



---

Creating A Single Global Electronic Market

1  
2

# **OASIS/ebXML Registry Services Specification v1.01 DRAFT**

## **OASIS/ebXML Registry Technical Committee**

27 October 2001

## 2 **1 Status of this Document**

3

4 Distribution of this document is unlimited.

5

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

7

### 8 ***This version:***

9 <http://www.oasis-open.org/committees/regrep/document/rsV1-01.pdf>

10

### 11 ***Latest version:***

12 <http://www.oasis-open.org/committees/regrep/documents/rsV1-01.pdf>

13

## 13 **2 OASIS/ebXML Registry Technical Committee**

14 This document, in its current form, has been approved by the OASIS/ebXML Registry  
15 Technical Committee as DRAFT Specification of the TC. At the time of this approval the  
16 following were members of the OASIS/ebXML Registry Technical Committee.

17

18 Nagwa Abdelghfour, Sun Microsystems  
19 Nicholas Berry, Boeing  
20 Kathryn Breining, Boeing  
21 Lisa Carnahan, US NIST (TC Chair)  
22 Dan Chang, IBM  
23 Joseph M. Chiusano, LMI  
24 Joe Dalman, Tie Commerce  
25 Suresh Damodaran, Sterling Commerce  
26 Vadim Draluk, BEA  
27 John Evdemon, Vitria Technologies  
28 Anne Fischer, Drummond Group  
29 Sally Fuger, AIAG  
30 Len Gallagher, NIST  
31 Michael Joya, XMLGlobal  
32 Una Kearns, Documentum  
33 Kyu-Chul Lee, Chungnam National University  
34 Megan MacMillan, Gartner Solista  
35 Norbert Mikula, DataChannel  
36 Joel Munter, Intel  
37 Farrukh Najmi, Sun Microsystems  
38 Joel Neu, Vitria Technologies  
39 Sanjay Patil, IONA  
40 Neal Smith, Chevron  
41 Nikola Stojanovic, Encoda Systems Inc.  
42 David Webber, XMLGlobal  
43 Prasad Yendluri, webmethods  
44 Yutaka Yoshida, Sun Microsystems

45

45	<b>Table of Contents</b>		
46	<b>1</b>	<b>Status of this Document</b> .....	<b>2</b>
47	<b>2</b>	<b>OASIS/ebXML Registry Technical Committee</b> .....	<b>3</b>
48		<b>Table of Contents</b> .....	<b>4</b>
49		<b>Table of Tables</b> .....	<b>8</b>
50	<b>3</b>	<b>Introduction</b> .....	<b>9</b>
51	3.1	Summary of Contents of Document .....	9
52	3.2	General Conventions .....	9
53	3.3	Audience .....	9
54	3.4	Related Documents .....	9
55	<b>4</b>	<b>Design Objectives</b> .....	<b>10</b>
56	4.1	Goals .....	10
57	4.2	Caveats and Assumptions .....	10
58	<b>5</b>	<b>System Overview</b> .....	<b>10</b>
59	5.1	What The ebXML Registry Does .....	10
60	5.2	How The ebXML Registry Works.....	10
61		5.2.1 Schema Documents Are Submitted .....	11
62		5.2.2 Business Process Documents Are Submitted .....	11
63		5.2.3 Seller's Collaboration Protocol Profile Is Submitted .....	11
64		5.2.4 Buyer Discovers The Seller .....	11
65		5.2.5 CPA Is Established .....	11
66	5.3	Where the Registry Services May Be Implemented.....	12
67	5.4	Implementation Conformance .....	12
68		5.4.1 Conformance as an ebXML Registry.....	12
69		5.4.2 Conformance as an ebXML Registry Client.....	12
70	<b>6</b>	<b>Registry Service and Interfaces</b> .....	<b>13</b>
71	6.1	Registry Service Description .....	13
72	6.2	Abstract Registry Service.....	14
73	6.3	Concrete Registry Services .....	15
74		6.3.1 Using SOAP .....	15
75		6.3.2 Using ebXML Message Service.....	15
76		6.3.2.1 Service and Action Elements .....	15
77		6.3.2.2 Synchronous and Asynchronous Responses .....	16
78		6.3.2.3 ebXML Registry Collaboration Profiles and Agreements.....	16
79	6.4	Interoperability Requirements .....	17
80	6.5	LifeCycleManager Interface.....	17
81	6.6	QueryManager Interface .....	17
82	6.7	Registry Clients.....	18
83		6.7.1 Registry Architecture .....	18
84		6.7.2 Client To Registry Communication Bootstrapping .....	20

85	6.7.3	RegistryClient Interface .....	20
86	6.7.4	Registry Response Class Hierarchy .....	20
87	<b>7</b>	<b>Object Management Service .....</b>	<b>21</b>
88	7.1	Life Cycle of a Repository Item .....	21
89	7.2	RegistryObject Attributes .....	22
90	7.3	The Submit Objects Protocol.....	22
91	7.3.1	Universally Unique ID Generation.....	23
92	7.3.2	ID Attribute And Object References .....	24
93	7.3.3	Sample SubmitObjectsRequest .....	24
94	7.4	The Add Slots Protocol.....	26
95	7.5	The Remove Slots Protocol.....	27
96	7.6	The Approve Objects Protocol.....	28
97	7.7	The Deprecate Objects Protocol.....	28
98	7.8	The Remove Objects Protocol.....	29
99	7.8.1	Deletion Scope DeleteRepositoryItemOnly.....	29
100	7.8.2	Deletion Scope DeleteAll .....	30
101	<b>8</b>	<b>Object Query Management Service.....</b>	<b>30</b>
102	8.1	Browse and Drill Down Query Support .....	31
103	8.1.1	Get Root Classification Nodes Request .....	31
104	8.1.2	Get Classification Tree Request .....	32
105	8.1.3	Get Classified Objects Request .....	32
106	8.1.3.1	Get Classified Objects Request Example .....	33
107	8.2	In the event of success, the registry sends a GetClassifiedObjectsResponse	
108		with a status of "success" back to the client. In the event of failure, the registry sends a	
109		GetClassifiedObjectsResponse with a status of "failure" back to the client. Filter Query	
110		Support	34
111	8.2	Filter Query Support.....	35
112	8.2.1	FilterQuery.....	37
113	8.2.2	RegistryEntryQuery.....	39
114	8.2.3	AuditableEventQuery.....	45
115	8.2.4	ClassificationNodeQuery.....	48
116	8.2.5	RegistryPackageQuery .....	51
117	8.2.6	OrganizationQuery .....	53
118	8.2.7	ReturnRegistryEntry.....	56
119	8.2.8	ReturnRepositoryItem.....	60
120	8.2.9	Registry Filters .....	64
121	8.2.10	XML Clause Constraint Representation.....	68
122	8.3	SQL Query Support .....	72
123	8.3.1	SQL Query Syntax Binding To [ebRIM].....	72
124	8.3.1.1	Interface and Class Binding .....	72
125	8.3.1.2	Accessor Method To Attribute Binding .....	73
126	8.3.1.3	Primitive Attributes Binding .....	73
127	8.3.1.4	Reference Attribute Binding .....	73
128	8.3.1.5	Complex Attribute Binding .....	73
129	8.3.1.6	Collection Attribute Binding .....	74

130	8.3.2	Semantic Constraints On Query Syntax.....	74
131	8.3.3	SQL Query Results .....	74
132	8.3.4	Simple Metadata Based Queries .....	75
133	8.3.5	RegistryEntry Queries .....	75
134	8.3.6	Classification Queries .....	75
135	8.3.6.1	Identifying ClassificationNodes .....	75
136	8.3.6.2	Getting Root Classification Nodes.....	75
137	8.3.6.3	Getting Children of Specified ClassificationNode .....	76
138	8.3.6.4	Getting Objects Classified By a ClassificationNode .....	76
139	8.3.6.5	Getting ClassificationNodes That Classify an Object... ..	76
140	8.3.7	Association Queries .....	76
141	8.3.7.1	Getting All Association With Specified Object As Its Source	
142		76	
143	8.3.7.2	Getting All Association With Specified Object As Its Target	
144		77	
145	8.3.7.3	Getting Associated Objects Based On Association Attributes	
146		77	
147	8.3.7.4	Complex Association Queries .....	77
148	8.3.8	Package Queries .....	77
149	8.3.8.1	Complex Package Queries .....	77
150	8.3.9	ExternalLink Queries .....	77
151	8.3.9.1	Complex ExternalLink Queries.....	78
152	8.3.10	Audit Trail Queries .....	78
153	8.4	Ad Hoc Query Request/Response .....	78
154	8.5	Content Retrieval.....	79
155	8.5.1	Identification Of Content Payloads .....	79
156	8.5.2	GetContentResponse Message Structure .....	79
157	8.6	Query And Retrieval: Typical Sequence.....	80
158	<b>9</b>	<b>Registry Security.....</b>	<b>81</b>
159	9.1	Integrity of Registry Content .....	82
160	9.1.1	Message Payload Signature.....	82
161	9.2	Authentication .....	82
162	9.2.1	Message Header Signature .....	82
163	9.3	Confidentiality .....	83
164	9.3.1	On-the-wire Message Confidentiality .....	83
165	9.3.2	Confidentiality of Registry Content .....	83
166	9.4	Authorization .....	83
167	9.4.1	Pre-defined Roles For Registry Users .....	83
168	9.4.2	Default Access Control Policies .....	83
169	<b>Appendix A</b>	<b>Web Service Architecture .....</b>	<b>84</b>
170	A.1	WSDL Terminology Primer .....	84
171	A.2	Registry Service Abstract Specification.....	85
172	A.3	Registry Service SOAP Binding .....	86
173	<b>Appendix B</b>	<b>ebXML Registry DTD Definition .....</b>	<b>89</b>

