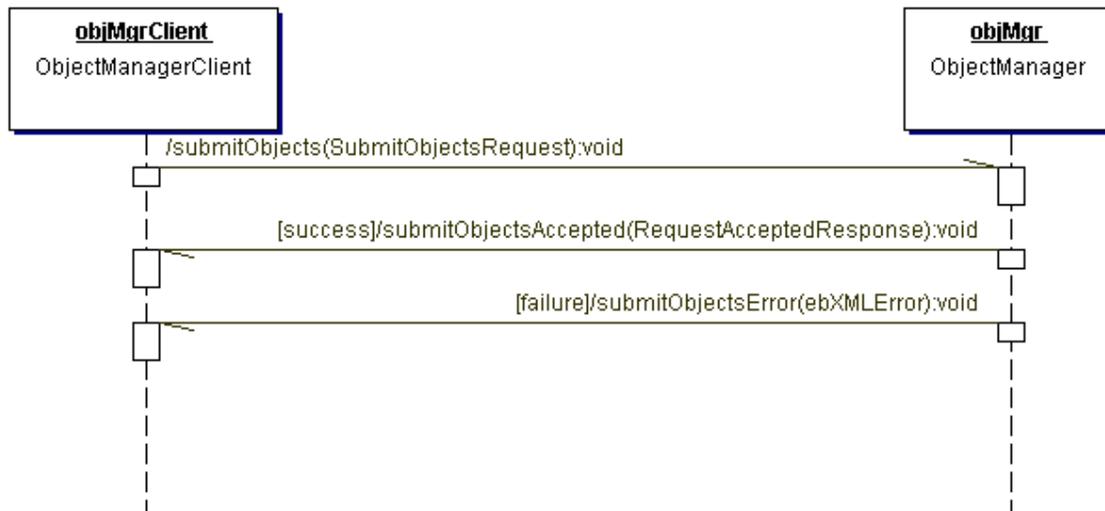
Figure 2: Life Cycle of a Managed Object

410 **7.2 Object Attributes**

411 A managed object content is associated with a set of standard metadata defined
 412 as attributes of the Object class and its sub-classes as described in [RIM]. These
 413 attributes reside outside of the actual managed object content and catalog
 414 descriptive information about the managed object content. XML DTD elements
 415 called ExtrinsicObject and IntrinsicObject (See Appendix A.2 for details.) are
 416 defined that encapsulates all object metadata attributes defined in [RIM] as
 417 attributes of the DTD elements.

418 **7.3 The Submit Objects Protocol**

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



423

424

Figure 3: Submit Objects Sequence Diagram

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

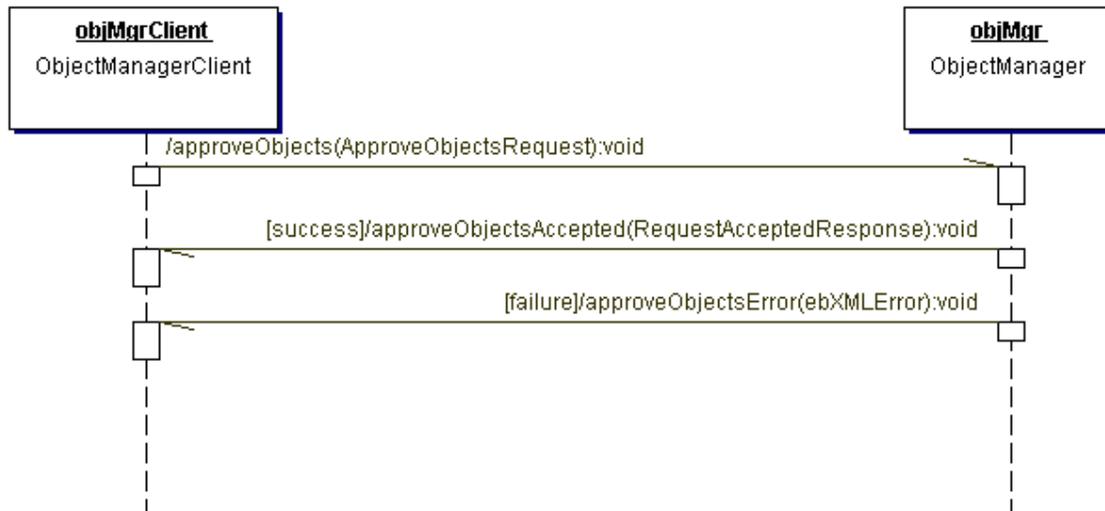
427 The SubmitObjectRequest message includes 1 or more SubmittedObject
 428 elements.