174	<b>Appendix C</b>	<b>Interpretation of UML Diagrams</b> .....	<b>100</b>
175	C.1	UML Class Diagram.....	100
176	C.2	UML Sequence Diagram.....	101
177	<b>Appendix D</b>	<b>SQL Query</b> .....	<b>101</b>
178	D.1	SQL Query Syntax Specification .....	101
179	D.2	Non-Normative BNF for Query Syntax Grammar .....	102
180	D.3	Relational Schema For SQL Queries.....	103
181	<b>Appendix E</b>	<b>Non-normative Content Based Ad Hoc Queries</b> .....	<b>110</b>
182	E.1.1	Automatic Classification of XML Content .....	110
183	E.1.2	Index Definition .....	110
184	E.1.3	Example Of Index Definition .....	111
185	E.1.4	Proposed XML Definition .....	111
186	E.1.5	Example of Automatic Classification .....	111
187	<b>Appendix F</b>	<b>Security Implementation Guideline</b> .....	<b>111</b>
188	F.1	Authentication .....	112
189	F.2	Authorization .....	112
190	F.3	Registry Bootstrap.....	112
191	F.4	Content Submission – Client Responsibility .....	112
192	F.5	Content Submission – Registry Responsibility.....	112
193	F.6	Content Delete/Deprecate – Client Responsibility.....	113
194	F.7	Content Delete/Deprecate – Registry Responsibility .....	113
195	<b>Appendix G</b>	<b>Native Language Support (NLS)</b> .....	<b>113</b>
196	G.1	Definitions .....	113
197	G.1.1	Coded Character Set (CCS):.....	113
198	G.1.2	Character Encoding Scheme (CES): .....	113
199	G.1.3	Character Set (charset):.....	114
200	G.2	NLS And Request / Response Messages .....	114
201	G.3	NLS And Storing of RegistryEntry .....	114
202	G.3.1	Character Set of <i>RegistryEntry</i> .....	114
203	G.3.2	Language Information of <i>RegistryEntry</i> .....	114
204	G.4	NLS And Storing of Repository Items .....	115
205	G.4.1	Character Set of Repository Items .....	115
206	G.4.2	Language information of repository item .....	115
207	<b>Appendix H</b>	<b>Terminology Mapping</b> .....	<b>115</b>
208	<b>References</b> .....		<b>117</b>
209	<b>Disclaimer</b> .....		<b>118</b>
210	<b>Contact Information</b> .....		<b>119</b>
211	<b>Copyright Statement</b> .....		<b>120</b>

212

## 213 **Table of Figures**

214	Figure 1: ebXML Registry Service.....	13
215	Figure 2: The Abstract ebXML Registry Service.....	14
216	Figure 3: A Concrete ebXML Registry Service.....	15
217	Figure 4: Registry Architecture Supports Flexible Topologies.....	19
218	Figure 6: Registry Repsonse Class Hierarchy.....	21
219	Figure 8: Life Cycle of a Repository Item.....	22
220	Figure 9: Submit Objects Sequence Diagram.....	23
221	Figure 10: Add Slots Sequence Diagram.....	27
222	Figure 11: Remove Slots Sequence Diagram.....	27
223	Figure 12: Approve Objects Sequence Diagram.....	28
224	Figure 13: Deprecate Objects Sequence Diagram.....	29
225	Figure 14: Remove Objects Sequence Diagram.....	30
226	Figure 15: Get Root Classification Nodes Sequence Diagram.....	31
227	Figure 16: Get Classification Tree Sequence Diagram.....	32
228	Figure 17: A Sample Geography Classification.....	33
229	Figure 18: Get Classified Objects Sequence Diagram.....	33
230	Figure 19: Example ebRIM Binding.....	35
231	Figure 20: The Clause base structure.....	68
232	Figure 21: Submit Ad Hoc Query Sequence Diagram.....	78
233	Figure 22: Typical Query and Retrieval Sequence.....	81

## 234 **Table of Tables**

235	Table 1: Terminology Mapping Table.....	116
-----	---	-----

236

237



## 237 **3 Introduction**

### 238 **3.1 Summary of Contents of Document**

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

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

### 244 **3.2 General Conventions**

245 The following conventions are used throughout this document:

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

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

### 257 **3.3 Audience**

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

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

### 262 **3.4 Related Documents**

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

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

## 269 **4 Design Objectives**

### 270 **4.1 Goals**

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

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

### 276 **4.2 Caveats and Assumptions**

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

280 It is assumed that:

- 281 1. Interoperability requirements dictate that that at least one of the normative  
282 interfaces as referenced in this specification must be supported.
- 283 2. All access to the Registry content is exposed via the interfaces defined for the  
284 Registry Services.
- 285 3. The Registry makes use of a Repository for storing and retrieving persistent  
286 information required by the Registry Services. This is an implementation detail  
287 that will not be discussed further in this specification.

## 288 **5 System Overview**

### 289 **5.1 What The ebXML Registry Does**

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

### 295 **5.2 How The ebXML Registry Works**

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

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

#### 306 **5.2.1 Schema Documents Are Submitted**

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

#### 311 **5.2.2 Business Process Documents Are Submitted**

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

#### 316 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

#### 321 **5.2.4 Buyer Discovers The Seller**

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

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

#### 328 **5.2.5 CPA Is Established**

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

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

### 335 **5.3 Where the Registry Services May Be Implemented**

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

### 338 **5.4 Implementation Conformance**

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

#### 346 **5.4.1 Conformance as an ebXML Registry**

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

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

#### 355 **5.4.2 Conformance as an ebXML Registry Client**

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

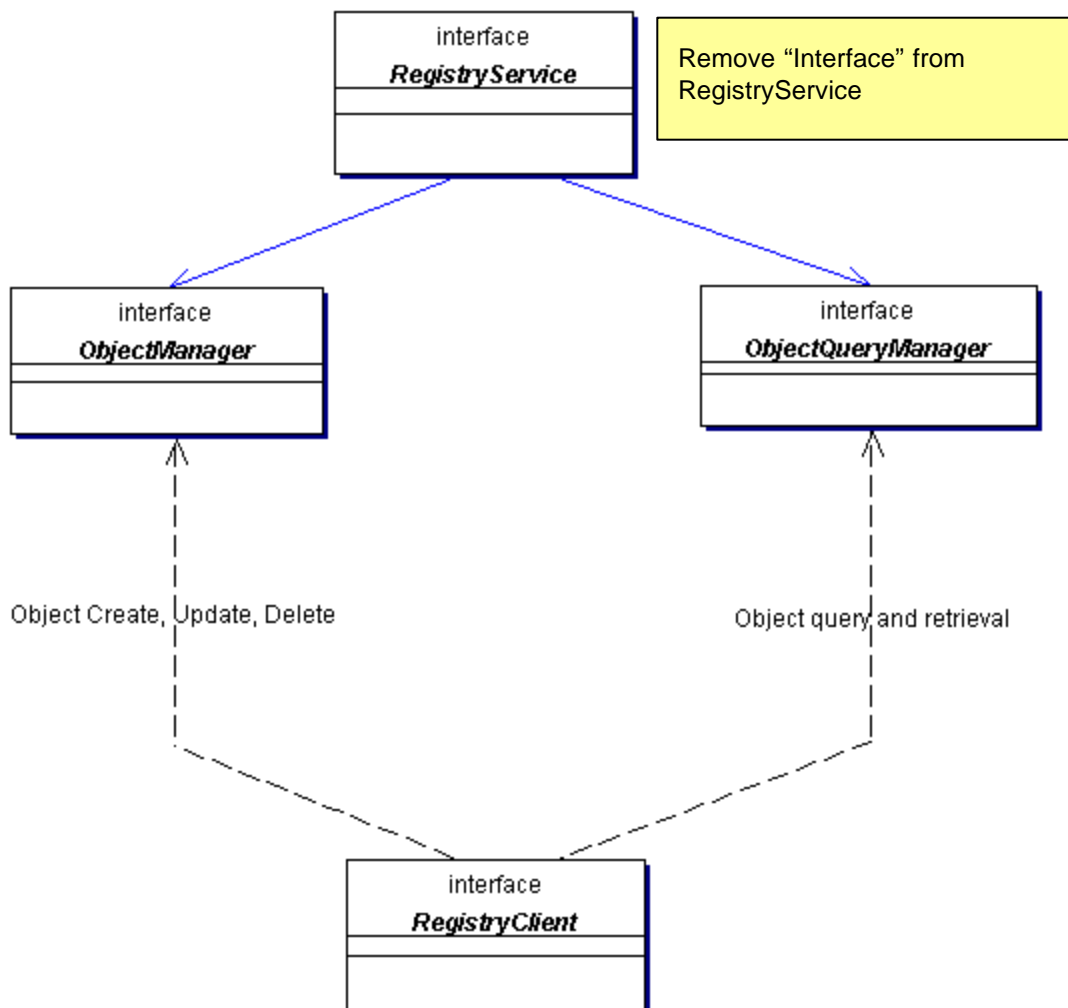
- 358 1. Supports the ebXML CPA and bootstrapping process.
- 359 2. Supports the syntax and the semantics of the Registry Client Interfaces.
- 360 3. Supports the defined ebXML Error Message DTD.
- 361 4. Supports the defined ebXML Registry DTD.

## 362 6 Registry Service and Interfaces

### 363 6.1 Registry Service Description

364 The ebXML Registry Service is comprised of a robust set of interfaces designed to  
 365 fundamentally manage the objects and inquiries associated with the ebXML Registry.  
 366 The two primary interfaces for the Registry Service consist of a Life Cycle Management  
 367 interface that controls the required processes necessary for managing an object within  
 368 the Registry and a Query Management Interface that controls the release of information  
 369 from the Registry. Both of these interfaces are accessed through the use of a Registry  
 370 Client Interface.

371 This specification defines the interfaces exposed by the Registry Service (Sections  
 372 6.5 and 6.6) as well as the interface for the Registry Client (Section 6.7). Figure 1 shows  
 373 the relationship between the interfaces and the Registry Service.



374

375

Figure 1: ebXML Registry Service

## Method Summary of RegistryService

<a href="#">LifeCycleManager</a>	<a href="#">getLifeCycleManager()</a> Returns the LifeCycleManager interface implemented by the Registry service.
<a href="#">QueryManager</a>	<a href="#">getQueryManager()</a> Returns the QueryManager interface implemented by the Registry service.

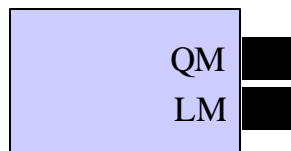
376 This version of the specification does not preclude ebXML Registries from cooperating  
 377 with each other to share information, nor does it preclude owners of ebXML Registries  
 378 from registering their ebXML registries with other registry systems, catalogs, or  
 379 directories.

380 Examples include:

- 381 • an ebXML Registry of Registries that serves as a centralized registration point;
- 382 • cooperative ebXML Registries, where registries register with each other in a  
 383 federation;
- 384 • registration of ebXML Registries with other Registry systems that act as white  
 385 pages or yellow pages. The document [ebXML-UDDI] provides an example of  
 386 ebXML Registries being discovered through a system of emerging white/yellow  
 387 pages known as UDDI.

## 388 6.2 Abstract Registry Service

389 The architecture defines an abstract registry service as shown in Figure 2. The figure  
 390 shows how an abstract ebXML Registry must provide two key functional interfaces  
 391 called `QueryManager`<sup>1</sup> (QM) and `LifeCycleManager`<sup>2</sup> (LM). When mapping to  
 392 WSDL, these interfaces are represented as `port` types within the WSDL description  
 393 in Appendix A.2.



Registry Service

Figure 2: The Abstract ebXML Registry Service

394

395

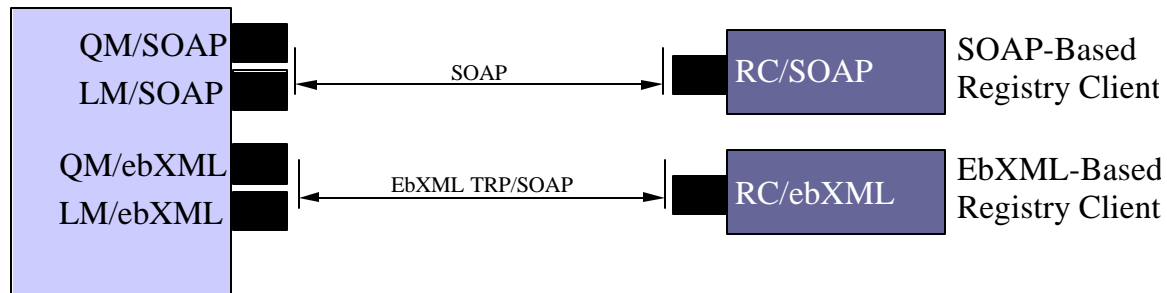
<sup>1</sup> Known as `ObjectQueryManager` in V1.0

<sup>2</sup> Known as `ObjectManager` in V1.0

## 396 6.3 Concrete Registry Services

397 The architecture further defines how concrete implementations of the abstract registry  
 398 may be realized as a web service. This is defined in Appendix A.3 using `binding` and  
 399 `service` definitions within the WSDL description, where the abstract `port` types are  
 400 mapped to `ports` bound to specific protocols.

401



402 Registry Service

403

Figure 3: A Concrete ebXML Registry Service

404 Figure 3 shows a concrete implementation of the abstract ebXML Registry as a web  
 405 service (name RegistryService) on the left side. The RegistryService provides the  
 406 QueryManager and LifeCycleManager interfaces available with multiple protocol  
 407 bindings (SOAP and ebXML TRP). Each interface/protocol combination is defined as a  
 408 port definition in the WSDL in Appendix A.3.

409 Figure 3 also shows two different clients of the ebXML Registry on the right side. The  
 410 top client uses SOAP protocol to access the registry while the lower client uses ebXML  
 411 TRP. Each client uses the appropriate `port` within the RegistryService `service` based  
 412 upon their protocol preference.

### 413 6.3.1 Using SOAP

414 Put any SOAP/HTTP specific packaging information here

### 415 6.3.2 Using ebXML Message Service

#### 416 6.3.2.1 Service and Action Elements

417 When using the ebXML Messaging Services Specification, ebXML Registry Services  
 418 elements correspond to Messaging Services elements as follows:

- 419 • The value of the Service element in the MessageHeader is an ebXML Registry  
 420 Service interface name (e.g., "LifeCycleManager"). The type attribute of the  
 421 Service element should have a value of "ebXMLRegistry".
- 422 • The value of the Action element in the MessageHeader is an ebXML Registry  
 423 Service method name (e.g., "submitObjects").



```
424 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
425 <eb:Action>submitObjects</eb:Action>
```

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

### 429 **6.3.2.2 Synchronous and Asynchronous Responses**

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

- 431 • Asynchronous response
  - 432 ○ MessageHeader only;
  - 433 ○ No registry response element (e.g., AdHocQueryResponse and  
434 GetContentResponse).
- 435 • Synchronous response
  - 436 ○ MessageHeader;
  - 437 ○ Registry response element including
    - 438 ▪ a status attribute (success or failure)
    - 439 ▪ an optional ebXML Error.

### 440 **6.3.2.3 ebXML Registry Collaboration Profiles and Agreements**

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

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

450 The CPA between clients and the Registry should describe the interfaces that the  
451 Registry and the client expose to each other for Registry-specific interactions. The  
452 definition of the Registry CPP template and a Registry Client CPP template are beyond  
453 the scope of this document.



## 454 6.4 Interoperability Requirements

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

## 460 6.5 LifeCycleManager Interface

461  
 462 This is the interface exposed by the Registry Service that implements the Object life  
 463 cycle management functionality of the Registry. Its' methods are invoked by the  
 464 Registry Client. For example, the client may use this interface to submit objects, to  
 465 classify and associate objects and to deprecate and remove objects. For this  
 466 specification the semantic meaning of submit, classify, associate, deprecate and  
 467 remove is found in [ebRIM].

468

### Method Summary of LifeCycleManager

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

## 469 6.6 QueryManager Interface

470

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

475

<b>Method Summary of QueryManager</b>	
<a href="#">RegistryResponse</a>	<b><a href="#">getClassificationTree</a></b> ( <a href="#">GetClassificationTreeRequest</a> req) Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.
<a href="#">RegistryResponse</a>	<b><a href="#">getClassifiedObjects</a></b> ( <a href="#">GetClassifiedObjectsRequest</a> req) Returns a collection of references to RegistryEntries classified under specified ClassificationItem.
<a href="#">RegistryResponse</a>	<b><a href="#">getContent</a></b> ( <a href="#">GetContentRequest</a> req) Returns the content of the specified Repository Item. The response includes all the content specified in the request as additional payloads within the response message.
<a href="#">RegistryResponse</a>	<b><a href="#">getRootClassificationNodes</a></b> ( <a href="#">GetRootClassificationNodesRequest</a> req) Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.
<a href="#">RegistryResponse</a>	<b><a href="#">submitAdhocQuery</a></b> ( <a href="#">AdhocQueryRequest</a> req) Submit an ad hoc query request.