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

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

438 **7.4 The Approve Objects Request**

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



444

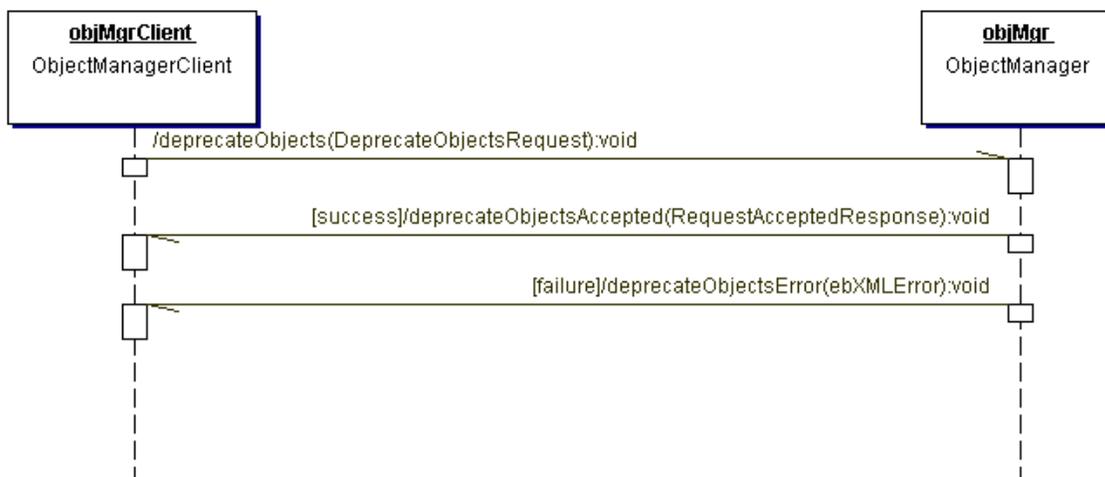
445

Figure 4: Approve Objects Sequence Diagram

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

448 7.5 The Deprecate Objects Request

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



455

456

Figure 5: Deprecate Objects Sequence Diagram

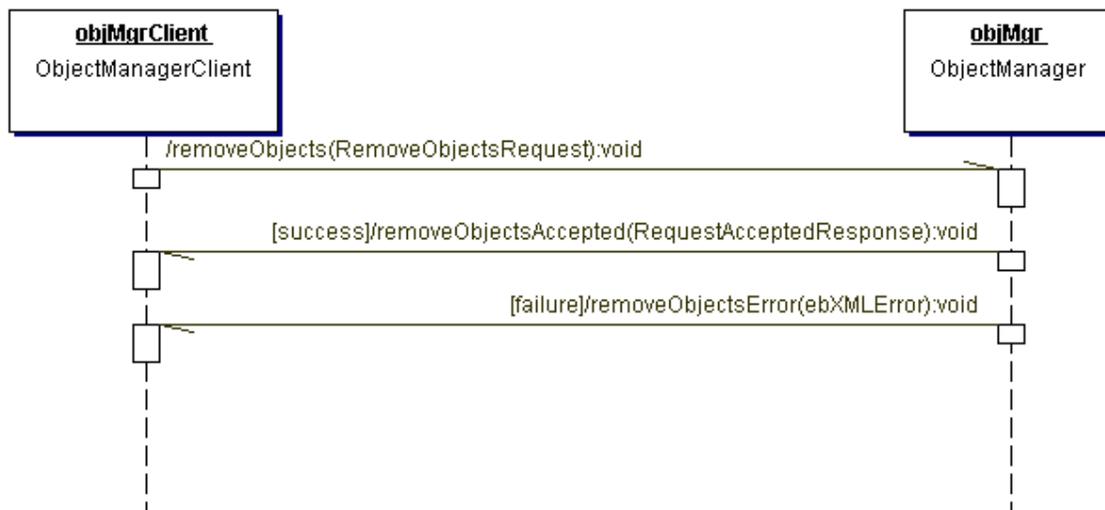
457 For details on the schema for the business documents shown in this process
 458 refer to Appendix A.2.

459 7.6 The Remove Objects Request

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

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

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



470

471 **Figure 6: Remove Objects Sequence Diagram**

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

474 8 Object Query Management Service

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

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

481 **8.1 Browse and Drill Down Query Support**

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

485 **8.1.1 Get Root Classification Nodes Request**

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



492
 493

Figure 7: Get Root Classification Nodes Sequence Diagram



494

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

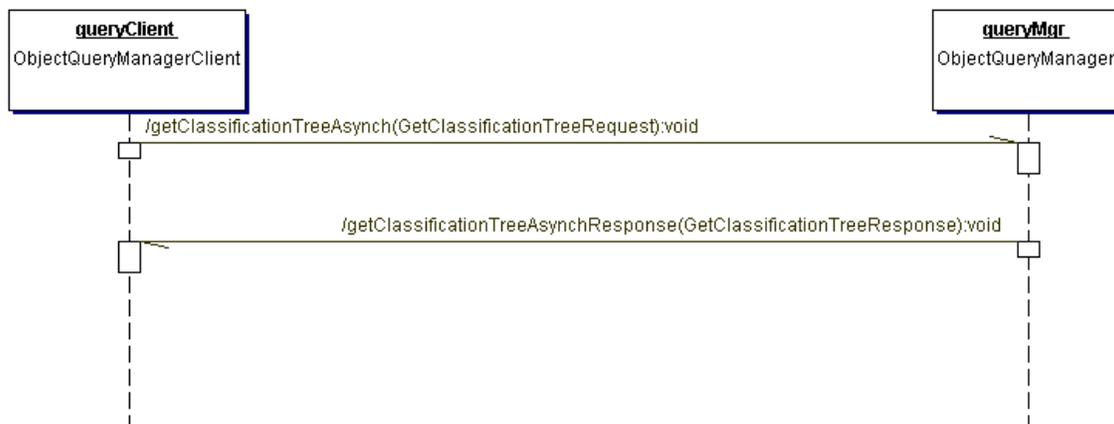
496 For details on the schema for the business documents shown in this process
 497 refer to Appendix A.2.

498 **8.1.2 Get Classification Tree Request**

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



506 **Figure 9: Get Classification Tree Sequence Diagram**
 507



508 **Figure 10: Get Classification Tree Asynchronous Sequence Diagram**
 509

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

512 8.1.3 Get Classified Objects Request

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

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

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

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

524 8.1.3.1 Get Classified Objects Request Example



525

526 Figure 11: A Sample Geography Classification

527 Let us say a classification tree has the structure shown in Figure 11:

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

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

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

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

543

544 **8.2 Ad Hoc Query Support**

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

548 **8.2.1 Query Language Syntax**

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

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

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

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

564 **8.2.2 Query Syntax Binding To [RIM]**

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

567 **8.2.2.1 Interface and Class Binding**

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

570 **8.2.2.2 Accessor Method To Attribute Binding**

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

574 **8.2.2.3 Primitive Attributes Binding**

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

577 **8.2.2.4 Reference Attribute Binding**

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

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. The use of Collection attributes are
589 restricted to be only within the IN clause of the query grammar.

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

592 **8.2.2.6 Semantic Constraints On Query Syntax**

593 This section defines simplifying constraints on the query syntax that cannot be
594 expressed in the BNF for the query syntax. These constraints must be applied in
595 the semantic analysis of the query.

- 596 1. Class names and attribute names must be processed in a case insensitive
597 manner.
- 598 2. Collection attributes must only be specified within an IN clause.

599 **8.2.3 Simple Metadata Based Queries**

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

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

```
606  
607 SELECT DISTINCT obj FROM ExtrinsicObject WHERE  
608     obj.name LIKE '%Acme%' AND  
609     obj.majorVersion >= 1 AND  
610     (obj.majorVersion >= 2 OR obj.minorVersion > 3);
```

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

613 **8.2.4 Classification Queries**

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

616 **8.2.4.1 Identifying ClassificationNodes**

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

622 **8.2.4.2 Getting Root Classification Nodes**

623 To get the collection of root ClassificationNodes the following query predicate
624 must be supported:

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

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

629 **8.2.4.3 Getting Children of Specified ClassificationNode**

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

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

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

635 **8.2.4.4 Getting Objects Classified By a ClassificationNode**

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

```
638 SELECT DISTINCT eo  
639 FROM ExtrinsicObject eo, ClassificationNode auto, ClassificationNode geo  
640 WHERE  
641     (geo IN (eo.classificationNodes) AND geo.path = 'Geography.Asia.Japan')  
642 AND  
643     (auto IN (eo.classificationNodes) AND auto.path = 'Industry.Automotive')
```

644 The above query gets the collection of ExtrinsicObjects that are classified by the
645 Automotive Industry and the Japan Geography. Note that according to the
646 semantics defined for GetClassifiedObjectsRequest, the query will also contain
647 any objects that are classified by descendents of the specified
648 ClassificationNodes.

649 **8.2.4.5 Getting ClassificationNodes That Classify an Object**

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

```
652 SELECT cn FROM ClassificationNode cn, ExtrinsicObject o WHERE  
653     o.ID = <id> AND  
654     cn IN (o.classificationNodes)
```

655 **8.2.5 Association Queries**

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

658 **8.2.5.1 Getting All Association With Specified Object As Its Source**

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

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

662 **8.2.5.2 Getting All Association With Specified Object As Its Target**

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

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

666 **8.2.5.3 Getting Associated Objects Based On Association Attributes**

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

669 Select Associations that have the specified name.

```
670 SELECT assoc FROM Association WHERE  
671     assoc.name = <name>
```

672 Select Associations that have the specified source role name.

```
673 SELECT assoc FROM Association WHERE  
674     assoc.sourceRole = <roleName>
```

675 Select Associations that have the specified target role name.

```
676 SELECT assoc FROM Association WHERE  
677     assoc.targetRole = <roleName>
```

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

```
680 SELECT DISTINCT assoc FROM Association WHERE  
681     assoc.associationType = <associationType>
```

682 **8.2.5.4 Complex Association Queries**

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

```
686 SELECT DISTINCT assoc FROM Association WHERE  
687     Assoc.sourceObject = <id> AND  
688     assoc.sourceRole = 'buysFrom' AND  
689     assoc.targetRole = 'sellsTo'
```

690 **8.2.6 Package Queries**

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

```
693 SELECT p FROM Package p, ExtrinsicObject obj WHERE  
694     obj.ID = <id> AND p IN (obj.packages)
```

695 To find all Association objects in a specified package, the following query is
696 specified:

```
697 SELECT a FROM Association, Package p WHERE  
698     p.ID = <id> AND a IN (p.memberObjects)
```

699 **8.2.6.1 Complex Package Queries**

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

```
702 SELECT p FROM Package p, ExtrinsicObject obj WHERE  
703     obj.ID = <id> AND p IN (obj.packages) AND  
704     p.name LIKE '%RosettaNet%' AND  
705     p.status != 'DEPRECATED'
```

706 **8.2.7 ExternalLink Queries**

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

```
709 SELECT l FROM ExternalLink, ExtrinsicObject obj WHERE  
710     obj.ID = <id> AND l IN (obj.externalLinks)
```

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

```
713 SELECT obj FROM ExtrinsicObject, ExternalLink l WHERE  
714     l.ID = <id> AND obj IN (l.linkedObjects)
```

715 **8.2.7.1 Complex ExternalLink Queries**

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

```
719 SELECT I FROM ExternalLink, ExtrinsicObject obj WHERE  
720     obj.ID = <id> AND I IN (obj.externalLinks) AND  
721     I.description LIKE '%legal%' AND  
722     I.externalURI LIKE '%http://%'
```

723 **8.2.8 Audit Trail Queries**

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

```
726 SELECT ev FROM AuditableEvent, ExtrinsicObject obj WHERE  
727     obj.ID = <id> AND ev IN (obj.auditTrail)
```

728 **8.2.9 Content Based Ad Hoc Queries**

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

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

735 **8.2.9.1 Automatic Classification of XML Content**

736 Content-based queries are indirectly supported through the existing classification
737 mechanism supported by the Registry.

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

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

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

749 [Note] This approach is conceptually similar to the
750 way databases support indexed retrieval. DBAs

751 define indexes on tables in the schema. When
752 data is added to the table, the data gets
753 automatically indexed.

754 **8.2.9.2 Index Definition**

755 This section describes how the logical indexes are defined in the
756 SubmittedObject element defined in the Registry DTD. The complete Registry
757 DTD is specified in Appendix A.2.

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

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

- 764 1. classificationNode: This attribute references a specific ClassificationNode
765 by its ID.
- 766 2. contentIdentifier: This attribute identifies a specific data element within the
767 document instances of the schema using an XPATH path expression as
768 defined by [XPT].

769 **8.2.9.3 Example Of Index Definition**

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

```
773 <ClassificationIndex  
774     classificationNode='id-for-role-classification-scheme'  
775     contentIdentifier='/Role//RoleName'  
776 />
```

777 **8.2.9.4 Example of Automatic Classification**

778 Assume that a CPP is submitted that defines two roles as "seller" and "buyer."
779 When the CPP is submitted it will automatically be classified by two
780 ClassificationNodes named "buyer" and "seller" that are both children of the
781 ClassificationNode (e.g. a node named Role) specified in the classificationNode
782 attribute of the ClassificationIndex. Note that if either of the two
783 ClassificationNodes named "buyer" and "seller" did not previously exist, the
784 ObjectManager would automatically create these ClassificationNodes.

785 **8.2.10 Ad Hoc Query Request/Response**

786 A client submits an ad hoc query to the ObjectQueryManager by sending an
 787 AdhocQueryRequest. The AdhocQueryRequest contains the query string in the
 788 queryString attribute.

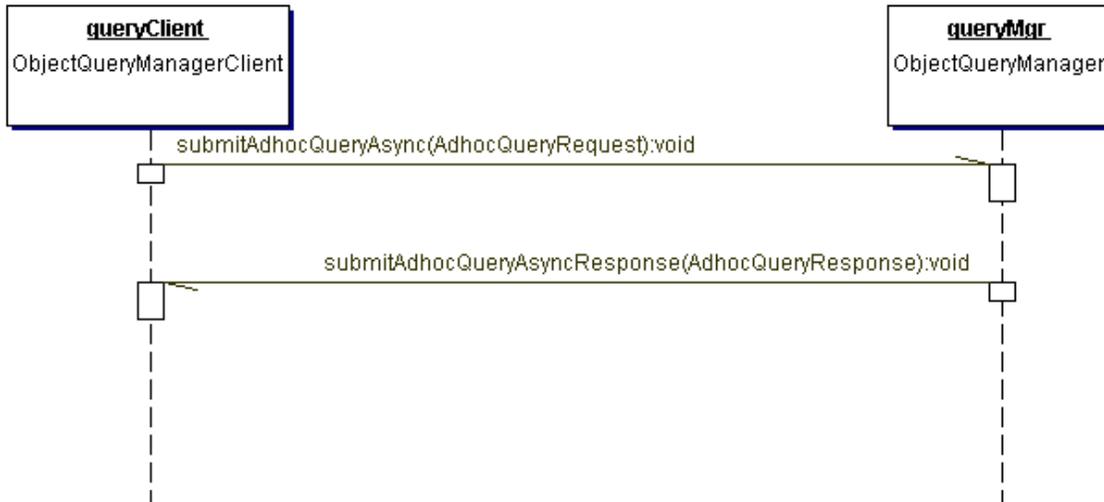
789 The ObjectQueryManager sends an AdhocQueryResponse either synchronously
 790 or asynchronously back to the client. The AdhocQueryResponse return a
 791 collection of objects whose element type is in the set of element types
 792 represented by the leaf nodes of the ManagedObject hierarchy in [RIM].