## 476 6.7 Registry Clients

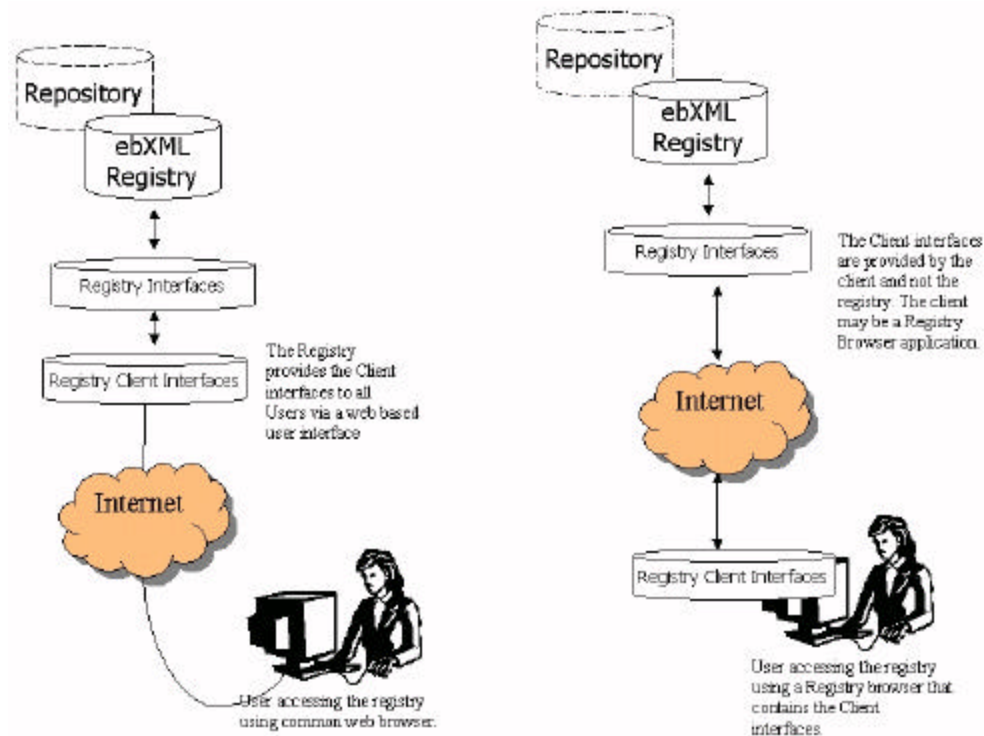
### 477 6.7.1 Registry Architecture

478 The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry  
 479 Clients. The Registry Client interfaces may be local to the registry or local to the user.  
 480 Figure 4 depicts the two possible topologies supported by the registry architecture with  
 481 respect to the Registry and Registry Clients.

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

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

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



497

498

499

Figure 4: Registry Architecture Supports Flexible Topologies

## 500 **6.7.2 Client To Registry Communication Bootstrapping**

501 Each ebXML Registry must provide a WSDL description for its RegistryService as  
 502 defined by Appendix A.3. A client uses the WSDL description to determine the address  
 503 information of the RegistryService in a protocol specific manner. For example the  
 504 SOAP/HTTP based ports of the RegistryService may be accessed via a URL specified  
 505 in the WSDL for the registry.

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

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

## 512 **6.7.3 RegistryClient Interface**

513  
 514 This is the principal interface implemented by a Registry client. The client provides this  
 515 interface when creating a connection to the Registry. It provides the methods that are  
 516 used by the Registry to deliver asynchronous responses to the client. Note that a client  
 517 need not provide a RegistryClient interface if the [CPA] between the client and the  
 518 registry does not support asynchronous responses.

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

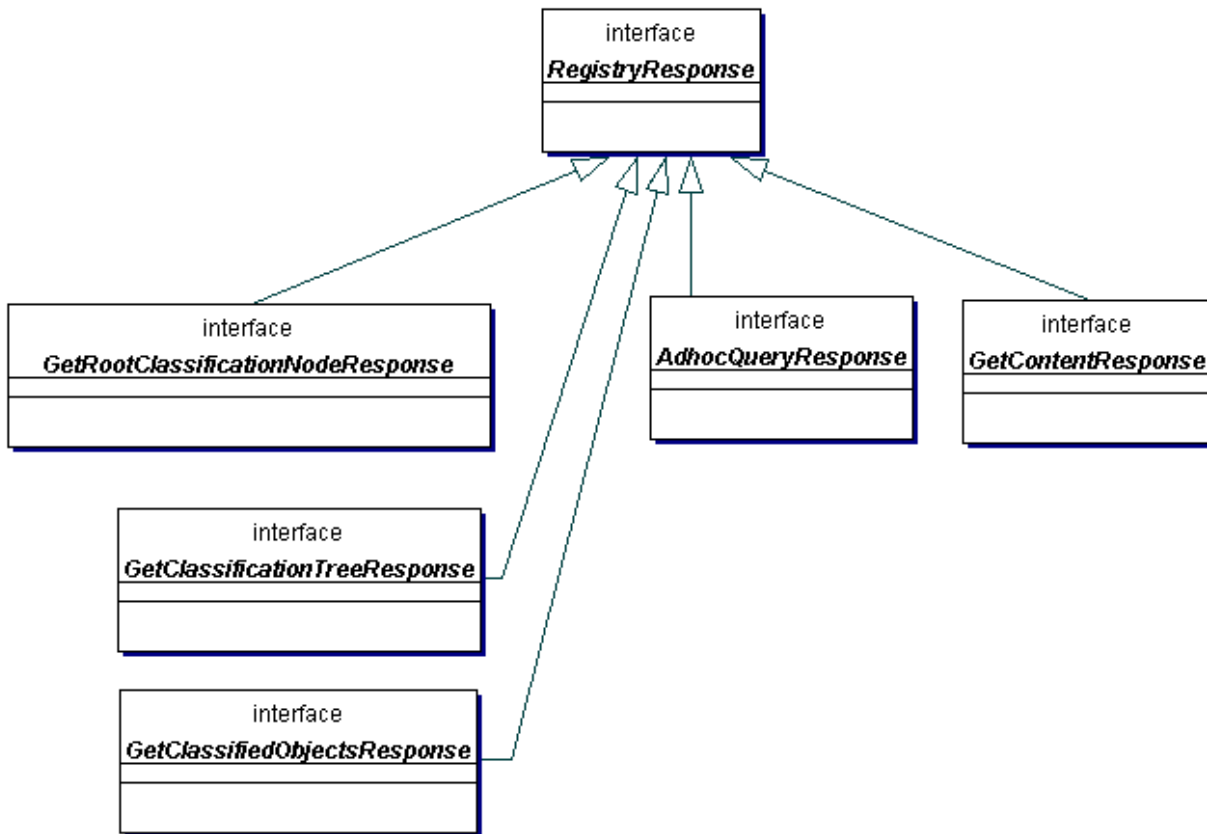
521

### **Method Summary of RegistryClient**

<pre>void onResponse(<a href="#">RegistryResponse</a> resp)</pre>	<p>Notifies client of the response sent by registry to previously submitted request.</p>
---	--

## 522 **6.7.4 Registry Response Class Hierarchy**

523 Since many of the responses from the registry have common attributes they are  
 524 arranged in the following class hierarchy. This hierarchy is reflected in the registry DTD.



525

526

Figure 6: Registry Response Class Hierarchy

## 527 7 Object Management Service

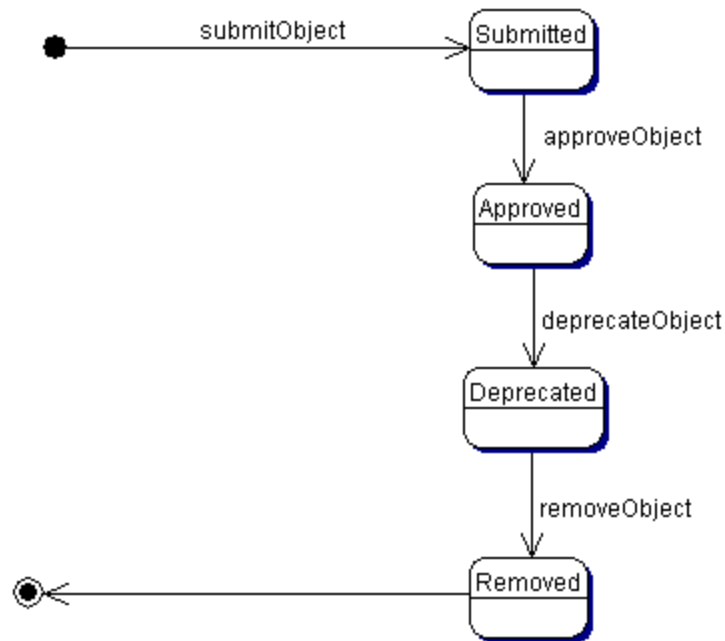
528 This section defines the ObjectManagement service of the Registry. The Object  
 529 Management Service is a sub-service of the Registry service. It provides the  
 530 functionality required by RegistryClients to manage the life cycle of repository items  
 531 (e.g. XML documents required for ebXML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

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

### 538 7.1 Life Cycle of a Repository Item

539 The main purpose of the ObjectManagement service is to manage the life cycle of  
 540 repository items.

541 Figure 7 shows the typical life cycle of a repository item. Note that the current version of  
 542 this specification does not support Object versioning. Object versioning will be added in  
 543 a future version of this specification.



544

545

Figure 7: Life Cycle of a Repository Item

## 546 7.2 RegistryObject Attributes

547 A repository item is associated with a set of standard metadata defined as attributes of  
 548 the RegistryObject class and its sub-classes as described in [ebRIM]. These attributes  
 549 reside outside of the actual repository item and catalog descriptive information about the  
 550 repository item. XML elements called ExtrinsicObject and IntrinsicObject (See Appendix  
 551 A for details) encapsulate all object metadata attributes defined in [ebRIM] as XML  
 552 attributes.

## 553 7.3 The Submit Objects Protocol

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

558



559

560

**Figure 8: Submit Objects Sequence Diagram**

561 For details on the schema for the *Business documents* shown in this process refer to  
562 Appendix A.

563 The SubmitObjectRequest message includes a RegistrEntryList element.

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

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

570 In the event of success, the registry sends a RegistryResponse with a status of  
571 “success” back to the client. In the event of failure, the registry sends a  
572 RegistryResponse with a status of “failure” back to the client.