793

794

Figure 12: Submit Ad Hoc Query Sequence Diagram



795

796

Figure 13: Submit Ad Hoc Query Asynchronous Sequence Diagram

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

799 8.3 Content Retrieval

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

811 8.3.1 Identification Of Content Payloads

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

816 ?? Use the ID for each ManagedObject instance that describes the managed
817 object content as the DocumentLabel element in the DocumentReference
818 for that object in the Manifest element of the ebXMLHeader.

819 8.3.2 GetContentResponse Message Structure

820 The following message fragment illustrates the structure of the
821 GetContentResponse Message that is returning a Collection of CPPs as a result
822 of a GetContentRequest that specified the IDs for the requested objects. Note
823 that the ID for each object retrieved in the message as additional payloads is
824 used as its DocumentLabel in the Manifest of the ebXMLHeader.

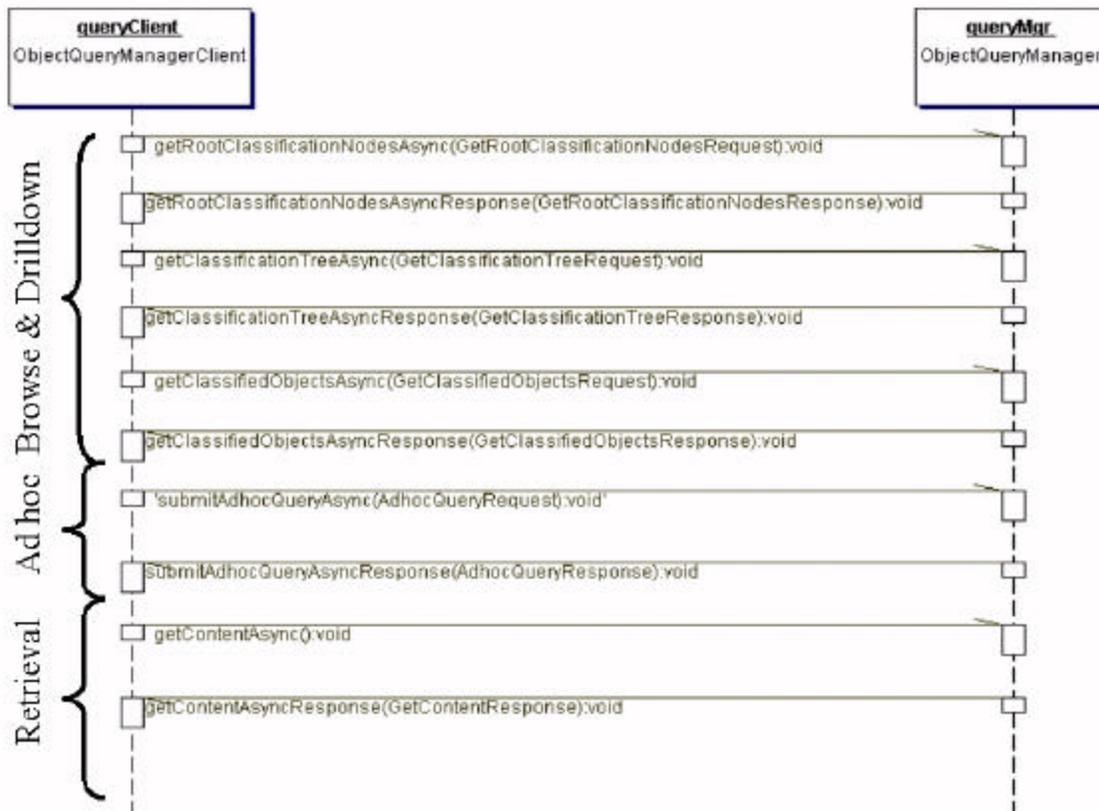
```
825 ...
826 --7250537.978150567601.JavaMail.najmi.irian
827 ...
828 <ebXMLHeader MessageType="Normal" Version="1.0">
829   <Manifest>
830     <DocumentReference>
831       <DocumentLabel>GetContentsResponse</DocumentLabel>
832       <DocumentId>6835fb:e3be512ac8:-8000</DocumentId>
833     </DocumentReference>
834     <DocumentReference>
835       <DocumentLabel> ID for CPP content #1 </DocumentLabel>
836       <DocumentId>....</DocumentId>
```

```
838     </DocumentReference>
839     <DocumentReference>
840         <DocumentLabel> ID for CPP content #2 </DocumentLabel>
841         <DocumentId>... </DocumentId>
842     </DocumentReference>
843 </Manifest>
844 <Header>
845     ...
846 </Header>
847 --7250537.978150567601.JavaMail.najmi.irian
848 Content-Type: application/xml
849 Content-Description: GetContentsResponse
850 Content-ID: 6835fb:e3be512ac8:-7ffc
851 Content-Length: 97
852
853 <?xml version="1.0" encoding="UTF-8"?>
854 <GetContentsResponse />
855
856 --7250537.978150567601.JavaMail.najmi.irian
857 Content-Type: application/xml
858 Content-Description: ID for CPP content #1
859 Content-ID: ....
860 ...
861 <CPP>
862 ...
863 </CPP>
864 --7250537.978150567601.JavaMail.najmi.irian
865 Content-Type: application/xml
866 Content-Description: ID for CPP content #2
867 Content-ID: ....
868 ...
869 <CPP>
870 ...
871 </CPP>
872 --7250537.978150567601.JavaMail.najmi.irian--
873
874
```

875

876 **8.4 Query And Retrieval: Typical Sequence**

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



879

880

Figure 14: Typical Query and Retrieval Sequence

881

9 Registry Security

882

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].

883

884

885

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.

886

887

888

889

890 **9.1 Integrity of Registry Content**

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

897 **9.1.1 Message Payload Signature**

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

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

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

904 **9.2 Authentication**

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

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

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

916 **9.2.1 Message Header Signature**

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

922 **9.3 Confidentiality**

923 **9.3.1 On-the-wire Message Confidentiality**

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

927 **9.3.2 Confidentiality of Registry Content**

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

932 **9.4 Authorization**

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

938 **9.4.1 Pre-defined Roles For Registry Users**

939 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.

940 **9.4.2 Default Access Control Policies**

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

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

946

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).