### 573 **7.3.1 Universally Unique ID Generation**

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

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

578 This id is usually generated by the registry. The id attribute for submitted objects may  
579 optionally be supplied by the client. If the client supplies the id and it conforms to the  
580 format of a URN that specifies a DCE 128 bit UUID then the registry assumes that the  
581 client wishes to specify the id for the object. In this case, the registry must honor a  
582 client-supplied id and use it as the id attribute of the object in the registry. If the id is  
583 found by the registry to not be globally unique, the registry must raise the error  
584 condition: InvalidIdError.

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

### 589 **7.3.2 ID Attribute And Object References**

590 The id attribute of an object may be used by other objects to reference the first object.  
591 Such references are common both within the SubmitObjectsRequest as well as within  
592 the registry. Within a SubmitObjectsRequest, the id attribute may be used to refer to an  
593 object within the SubmitObjectsRequest as well as to refer to an object within the  
594 registry. An object in the SubmitObjectsRequest that needs to be referred to within the  
595 request document may be assigned an id by the submitter so that it can be referenced  
596 within the request. The submitter may give the object a proper uuid URN, in which case  
597 the id is permanently assigned to the object within the registry. Alternatively, the  
598 submitter may assign an arbitrary id (not a proper uuid URN) as long as the id is unique  
599 within the request document. In this case the id serves as a linkage mechanism within  
600 the request document but must be ignored by the registry and replaced with a registry  
601 generated id upon submission.

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

### 606 **7.3.3 Sample SubmitObjectsRequest**

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

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

```
613  
614 <?xml version = "1.0" encoding = "UTF-8"?>  
615 <!DOCTYPE SubmitObjectsRequest SYSTEM "file:///home/najmi/Registry.dtd">  
616  
617 <SubmitObjectsRequest>  
618   <RegistryEntryList>  
619
```



```
620 <!--
621 The following 3 objects package specified ExtrinsicObject in specified
622 Package, where both the Package and the ExtrinsicObject are
623 being submitted
624 -->
625 <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
626 <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
627   objectType = "CPP" name = "Widget Profile"
628   description = "ACME's profile for selling widgets"/>
629 <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
630   sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>
631
632 <!--
633 The following 3 objects package specified ExtrinsicObject in specified Package,
634 Where the Package is being submitted and the ExtrinsicObject is
635 already in registry
636 -->
637 <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
638 <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
639 <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
640   associationType = "Packages" sourceObject = "acmePackage2"
641   targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
642
643 <!--
644 The following 3 objects package specified ExtrinsicObject in specified Package,
645 where the Package and the ExtrinsicObject are already in registry
646 -->
647 <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
648 <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
649 <!-- id is unspecified implying that registry must create a uuid for this object -->
650 <Association associationType = "Packages"
651   sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
652   targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
653
654 <!--
655 The following 3 objects externally link specified ExtrinsicObject using
656 specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
657 are being submitted
658 -->
659 <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
660 <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
661   name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
662 <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
663   sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
664
665 <!--
666 The following 2 objects externally link specified ExtrinsicObject using specified
667 ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
668 is already in registry. Note that the targetObject points to an ObjectRef in a
669 previous line
670 -->
671 <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
672 <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
673   associationType = "ExternallyLinks" sourceObject = "acmeLink2"
674   targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
675
676 <!--
677 The following 2 objects externally identify specified ExtrinsicObject using specified
678 ExternalIdentifier, where the ExternalIdentifier is being submitted and the
679 ExtrinsicObject is already in registry. Note that the targetObject points to an
680 ObjectRef in a previous line
681 -->
682 <ExternalIdentifier id = "acmeDUNSID" name = "DUNS" description = "DUNS ID for ACME"
683   value = "13456789012"/>
684 <Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc"
685   associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSID"
686   targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
687
688 <!--
689 The following show submission of a brand new classification scheme in its entirety
```

```

690 -->
691 <ClassificationNode id = "geographyNode" name = "Geography"
692   description = "The Geography scheme example from Registry Services Spec" />
693 <ClassificationNode id = "asiaNode" name = "Asia"
694   description = "The Asia node under the Geography node" parent="geographyNode" />
695 <ClassificationNode id = "japanNode" name = "Japan"
696   description = "The Japan node under the Asia node" parent="asiaNode" />
697 <ClassificationNode id = "koreaNode" name = "Korea"
698   description = "The Korea node under the Asia node" parent="asiaNode" />
699 <ClassificationNode id = "europeNode" name = "Europe"
700   description = "The Europe node under the Geography node" parent="geographyNode" />
701 <ClassificationNode id = "germanyNode" name = "Germany"
702   description = "The Germany node under the Asia node" parent="europeNode" />
703 <ClassificationNode id = "northAmericaNode" name = "North America"
704   description = "The North America node under the Geography node"
705   parent="geographyNode" />
706 <ClassificationNode id = "usNode" name = "US"
707   description = "The US node under the Asia node" parent="northAmericaNode" />
708
709 <!--
710 The following show submission of a Automotive sub-tree of ClassificationNodes that
711 gets added to an existing classification scheme named 'Industry'
712 that is already in the registry
713 -->
714 <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
715 <ClassificationNode id = "automotiveNode" name = "Automotive"
716   description = "The Automotive sub-tree under Industry scheme"
717   parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
718 <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
719   description = "The Parts Supplier node under the Automotive node"
720   parent="automotiveNode" />
721 <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
722   description = "The Engine Supplier node under the Automotive node"
723   parent="automotiveNode" />
724
725 <!--
726 The following show submission of 2 Classifications of an object that is already in
727 the registry using 2 ClassificationNodes. One ClassificationNode
728 is being submitted in this request (Japan) while the other is already in the registry.
729 -->
730 <Classification id = "japanClassification"
731   description = "Classifies object by /Geography/Asia/Japan node"
732   classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
733   classificationNode="japanNode" />
734 <Classification id = "classificationUsingExistingNode"
735   description = "Classifies object using a node in the registry"
736   classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
737   classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
738 <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />
739
740
741 </RegistryEntryList>
742 </SubmitObjectsRequest>

```

## 743 7.4 The Add Slots Protocol

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



747

748

**Figure 9: Add Slots Sequence Diagram**

749 In the event of success, the registry sends a RegistryResponse with a status of  
 750 “success” back to the client. In the event of failure, the registry sends a  
 751 RegistryResponse with a status of “failure” back to the client.

## 752 7.5 The Remove Slots Protocol

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



755

756

**Figure 10: Remove Slots Sequence Diagram**

757 In the event of success, the registry sends a RegistryResponse with a status of  
 758 “success” back to the client. In the event of failure, the registry sends a  
 759 RegistryResponse with a status of “failure” back to the client.

## 760 7.6 The Approve Objects Protocol

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



765

766

Figure 11: Approve Objects Sequence Diagram

767 In the event of success, the registry sends a RegistryResponse with a status of  
 768 “success” back to the client. In the event of failure, the registry sends a  
 769 RegistryResponse with a status of “failure” back to the client.

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

## 772 7.7 The Deprecate Objects Protocol

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



778

779

**Figure 12: Deprecate Objects Sequence Diagram**

780 In the event of success, the registry sends a RegistryResponse with a status of  
 781 “success” back to the client. In the event of failure, the registry sends a  
 782 RegistryResponse with a status of “failure” back to the client.

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

## 785 **7.8 The Remove Objects Protocol**

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

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

### 792 **7.8.1 Deletion Scope DeleteRepositoryItemOnly**

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

796 **7.8.2 Deletion Scope DeleteAll**

797 This deletionScope specifies that the request should delete both the RegistryEntry and  
 798 the repository item for the specified registry entries. Only if all references (e.g.  
 799 Associations, Classifications, ExternalLinks) to a RegistryEntry have been removed, can  
 800 that RegistryEntry then be removed using a RemoveObjectsRequest with  
 801 deletionScope DeleteAll. Attempts to remove a RegistryEntry while it still has references  
 802 raises an error condition: InvalidRequestError.

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



804

805 **Figure 13: Remove Objects Sequence Diagram**

806 In the event of success, the registry sends a RegistryResponse with a status of  
 807 “success” back to the client. In the event of failure, the registry sends a  
 808 RegistryResponse with a status of “failure” back to the client.

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

811 **8 Object Query Management Service**

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

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

- 816 1. Browse and Drill Down Query
- 817 2. Filtered Query
- 818 3. SQL Query

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

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

826 Any errors in the query request messages are indicated in the corresponding query  
 827 response message.

## 828 8.1 Browse and Drill Down Query Support

829 The browse and drill down query style is supported by a set of interaction protocols  
 830 between the ObjectQueryManagerClient and the ObjectQueryManager. Sections 8.1.1,  
 831 8.1.2 and 8.1.3 describe these protocols.

### 832 8.1.1 Get Root Classification Nodes Request

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



839

840 **Figure 14: Get Root Classification Nodes Sequence Diagram**

841 In the event of success, the registry sends a GetRootClassificationNodeResponse with  
 842 a status of “success” back to the client. In the event of failure, the registry sends a  
 843 GetRootClassificationNodeResponse with a status of “failure” back to the client.

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

## 846 8.1.2 Get Classification Tree Request

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

853



854

855 Figure 15: Get Classification Tree Sequence Diagram

856 In the event of success, the registry sends a GetClassificationTreeResponse with a  
 857 status of "success" back to the client. In the event of failure, the registry sends a  
 858 GetClassificationTreeResponse with a status of "failure" back to the client.

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

## 861 8.1.3 Get Classified Objects Request

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

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

868 When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it should  
 869 return Objects that are:

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



872 **8.1.3.1 Get Classified Objects Request Example**

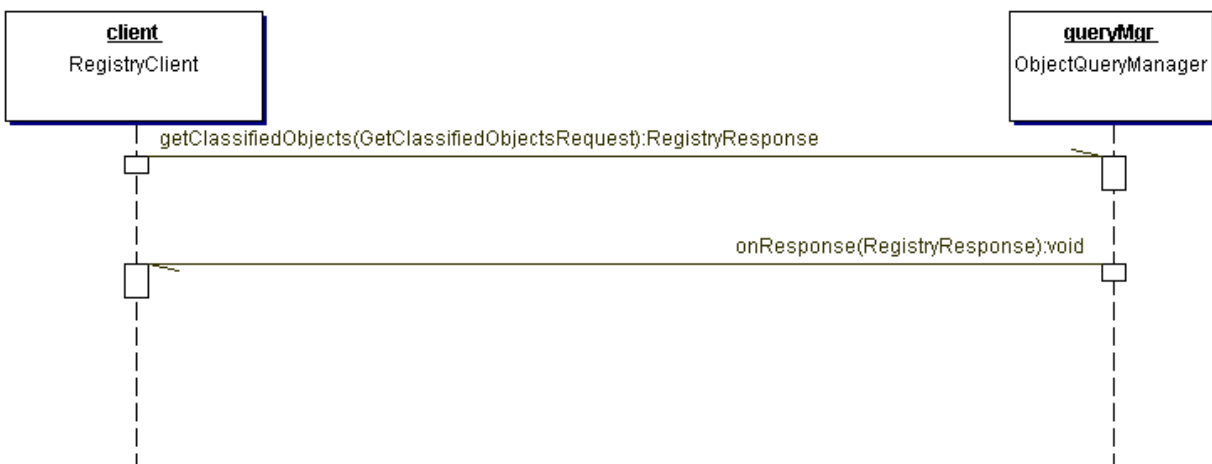
873

874

**Figure 16: A Sample Geography Classification**

875 Let us say a classification tree has the structure shown in Figure 16:

- 876
- 877
- 878
- 879
- 880
- 881
- 882
- 883
- 884
- 885
- 886
- 887
- 888
- 889
- If the Geography node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by Geography *or* North America *or* US *or* Asia *or* Japan *or* Korea *or* Europe *or* Germany.
  - If the Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by Asia *or* Japan *or* Korea.
  - If the Japan *and* Korea nodes are specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by both Japan *and* Korea.
  - If the North America *and* Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by (North America *or* US) *and* (Asia *or* Japan *or* Korea).



890

891

**Figure 17: Get Classified Objects Sequence Diagram**

892  
893  
894  
895  
896

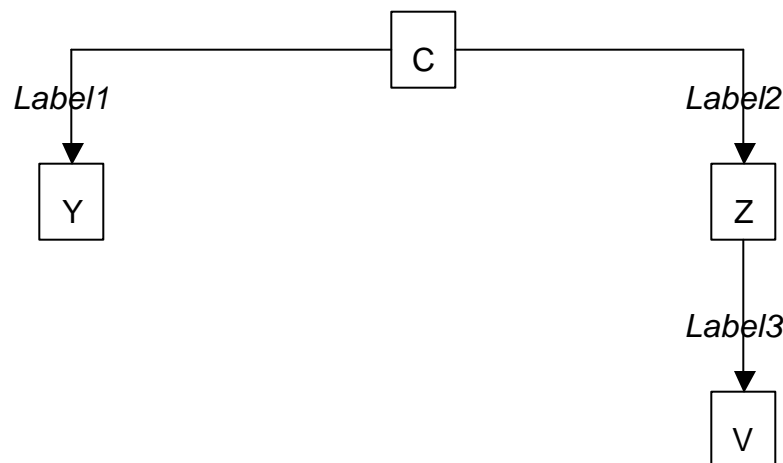
**8.2 In the event of success, the registry sends a GetClassifiedObjectsResponse with a status of “success” back to the client. In the event of failure, the registry sends a GetClassifiedObjectsResponse with a status of “failure” back to the client.**

## 897 **Filter Query Support**

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

904 A client submits a FilterQuery, a ReturnRegistryEntry query, or a ReturnRepositoryItem  
905 query to the ObjectQueryManager as part of an AdhocQueryRequest. The  
906 ObjectQueryManager sends an AdhocQueryResponse back to the client, enclosing the  
907 appropriate FilterQueryResponse, ReturnRegistryEntryResponse, or  
908 ReturnRepositoryItemResponse specified herein. The sequence diagrams for  
909 AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.4.

910 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a  
911 hierarchy of classes derived from a single class and its associations with other classes  
912 as defined by ebRIM. Each choice of a class pre-determines a virtual XML document  
913 that can be queried as a tree. For example, let C be a class, let Y and Z be classes that  
914 have direct associations to C, and let V be a class that is associated with Z. The ebRIM  
915 Binding for C might be as in Figure 18.



916  
917  
918  
919  
920  
921  
922  
923  
924  
925 **Figure 18: Example ebRIM Binding**

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

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

939 In the above example, the CQuery element will have three subelements, one a CFilter  
940 on the C class to eliminate C instances that do not satisfy the predicate of the CFilter,  
941 another a YFilter on the Y class to eliminate branches from C to Y where the target of  
942 the association does not satisfy the YFilter, and a third to eliminate branches along a  
943 path from C through Z to V. The third element is called a *branch* element because it  
944 allows class filters on each class along the path from X to V. In general, a branch  
945 element will have subelements that are themselves class filters, other branch elements,  
946 or a full-blown query on the terminal class in the path.

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

952 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is  
953 intended to be stable, the FilterQuery syntax is stable. However, if new structures are  
954 added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the  
955 same time.

956 Support for FilterQuery is required of every conforming ebXML Registry implementation,  
957 but other query options are possible. The Registry will hold a self-describing CPP that  
958 identifies all supported AdhocQuery options. This profile is described in Section 1.1.1.1.

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

962 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a  
963 way to structure an XML document as an expansion of the result of a  
964 RegistryEntryQuery. The ReturnRegistryEntry element specified in Section 8.2.7 allows  
965 one to specify what metadata one wants returned with each registry entry identified in  
966 the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section  
967 8.2.8 allows one to specify what repository items one wants returned based on their  
968 relationships to the registry entries identified by the result of a RegistryEntryQuery.  
969

969 **8.2.1 FilterQuery**970 **Purpose**

971 To identify a set of registry instances from a specific registry class. Each alternative  
 972 assumes a specific binding to ebRIM. The query result for each query alternative is a  
 973 set of references to instances of the root class specified by the binding. The status is a  
 974 success indication or a collection of warnings and/or exceptions.

975 **Definition**

```

976
977 <!ELEMENT FilterQuery
978   (
979     | RegistryEntryQuery
980     | AuditableEventQuery
981     | ClassificationNodeQuery
982     | RegistryPackageQuery
983     | OrganizationQuery          )>
984
985 <!ELEMENT FilterQueryResult
986   (
987     | RegistryEntryQueryResult
988     | AuditableEventQueryResult
989     | ClassificationNodeQueryResult
990     | RegistryPackageQueryResult
991     | OrganizationQueryResult   )>
992
993 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
994
995 <!ELEMENT RegistryEntryView EMPTY >
996 <!ATTLIST RegistryEntryView
997   objectURN      CDATA      #REQUIRED
998   contentURI     CDATA      #IMPLIED
999   objectID       CDATA      #IMPLIED >
1000
1001 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
1002
1003 <!ELEMENT AuditableEventView EMPTY >
1004 <!ATTLIST AuditableEventView
1005   objectID       CDATA      #REQUIRED
1006   timestamp     CDATA      #REQUIRED >
1007
1008 <!ELEMENT ClassificationNodeQueryResult
1009   (ClassificationNodeView*)>
1010
1011 <!ELEMENT ClassificationNodeView EMPTY >
1012 <!ATTLIST ClassificationNodeView
1013   objectURN      CDATA      #REQUIRED
1014   contentURI     CDATA      #IMPLIED
1015   objectID       CDATA      #IMPLIED >
1016
1017 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
1018
1019 <!ELEMENT RegistryPackageView EMPTY >
1020 <!ATTLIST RegistryPackageView

```

```

1019     objectURN      CDATA      #REQUIRED
1020     contentURI     CDATA      #IMPLIED
1021     objectID       CDATA      #IMPLIED  >
1022
1023     <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
1024
1025     <!ELEMENT OrganizationView EMPTY >
1026     <!ATTLIST OrganizationView
1027         orgURN      CDATA      #REQUIRED
1028         objectID    CDATA      #IMPLIED  >
1029
1030