947

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

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

964

965 **Appendix A Schemas and DTD Definitions**

966 The following are definitions for the various ebXML Message payloads described
967 in this document.

968 **A.1 ebXMLError Message DTD**

969 See [ERR] for ebXMLError Message DTD.

970 A.2 ebXML Registry DTD

```

971 <?xml version='1.0' encoding='UTF-8' ?>
972
973 <!--Generated by XML Authority-->
974 <!-- $Header: /jse/jaxr/schema/Registry.dtd,v 1.7 2001/01/10 17:56:28 najmi Exp
975 $ -->
976 <!ENTITY % errorSchema SYSTEM "ebXMLError.dtd">
977
978 %errorSchema;
979
980 <!ENTITY % VersionAttribute " version CDATA #REQUIRED">
981
982 <!ENTITY % ObjectAttributes " description CDATA #IMPLIED
983 ID CDATA #REQUIRED
984 name CDATA #REQUIRED">
985
986 <!ENTITY % ManagedObjectAttributes " %ObjectAttributes;
987 status (SUBMITTED | APPROVED | DEPRECATED )
988 'SUBMITTED'
989 majorVersion CDATA '1'
990 minorVersion CDATA '0'">
991
992 <!ELEMENT ManagedObject EMPTY>
993 <!ATTLIST ManagedObject %ManagedObjectAttributes; >
994 <!ELEMENT ExtrinsicObject EMPTY>
995 <!ATTLIST ExtrinsicObject %ManagedObjectAttributes;
996 contentURN CDATA #IMPLIED
997 mimeType CDATA #IMPLIED
998 objectType (PARTY_AGREEMENT |
999 PARTY_PROFILE |
1000 PROCESS |
1001 ROLE |
1002 SERVICE_INTERFACE |
1003 SOFTWARE_COMPONENT |
1004 TRANSPORT |
1005 UML_MODEL |
1006 UNKNOWN |
1007 XML_SCHEMA ) #REQUIRED
1008 opaque CDATA 'false'
1009 a-dtype NMTOKENS 'opaque boolean' >
1010 <!--
1011 A ClassificationIndex is specified on SCHEMA ExtrinsicObjects to define
1012 an automatic Classification of instance objects of the schema using
1013 the specified classificationNode as parent and a ClassificationNode

```

```
1014 created or selected by the object content as selected by the contentIdentifier
1015 -->
1016 <!ELEMENT ClassificationIndex EMPTY>
1017 <!ATTLIST ClassificationIndex %ObjectAttributes;
1018         classificationNode CDATA #REQUIRED
1019         contentIdentifier CDATA #REQUIRED >
1020 <!-- ClassificationIndexList contains new ClassificationIndexes -->
1021 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
1022
1023 <!ENTITY % IntrinsicObjectAttributes " %ManagedObjectAttributes;">
1024
1025 <!ELEMENT IntrinsicObject EMPTY>
1026 <!ATTLIST IntrinsicObject %ManagedObjectAttributes; >
1027 <!-- Leaf classes that reflect the concrete classes in RIM -->
1028 <!ELEMENT ManagedObjectList (Association | Classification |
1029 ClassificationNode | ExternalLink | Organization | ExtrinsicObject)*>
1030
1031 <!-- Reference to an Object via its URN specified by it ID attribute -->
1032 <!ELEMENT ObjectRef EMPTY>
1033 <!ATTLIST ObjectRef uuid CDATA #REQUIRED >
1034 <!ELEMENT ObjectRefList (ObjectRef)*>
1035
1036 <!--
1037 An ExternalLink specifies a link from a ManagedObject and an external URI
1038
1039 The sourceObjectRef is ref to the ManagedObject
1040
1041 The sourceObjectRef is optional when Association is defined as part of
1042 a SubmittedObject.
1043 -->
1044 <!ELEMENT ExternalLink EMPTY>
1045 <!ATTLIST ExternalLink %IntrinsicObjectAttributes;
1046         sourceObjectRef CDATA #IMPLIED
1047         uri CDATA #IMPLIED >
1048 <!-- ExternalLinkList contains new ExternalLinks or refs to pre-existing
1049 ExternalLinks -->
1050 <!ELEMENT ExternalLinkList (ExternalLink | ObjectRef)*>
1051
1052 <!--
1053 An Association specifies references to two previously submitted
1054 managed objects.
1055
1056 The sourceObjectRef is ref to the sourceObject in association
1057 The targetObjectRef is ref to the targetObject in association
1058
```

1059 The sourceObjectRef is optional when Association is defined part of
 1060 a SubmittedObject.

1061 -->
 1062 <!ELEMENT Association EMPTY>
 1063 <!ATTLIST Association %IntrinsicObjectAttributes;
 1064 fromLabel CDATA #IMPLIED
 1065 toLabel CDATA #IMPLIED
 1066 associationType (CLASSIFIED_BY |
 1067 CONTAINED_BY |
 1068 CONTAINS |
 1069 EXTENDS |
 1070 IMPLEMENTS |
 1071 INSTANCE_OF |
 1072 RELATED_TO |
 1073 SUPERSEDED_BY |
 1074 SUPERSEDES |
 1075 USED_BY |
 1076 USES) #FIXED 'RELATED_TO'
 1077 bidirection CDATA 'false'
 1078 sourceObjectRef CDATA #REQUIRED
 1079 targetObjectRef CDATA #REQUIRED
 1080 a-dtype NMTOKENS 'bidirection boolean' >
 1081 <!ELEMENT AssociationList (Association)*>
 1082
 1083 <!--
 1084 A Classification specifies references to two previously submitted
 1085 managed objects.
 1086
 1087 The sourceObjectRef is ref to the sourceObject in Classification
 1088 The targetObjectRef is ref to the targetObject in Classification
 1089
 1090 The sourceObjectRef is optional when Classification is defined as part of
 1091 a SubmittedObject.
 1092 -->
 1093 <!ELEMENT Classification EMPTY>
 1094 <!ATTLIST Classification %IntrinsicObjectAttributes;
 1095 sourceObjectRef CDATA #REQUIRED
 1096 targetObjectRef CDATA #REQUIRED >
 1097 <!ELEMENT ClassificationList (Classification)*>
 1098
 1099 <!ELEMENT Package EMPTY>
 1100 <!ATTLIST Package %IntrinsicObjectAttributes; >
 1101 <!-- PackageList contains new Packages or refs to pre-existing Packages -->
 1102 <!ELEMENT PackageList (Package | ObjectRef)*>
 1103

```
1104 <!ENTITY % TelephoneNumberAttributes " areaCode CDATA #REQUIRED
1105         contryCode CDATA #REQUIRED
1106         extension CDATA #IMPLIED
1107         number CDATA #REQUIRED
1108         url CDATA #IMPLIED">
1109
1110 <!ELEMENT TelephoneNumber EMPTY>
1111 <!ATTLIST TelephoneNumber %TelephoneNumberAttributes; >
1112 <!ELEMENT FaxNumber EMPTY>
1113 <!ATTLIST FaxNumber %TelephoneNumberAttributes; >
1114 <!ELEMENT MobileTelephoneNumber EMPTY>
1115 <!ATTLIST MobileTelephoneNumber %TelephoneNumberAttributes; >
1116 <!-- PostalAddress -->
1117 <!ELEMENT PostalAddress EMPTY>
1118 <!ATTLIST PostalAddress city CDATA #REQUIRED
1119         country CDATA #REQUIRED
1120         postalCode CDATA #REQUIRED
1121         state CDATA #REQUIRED
1122         street CDATA #REQUIRED >
1123 <!-- PersonName -->
1124 <!ELEMENT PersonName EMPTY>
1125 <!ATTLIST PersonName firstName CDATA #REQUIRED
1126         middleName CDATA #REQUIRED
1127         lastName CDATA #REQUIRED >
1128 <!-- Contact -->
1129 <!ELEMENT Contact (PostalAddress , PersonName , FaxNumber? ,
1130 TelephoneNumber , MobileTelephoneNumber? )>
1131 <!ATTLIST Contact email CDATA #REQUIRED >
1132 <!-- Organization -->
1133 <!ELEMENT Organization (PostalAddress , Contact , FaxNumber? ,
1134 TelephoneNumber )>
1135 <!ATTLIST Organization %IntrinsicObjectAttributes;
1136         parent CDATA #IMPLIED >
1137 <!--
1138 ClassificationNode is used to submit a Classification tree to the Registry.
1139 Note that this is a recursive schema definition.
1140
1141 The parent attribute of a node in tree is implied by the enclosing
1142 ClassificationNode
1143 The children nodes of a node are implied by enclosing immediate child elements
1144 of type ClassificationNode.
1145 -->
1146 <!ELEMENT ClassificationNode EMPTY>
1147 <!ATTLIST ClassificationNode %IntrinsicObjectAttributes;>
1148
```

```
1149 <!--
1150 parent is the URN to the parent node. parent is optional if ClassificationNode is
1151 enclosed
1152 in a parent ClassificationNode or if it is a root ClassificationNode
1153 -->
1154 <!ATTLIST ClassificationNode parent          CDATA #IMPLIED>
1155
1156 <!ELEMENT ClassificationNodeList (ClassificationNode)*>
1157
1158 <!--
1159 End information model mapping.
1160
1161 Begin Registry Services Interface
1162 -->
1163 <!ELEMENT RequestAcceptedResponse EMPTY>
1164 <!ATTLIST RequestAcceptedResponse %VersionAttribute;
1165          xml:lang          NMTOKEN #REQUIRED
1166          interfacedId      CDATA #REQUIRED
1167          requestMessage    CDATA #REQUIRED
1168          actionId         CDATA #REQUIRED >
1169 <!--
1170 The SubmittedObject provides meta data for submitted object
1171 Note object being submitted is in a separate document that is not
1172 in this DTD.
1173 -->
1174 <!ELEMENT SubmitObjectsRequest (SubmittedObject+ )>
1175 <!ATTLIST SubmitObjectsRequest %VersionAttribute; >
1176 <!--
1177 The ExtrinsicObject provides meta data about the object being submitted
1178 ClassificationList can be optionally specified to define Classifications
1179 for the SubmittedObject
1180
1181 AssociationList can be optionally specified to define Associations
1182 for the SubmittedObject
1183
1184 The ExternalLinkList provides zero or more external objects related to
1185 the object being submitted.
1186 -->
1187 <!ELEMENT SubmittedObject (ExtrinsicObject? , ClassificationIndexList? ,
1188 ClassificationList? , AssociationList? , ExternalLinkList? , PackageList? )>
1189
1190 <!--
1191 The ObjectRefList is the list of
1192 refs to the managed objects being approved.
1193 -->
```

```

1194 <!ELEMENT ApproveObjectsRequest (ObjectRefList )>
1195 <!ATTLIST ApproveObjectsRequest %VersionAttribute; >
1196 <!--
1197 The ObjectRefList is the list of
1198 refs to the managed objects being deprecated.
1199 -->
1200 <!ELEMENT DeprecateObjectsRequest (ObjectRefList )>
1201 <!ATTLIST DeprecateObjectsRequest %VersionAttribute; >
1202 <!--
1203 The ObjectRefList is the list of
1204 refs to the managed objects being removed
1205 -->
1206 <!ELEMENT RemoveObjectsRequest (ObjectRefList )>
1207 <!ATTLIST RemoveObjectsRequest %VersionAttribute; >
1208 <!ELEMENT GetRootClassificationNodesRequest EMPTY>
1209 <!ATTLIST GetRootClassificationNodesRequest %VersionAttribute;>
1210
1211 <!--
1212 The namePattern follows SQL-92 syntax for the pattern specified in
1213 LIKE clause. It allows for selecting only those root nodes that match
1214 the namePattern. The default value of '*' matches all root nodes.
1215 -->
1216 <!ATTLIST GetRootClassificationNodesRequest namePattern CDATA "*">
1217
1218 <!--
1219 The response includes a ClassificationNodeList which has zero or more
1220 ClassificationNodes
1221 -->
1222 <!ELEMENT GetRootClassificationNodesResponse (ClassificationNodeList |
1223 ebXMLError )>
1224 <!ATTLIST GetRootClassificationNodesResponse %VersionAttribute; >
1225 <!--
1226 Get the classification tree under the ClassificationNode specified parentRef.
1227
1228 If depth is 1 just fetch immediate child
1229 nodes, otherwise fetch the descendant tree upto the specified depth level.
1230 If depth is 0 that implies fetch entire sub-tree
1231 -->
1232 <!ELEMENT GetClassificationTreeRequest EMPTY>
1233 <!ATTLIST GetClassificationTreeRequest %VersionAttribute;
1234             parent CDATA #REQUIRED
1235             depth CDATA '1' >
1236 <!--
1237 The response includes a ClassificationNodeList which includes only
1238 immediate ClassificationNode children nodes if depth attribute in

```

1239 GetClassificationTreeRequest was 1, otherwise the decendent nodes
1240 upto specified depth level are returned.

1241 -->
1242 <!ELEMENT GetClassificationTreeResponse (ClassificationNodeList |
1243 ebXMLError)>
1244 <!ATTLIST GetClassificationTreeResponse %VersionAttribute; >
1245 <!--

1246 Get refs to all managed objects that are classified by all the
1247 ClassificationNodes specified by ObjectRefList.
1248 Note this is an implicit logical AND operation
1249 -->
1250 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
1251 <!--

1252 objectType attribute can specify the type of objects that the registry
1253 client is interested in, that is classified by this ClassificationNode.
1254 It is a String that matches a choice in the type attribute of ExtrinsicObject.
1255 The default value of '*' implies that client is interested in all types
1256 of managed objects that are classified by the specified ClassificationNode.
1257 -->
1258 <!--

1259 The response includes a ManagedObjectList which has zero or more
1260 ManagedObjects that are classified by the ClassificationNodes
1261 specified in the ObjectRefList in GetClassifiedObjectsRequest.
1262 -->
1263 <!ELEMENT GetClassifiedObjectsResponse (ManagedObjectList | ebXMLError
1264)>
1265 <!ATTLIST GetClassifiedObjectsResponse %VersionAttribute; >
1266 <!--