```

### 1031 **Semantic Rules**

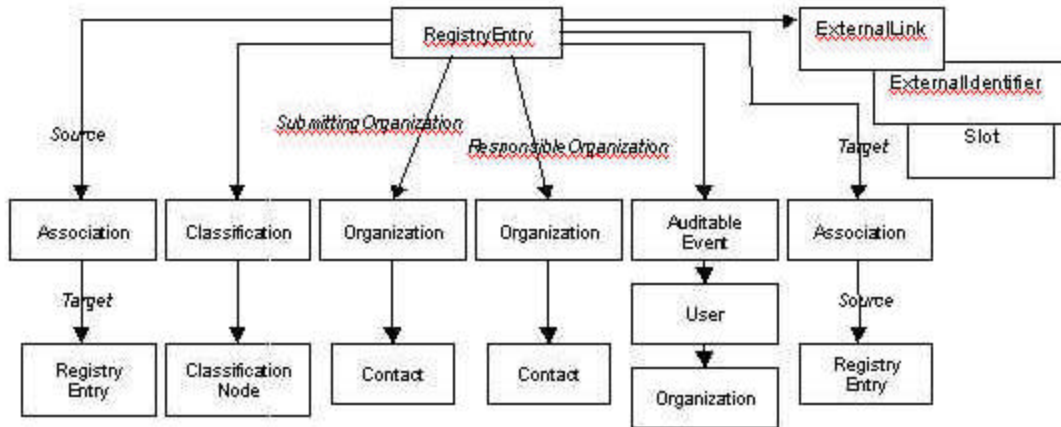
- 1032 1. The semantic rules for each FilterQuery alternative are specified in subsequent  
1033 subsections.
- 1034 2. Each FilterQueryResult is a set of XML reference elements to identify each instance  
1035 of the result set. Each XML attribute carries a value derived from the value of an  
1036 attribute specified in the Registry Information Model as follows:
- 1037 a) objectID is the value of the ID attribute of the RegistryObject class,  
1038 b) objectURN and orgURN are URN values derived from the object ID ,  
1039 c) contentURI is a URL value derived from the contentURI attribute of the  
1040 RegistryEntry class,  
1041 d) timestamp is a literal value to represent the value of the timestamp attribute of  
1042 the AuditableEvent class.
- 1043 3. If an error condition is raised during any part of the execution of a FilterQuery, then  
1044 the status attribute of the XML RegistryResult is set to “failure” and no query result  
1045 element is returned; instead, a RegistryErrorList element must be returned with its  
1046 highestSeverity element set to “error”. At least one of the RegistryError elements in  
1047 the RegistryErrorList will have its severity attribute set to “error”.
- 1048 4. If no error conditions are raised during execution of a FilterQuery, then the status  
1049 attribute of the XML RegistryResult is set to “success” and an appropriate query  
1050 result element must be included. If a RegistryErrorList is also returned, then the  
1051 highestSeverity attribute of the RegistryErrorList is set to “warning” and the serverity  
1052 attribute of each RegistryError is set to “warning”.
- 1053
- 1054
- 1055

1055 **8.2.2 RegistryEntryQuery**1056 **Purpose**

1057 To identify a set of registry entry instances as the result of a query over selected registry  
1058 metadata.

1059 **ebRIM Binding**

1060



1061

1062 **Definition**

1063

```

1064 <!ELEMENT RegistryEntryQuery
1065   ( RegistryEntryFilter?,
1066     SourceAssociationBranch*,
1067     TargetAssociationBranch*,
1068     HasClassificationBranch*,
1069     SubmittingOrganizationBranch?,
1070     ResponsibleOrganizationBranch?,
1071     ExternalIdentifierFilter*,
1072     ExternalLinkFilter*,
1073     SlotFilter*,
1074     HasAuditableEventBranch* )>
1075
1076 <!ELEMENT SourceAssociationBranch
1077   ( AssociationFilter?,
1078     RegistryEntryFilter? )>
1079
1080 <!ELEMENT TargetAssociationBranch
1081   ( AssociationFilter?,
1082     RegistryEntryFilter? )>
1083
1084 <!ELEMENT HasClassificationBranch
1085   ( ClassificationFilter?,
1086     ClassificationNodeFilter? )>
  
```

```
1087
1088 <!ELEMENT SubmittingOrganizationBranch
1089 ( OrganizationFilter?,
1090 ContactFilter? )>
1091
1092 <!ELEMENT ResponsibleOrganizationBranch
1093 ( OrganizationFilter?,
1094 ContactFilter? )>
1095
1096 <!ELEMENT HasAuditableEventBranch
1097 ( AuditableEventFilter?,
1098 UserFilter?,
1099 OrganizationFilter? )>
```

## 1100 **Semantic Rules**

- 1101 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The  
1102 following steps will eliminate instances in RE that do not satisfy the conditions of the  
1103 specified filters.
- 1104 a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below;  
1105 otherwise, let x be a registry entry in RE. If x does not satisfy the  
1106 RegistryEntryFilter as defined in Section 8.2.9, then remove x from RE.
- 1107 b) If a SourceAssociationBranch element is not specified, or if RE is empty, then  
1108 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1109 source object of some Association instance, then remove x from RE; otherwise,  
1110 treat each SourceAssociationBranch element separately as follows:
- 1111 If no AssociationFilter is specified within SourceAssociationBranch, then let AF  
1112 be the set of all Association instances that have x as a source object; otherwise,  
1113 let AF be the set of Association instances that satisfy the AssociationFilter and  
1114 have x as the source object. If AF is empty, then remove x from RE. If no  
1115 RegistryEntryFilter is specified within SourceAssociationBranch, then let RET be  
1116 the set of all RegistryEntry instances that are the target object of some element  
1117 of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the  
1118 RegistryEntryFilter and are the target object of some element of AF. If RET is  
1119 empty, then remove x from RE.
- 1120 c) If a TargetAssociationBranch element is not specified, or if RE is empty, then  
1121 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1122 target object of some Association instance, then remove x from RE; otherwise,  
1123 treat each TargetAssociationBranch element separately as follows:



- 1124 If no AssociationFilter is specified within TargetAssociationBranch, then let AF be  
1125 the set of all Association instances that have x as a target object; otherwise, let  
1126 AF be the set of Association instances that satisfy the AssociationFilter and have  
1127 x as the target object. If AF is empty, then remove x from RE. If no  
1128 RegistryEntryFilter is specified within TargetAssociationBranch, then let RES be  
1129 the set of all RegistryEntry instances that are the source object of some element  
1130 of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the  
1131 RegistryEntryFilter and are the source object of some element of AF. If RES is  
1132 empty, then remove x from RE.
- 1133 d) If a HasClassificationBranch element is not specified, or if RE is empty, then  
1134 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the  
1135 source object of some Classification instance, then remove x from RE; otherwise,  
1136 treat each HasClassificationBranch element separately as follows:
- 1137 If no ClassificationFilter is specified within the HasClassificationBranch, then let  
1138 CL be the set of all Classification instances that have x as a source object;  
1139 otherwise, let CL be the set of Classification instances that satisfy the  
1140 ClassificationFilter and have x as the source object. If CL is empty, then remove  
1141 x from RE. If no ClassificationNodeFilter is specified within  
1142 HasClassificationBranch, then let CN be the set of all ClassificationNode  
1143 instances that are the target object of some element of CL; otherwise, let CN be  
1144 the set of RegistryEntry instances that satisfy the ClassificationNodeFilter and  
1145 are the target object of some element of CL. If CN is empty, then remove x from  
1146 RE.
- 1147 e) If a SubmittingOrganizationBranch element is not specified, or if RE is empty,  
1148 then continue below; otherwise, let x be a remaining registry entry in RE. If x  
1149 does not have a submitting organization, then remove x from RE. If no  
1150 OrganizationFilter is specified within SubmittingOrganizationBranch, then let SO  
1151 be the set of all Organization instances that are the submitting organization for x;  
1152 otherwise, let SO be the set of Organization instances that satisfy the  
1153 OrganizationFilter and are the submitting organization for x. If SO is empty, then  
1154 remove x from RE. If no ContactFilter is specified within  
1155 SubmittingOrganizationBranch, then let CT be the set of all Contact instances  
1156 that are the contacts for some element of SO; otherwise, let CT be the set of  
1157 Contact instances that satisfy the ContactFilter and are the contacts for some  
1158 element of SO. If CT is empty, then remove x from RE.

- 1159 f) If a ResponsibleOrganizationBranch element is not specified, or if RE is empty,  
1160 then continue below; otherwise, let x be a remaining registry entry in RE. If x  
1161 does not have a responsible organization, then remove x from RE. If no  
1162 OrganizationFilter is specified within ResponsibleOrganizationBranch, then let  
1163 RO be the set of all Organization instances that are the responsible organization  
1164 for x; otherwise, let RO be the set of Organization instances that satisfy the  
1165 OrganizationFilter and are the responsible organization for x. If RO is empty, then  
1166 remove x from RE. If no ContactFilter is specified within  
1167 SubmittingOrganizationBranch, then let CT be the set of all Contact instances  
1168 that are the contacts for some element of RO; otherwise, let CT be the set of  
1169 Contact instances that satisfy the ContactFilter and are the contacts for some  
1170 element of RO. If CT is empty, then remove x from RE.
- 1171 g) If an ExternalLinkFilter element is not specified, or if RE is empty, then continue  
1172 below; otherwise, let x be a remaining registry entry in RE. If x is not linked to  
1173 some ExternalLink instance, then remove x from RE; otherwise, treat each  
1174 ExternalLinkFilter element separately as follows:  
1175 Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter and  
1176 are linked to x. If EL is empty, then remove x from RE.
- 1177 h) If an ExternalIdentifierFilter element is not specified, or if RE is empty, then  
1178 continue below; otherwise, let x be a remaining registry entry in RE. If x is not  
1179 linked to some ExternalIdentifier instance, then remove x from RE; otherwise,  
1180 treat each ExternalIdentifierFilter element separately as follows:  
1181 Let EI be the set of ExternalIdentifier instances that satisfy the  
1182 ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RE.
- 1183 i) If a SlotFilter element is not specified, or if RE is empty, then continue below;  
1184 otherwise, let x be a remaining registry entry in RE. If x is not linked to some Slot  
1185 instance, then remove x from RE; otherwise, treat each SlotFilter element  
1186 separately as follows:  
1187 Let SL be the set of Slot instances that satisfy the SlotFilter and are linked to x. If  
1188 SL is empty, then remove x from RE.
- 1189 j) If a HasAuditableEventBranch element is not specified, or if RE is empty, then  
1190 continue below; otherwise, let x be a remaining registry entry in RE. If x is not  
1191 linked to some AuditableEvent instance, then remove x from RE; otherwise, treat  
1192 each HasAuditableEventBranch element separately as follows:  
1193 If an AuditableEventFilter is not specified within HasAuditableEventBranch, then  
1194 let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the  
1195 set of AuditableEvent instances that satisfy the AuditableEventFilter and are  
1196 auditable events for x. If AE is empty, then remove x from RE. If a UserFilter is  
1197 not specified within HasAuditableEventBranch, then let AI be the set of all User  
1198 instances linked to an element of AE; otherwise, let AI be the set of User  
1199 instances that satisfy the UserFilter and are linked to an element of AE.