1267 An Ad hoc query request specifies a query string as defined by [RS] in the
1268 queryString attribute
1269 -->
1270 <!ELEMENT AdhocQueryRequest EMPTY>
1271 <!ATTLIST AdhocQueryRequest %VersionAttribute;
1272 queryString CDATA #REQUIRED >
1273 <!--

1274 The response includes a ManagedObjectList which has zero or more
1275 ManagedObjects that match the query specified in AdhocQueryRequest.
1276 -->
1277 <!ELEMENT AdhocQueryResponse (ManagedObjectList | ebXMLError)>
1278 <!ATTLIST AdhocQueryResponse %VersionAttribute; >
1279 <!--

1280 Gets the actual content (not metadata) specified by the ObjectRefList
1281 -->
1282 <!ELEMENT GetContentRequest (ObjectRefList)>

```
1284 <!ATTLIST GetContentRequest %VersionAttribute; >
1285 <!--
1286 The GetObjectsResponse will have no sub-elements if there were no errors.
1287 The actual contents will be in the other payloads of the message.
1288 If any errors were encountered the message will contain the ebXMLError and
1289 the content payloads will be empty.
1290 -->
1291 <!ELEMENT GetContentResponse (ebXMLError? )>
1292 <!ATTLIST GetContentResponse %VersionAttribute; >
1293 <!--
1294 The contrived root node
1295 -->
1296 <!ELEMENT RootElement (RequestAcceptedResponse | ebXMLError |
1297 SubmitObjectsRequest | ApproveObjectsRequest | DeprecateObjectsRequest |
1298 RemoveObjectsRequest | GetRootClassificationNodesRequest |
1299 GetRootClassificationNodesResponse | GetClassificationTreeRequest |
1300 GetClassificationTreeResponse | GetClassifiedObjectsRequest |
1301 GetClassifiedObjectsResponse | AdhocQueryRequest | AdhocQueryResponse |
1302 GetContentRequest | GetContentResponse )>
```

1303 **Appendix B Interpretation of UML Diagrams**

1304 This section describes in *abstract terms* the conventions used to define ebXML
1305 business process description in UML.

1306 **B.1 UML Class Diagram**

1307 A UML class diagram is used to describe the Service Interfaces (as defined by
1308 [CPA]) required to implement an ebXML Registry Services and clients. See
1309 Figure 1 on page 12 for an example. The UML class diagram contains:

1310

- 1311 1. A collection of UML interfaces where each interface represents a Service
1312 Interface for a Registry service.
- 1313 2. Tabular description of methods on each interface where each method
1314 represents an Action (as defined by [CPA]) within the Service Interface
1315 representing the UML interface.
- 1316 3. Each method within a UML interface specifies one or more parameters,
1317 where the type of each method argument represents the ebXML message
1318 type that is exchanged as part of the Action corresponding to the method.
1319 Multiple arguments imply multiple payload documents within the body of
1320 the corresponding ebXML message.

1321 **B.2 UML Sequence Diagram**

1322 A UML sequence diagram is used to specify the business protocol representing
 1323 the interactions between the UML interfaces for a Registry specific ebXML
 1324 business process. A UML sequence diagram provides the necessary information
 1325 to determine the sequencing of messages, request to response association as
 1326 well as request to error response association as described by [CPA].

1327 Each sequence diagram shows the sequence for a specific conversation protocol
 1328 as method calls from the requestor to the responder. Method invocation may be
 1329 synchronous or asynchronous based on the UML notation used on the arrow-
 1330 head for the link. A half arrow-head represents asynchronous communication. A
 1331 full arrow-head represents synchronous communication.

1332 Each method invocation may be followed by a response method invocation from
 1333 the responder to the requestor to indicate the ResponseName for the previous
 1334 Request. Possible error response is indicated by a conditional response method
 1335 invocation from the responder to the requestor. See Figure 3 on page 19 for an
 1336 example.

1337 **Appendix C BNF for Query Syntax Grammar**

1338 The following BNF defines the grammar for the registry query syntax. This
 1339 grammer is a proper sub-set of SQL-92 as defined by [SQL].

```

1340 /*****
1341  * The Registry Query (Subset of SQL-92) grammar starts here
1342  *****/
1343
1344 RegistryQuery = SQLSelect ["," ]
1345
1346 SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
1347
1348 SQLSelectCols = ( "ALL" | "DISTINCT" ) * [ ID ]
1349
1350 SQLTableList = SQLTableRef ( "," SQLTableRef ) *
1351
1352 SQLTableRef = ID [ ID ]
1353
1354 SQLWhere = "WHERE" SQLOrExpr
1355
1356 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
1357
1358 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *
1359
1360
    
```

```
1361 SQLNotExpr = [ "NOT" ] SQLCompareExpr
1362
1363 SQLCompareExpr =
1364     (SQLColRef "IS") SQLIsClause
1365     | SQLSumExpr [ SQLCompareExprRight ]
1366
1367
1368 SQLCompareExprRight =
1369     SQLLikeClause
1370     | SQLInClause
1371     | SQLCompareOp SQLSumExpr
1372
1373 SQLCompareOp =
1374     "="
1375     | "!="
1376     | ">"
1377     | ">="
1378     | "<"
1379     | "<="
1380
1381 SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
1382
1383 SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
1384
1385 SQLLValueElement = "NULL" | SQLSumExpr
1386
1387 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
1388
1389 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
1390
1391 SQLPattern = STRING_LITERAL
1392
1393 SQLLiteral =
1394     STRING_LITERAL
1395     | INTEGER_LITERAL
1396     | FLOATING_POINT_LITERAL
1397
1398 SQLColRef = SQLLvalue
1399
1400 SQLLvalue = SQLLvalueTerm
1401
1402 SQLLvalueTerm = ID ( "." ID )*
1403
1404 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr )*
```

```

1406 SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*
1407
1408 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
1409
1410 SQLTerm = "(" SQLOrExpr ")"
1411 | SQLColRef
1412 | SQLLiteral
1413
1414 INTEGER_LITERAL = ([ "0"-"9" ])+
1415
1416 FLOATING_POINT_LITERAL =
1417     ([ "0"-"9" ])+ "." ([ "0"-"9" ])+ (EXPONENT)?
1418 | "." ([ "0"-"9" ])+ (EXPONENT)?
1419 | ([ "0"-"9" ])+ EXPONENT
1420 | ([ "0"-"9" ])+ (EXPONENT)?
1421
1422 EXPONENT = [ "e", "E" ] ([ "+", "-" ])? ([ "0"-"9" ])+
1423
1424 STRING_LITERAL: "'" (~["'"])* ( "'" (~["'"])* ) * "'"
1425
1426 ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
1427 LETTER = [ "A"-"Z", "a"-"z" ]
1428 DIGIT = [ "0"-"9" ]

```

1429 **Appendix D Security Implementation Guideline**

1430 This section provides a suggested blueprint for how security processing may be
 1431 implemented in the Registry. It is meant to be illustrative not prescriptive.
 1432 Registries may choose to have different implementations as long as they support
 1433 the default security roles and authorization rules described in this document.

1434 **D.1 Authentication**

- 1435 1. As soon as a message is received, the first work is the authentication. A
 1436 principal object is created.
- 1437 2. If the message is signed, it is verified (including the validity of the certificate)
 1438 and the DN of the certificate becomes the identity of the principal. Then the
 1439 Registry is searched for the principal and if found, the roles and groups are
 1440 filled in.
- 1441 3. If the message is not signed, an empty principal is created with the role
 1442 RegistryGuest. This step is for symmetry and to decouple the rest of the
 1443 processing.

1444 4. Then the message is processed for the command and the objects it will act on

1445 **D.2 Authorization**

1446 For every object, the access controller will iterate through all the
1447 AccessControlPolicy objects with the object and see if there is a chain through
1448 the permission objects to verify that the requested method is permitted for the
1449 Principal. If any of the permission objects which the object is associated with has
1450 a common role, or identity, or group with the principal, the action is permitted.

1451 **D.3 Registry Bootstrap**

1452 When a Registry is newly created, a default Principal object should be created
1453 with the identity of the Registry Admin's certificate DN with a role RegistryAdmin.
1454 This way, any message signed by the Registry Admin will get all the privileges.

1455 When a Registry is newly created, a singleton instance of AccessControlPolicy is
1456 created as the default AccessControlPolicy. This includes the creation of the
1457 necessary Permission instances as well as the Privileges and Privilege attributes.

1458 **D.4 Content Submission – Client Responsibility**

1459 The Registry client has to sign the contents before submission – otherwise the
1460 content will be rejected.

1461 **D.5 Content Submission – Registry Responsibility**

- 1462 1. Like any other request, the client will be first authenticated. In this case, the
1463 Principal object will get the DN from the certificate.
- 1464 2. As per the request in the message, the ManagedObject will be created.
- 1465 3. The ManagedObject is assigned the singleton default AccessControlPolicy.
- 1466 4. If a principal with the identity of the SO is not available, an identity object with
1467 the SO's DN is created
- 1468 5. A principal with this identity is created

1469 **D.6 Content Delete/Deprecate – Client Responsibility**

1470 The Registry client has to sign the payload (not entire message) before
1471 submission, for authentication purposes; otherwise, the request will be
1472 rejected

1473 **D.7 Content Delete/Deprecate – Registry Responsibility**

- 1474 1. Like any other request, the client will be first authenticated. In this case, the
 1475 Principal object will get the DN from the certificate. As there will be a principal
 1476 with this identity in the Registry, the Principal object will get all the roles from
 1477 that object
- 1478 2. As per the request in the message (delete or deprecate), the appropriate
 1479 method in the Object will be accessed.
- 1480 3. The access controller performs the authorization by iterating through the
 1481 Permission objects associated with this object via the singleton default
 1482 AccessControlPolicy.
- 1483 4. If authorization succeeds then the action will be permitted. Otherwise an error
 1484 response is sent back with a suitable AuthorizationException error message.

1485 **Appendix E Terminology Mapping**

1486 While every attempt has been made to use the same terminology used in other
 1487 works there are some terminology differences.

1488 The following table shows the terminology mapping between this specification
 1489 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	

1490 **Table 1: Terminology Mapping Table**

1491 **10 References**

- 1492 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 1493 [TA] ebXML Technical Architecture
- 1494 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf
- 1495 [OAS] OASIS Information Model
- 1496 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 1497 [ISO] ISO 11179 Information Model
- 1498 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 1499
- 1500 [BDM] Registry and Repository: Business Domain Model
- 1501 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 1502 [RIM] ebXML Registry Information Model
- 1503 http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
- 1504 [BPM] ebXML Business Process Metamodel Specification Schema
- 1505 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 1506 [CPA] Trading-Partner Specification
- 1507 http://www.ebxml.org/project_teams/trade_partner/private/
- 1508 [CTB] Context table informal document from Core Components
- 1509 [MS] ebXML Messaging Service Specification, Version 0.21
- 1510 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
- 1511 [ERR] ebXML TRP Error Handling Specification
- 1512 http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf
- 1513 [SEC] ebXML Security Specification
- 1514 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 1515 [XPT] XML Path Language (XPath) Version 1.0
- 1516 <http://www.w3.org/TR/xpath>
- 1517 [SQL] Structured Query Language (FIPS PUB 127-2)
- 1518 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>

1519 **11 Disclaimer**

1520 The views and specification expressed in this document are those of the authors
1521 and are not necessarily those of their employers. The authors and their
1522 employers specifically disclaim responsibility for any problems arising from
1523 correct or incorrect implementation or use of this design.

1524

1524 12 Contact Information**1525 Team Leader**

1526 Name: Scott Nieman
1527 Company: Norstan Consulting
1528 Street: 5101 Shady Oak Road
1529 City, State, Postal Code: Minnetonka, MN 55343
1530 Country: USA
1531 Phone: 952.352.5889
1532 Email: Scott.Nieman@Norstan

1533

1534 Vice Team Lead

1535 Name: Yutaka Yoshida
1536 Company: Sun Microsystems
1537 Street: 901 San Antonio Road, MS UMPK17-102
1538 City, State, Postal Code: Palo Alto, CA 94303
1539 Country: USA
1540 Phone: 650.786.5488
1541 Email: Yutaka.Yoshida@eng.sun.com

1542

1543 Editor

1544 Name: Farrukh S. Najmi
1545 Company: Sun Microsystems
1546 Street: 1 Network Dr., MS BUR02-302
1547 City, State, Postal Code: Burlington, MA, 01803-0902
1548 Country: USA
1549 Phone: 781.442.0703
1550 Email: najmi@east.sun.com

1551

1552

1552 **Copyright Statement**

1553 Copyright © ebXML 2000. All Rights Reserved.

1554

1555 This document and translations of it may be copied and furnished to others, and
1556 derivative works that comment on or otherwise explain it or assist in its
1557 implementation may be prepared, copied, published and distributed, in whole or
1558 in part, without restriction of any kind, provided that the above copyright notice
1559 and this paragraph are included on all such copies and derivative works.
1560 However, this document itself may not be modified in any way, such as by
1561 removing the copyright notice or references to the Internet Society or other
1562 Internet organizations, except as needed for the purpose of developing Internet
1563 standards in which case the procedures for copyrights defined in the Internet
1564 Standards process must be followed, or as required to translate it into languages
1565 other than English.

1566

1567 The limited permissions granted above are perpetual and will not be revoked by
1568 ebXML or its successors or assigns.

1569

1570 This document and the information contained herein is provided on an
1571 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR
1572 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE
1573 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
1574 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
1575 PARTICULAR PURPOSE.