1200 If AI is empty, then remove x from RE. If an OrganizationFilter is not specified  
 1201 within HasAuditableEventBranch, then let OG be the set of all Organization  
 1202 instances that are linked to an element of AI; otherwise, let OG be the set of  
 1203 Organization instances that satisfy the OrganizationFilter and are linked to an  
 1204 element of AI. If OG is empty, then remove x from RE.

1205 2. If RE is empty, then raise the warning: *registry entry query result is empty*.

1206 3. Return RE as the result of the RegistryEntryQuery.

1207

## 1208 Examples

1209 A client wants to establish a trading relationship with XYZ Corporation and wants to  
 1210 know if they have registered any of their business documents in the Registry. The  
 1211 following query returns a set of registry entry identifiers for currently registered items  
 1212 submitted by any organization whose name includes the string "XYZ". It does not return  
 1213 any registry entry identifiers for superseded, replaced, deprecated, or withdrawn items.

```

1214
1215 <RegistryEntryQuery>
1216   <RegistryEntryFilter>
1217     status EQUAL "Approved"           -- code by Clause, Section 8.2.10
1218   </RegistryEntryFilter>
1219   <SubmittingOrganizationBranch>
1220     <OrganizationFilter>
1221       name CONTAINS "XYZ"             -- code by Clause, Section 8.2.10
1222     </OrganizationFilter>
1223   </SubmittingOrganizationBranch>
1224 </RegistryEntryQuery>
1225
```

1226 A client is using the United Nations Standard Product and Services Classification  
 1227 (UNSPSC) scheme and wants to identify all companies that deal with products  
 1228 classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client  
 1229 knows that companies have registered their party profile documents in the Registry, and  
 1230 that each profile has been classified by the products the company deals with. The  
 1231 following query returns a set of registry entry identifiers for profiles of companies that  
 1232 deal with integrated circuit components.

```

1233
1234 <RegistryEntryQuery>
1235   <RegistryEntryFilter>
1236     objectType EQUAL "CPP" AND         -- code by Clause, Section 8.2.10
1237     status EQUAL "Approved"
1238   </RegistryEntryFilter>
1239   <HasClassificationBranch>
1240     <ClassificationNodeFilter>
1241       id STARTSWITH "urn:un:spsc:321118" -- code by Clause, Section 8.2.10
1242     </ClassificationNodeFilter>
1243   </HasClassificationBranch>
1244 </RegistryEntryQuery>

```

1245 A client application needs all items that are classified by two different classification  
1246 schemes, one based on "Industry" and another based on "Geography". Both schemes  
1247 have been defined by ebXML and are registered. The root nodes of each scheme are  
1248 identified by "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The  
1249 following query identifies registry entries for all registered items that are classified by  
1250 "Industry/Automotive" and by "Geography/Asia/Japan".

```
1251
1252 <RegistryEntryQuery>
1253   <HasClassificationBranch>
1254     <ClassificationNodeFilter>
1255       id STARTSWITH "urn:ebxml:cs:industry" AND
1256       path EQUAL "Industry/Automotive"      -- code by Clause, Section 8.2.10
1257     </ClassificationNodeFilter>
1258     <ClassificationNodeFilter>
1259       id STARTSWITH "urn:ebxml:cs:geography" AND
1260       path EQUAL "Geography/Asia/Japan"    -- code by Clause, Section 8.2.10
1261     </ClassificationNodeFilter>
1262   </HasClassificationBranch>
1263 </RegistryEntryQuery>
```

1264 A client application wishes to identify all registry Package instances that have a given  
1265 registry entry as a member of the package. The following query identifies all registry  
1266 packages that contain the registry entry identified by URN "urn:path:myitem" as a  
1267 member:

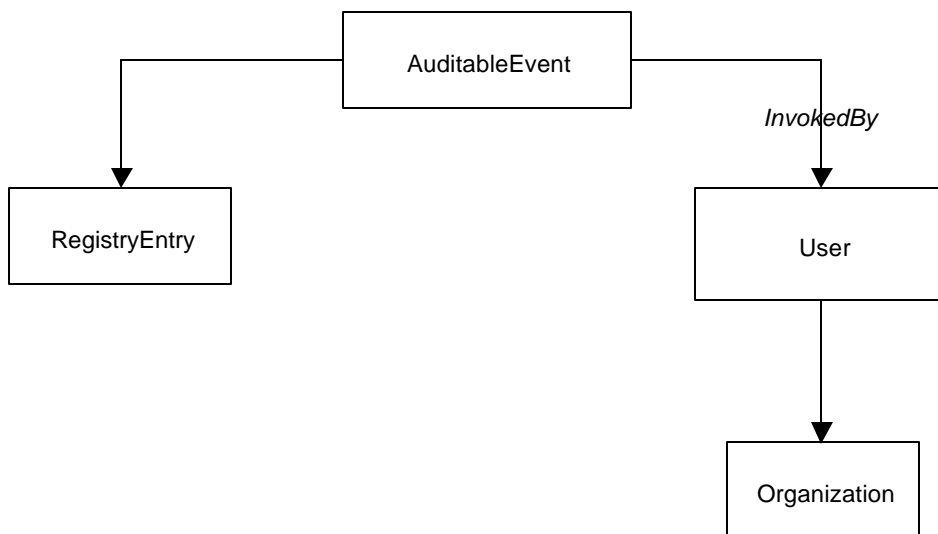
```
1268
1269 <RegistryEntryQuery>
1270   <RegistryEntryFilter>
1271     objectType EQUAL "RegistryPackage"      -- code by Clause, Section 8.2.10
1272   </RegistryEntryFilter>
1273   <SourceAssociationBranch>
1274     <AssociationFilter>                    -- code by Clause, Section 8.2.10
1275       associationType EQUAL "HasMember" AND
1276       targetObject EQUAL "urn:path:myitem"
1277     </AssociationFilter>
1278   </SourceAssociationBranch>
1279 </RegistryEntryQuery>
```

1280 A client application wishes to identify all ClassificationNode instances that have some  
1281 given keyword as part of their name or description. The following query identifies all  
1282 registry classification nodes that contain the keyword "transistor" as part of their name  
1283 or as part of their description.

```
1284
1285 <RegistryEntryQuery>
1286   <RegistryEntryFilter>
1287     ObjectType="ClassificationNode" AND
1288     (name CONTAINS "transistor" OR        -- code by Clause, Section 8.2.10
1289     description CONTAINS "transistor")
1290   </RegistryEntryFilter>
1291 </RegistryEntryQuery>
1292
```

1292 **8.2.3 AuditableEventQuery**1293 **Purpose**

1294 To identify a set of auditable event instances as the result of a query over selected  
 1295 registry metadata.

1296 **ebRIM Binding**1297 **Definition**

```

1298
1299 <!ELEMENT AuditableEventQuery
1300   ( AuditableEventFilter?,
1301     RegistryEntryQuery*,
1302     InvokedByBranch? )>
1303
1304 <!ELEMENT InvokedByBranch
1305   ( UserFilter?,
1306     OrganizationQuery? )>
  
```

1307

1308 **Semantic Rules**

1309 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The  
 1310 following steps will eliminate instances in AE that do not satisfy the conditions of the  
 1311 specified filters.

1312

- 1313 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below;  
 1314 otherwise, let x be an auditable event in AE. If x does not satisfy the  
 1315 AuditableEventFilter as defined in Section 8.2.9, then remove x from AE.
- 1316 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue  
 1317 below; otherwise, let x be a remaining auditable event in AE. Treat each  
 1318 RegistryEntryQuery element separately as follows:  
 1319 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If  
 1320 x is not an auditable event for some registry entry in RE, then remove x from AE.
- 1321 c) If an InvokedByBranch element is not specified, or if AE is empty, then continue  
 1322 below; otherwise, let x be a remaining auditable event in AE.
- 1323 Let u be the user instance that invokes x. If a UserFilter element is specified within the  
 1324 InvokedByBranch, and if u does not satisfy that filter, then remove x from AE; otherwise,  
 1325 continue below.
- 1326 If an OrganizationQuery element is not specified within the InvokedByBranch,  
 1327 then continue below; otherwise, let OG be the set of Organization instances that  
 1328 are identified by the organization attribute of u and are in the result set of the  
 1329 OrganizationQuery. If OG is empty, then remove x from AE.
- 1330 2. If AE is empty, then raise the warning: *auditable event query result is empty*.
- 1331 3. Return AE as the result of the AuditableEventQuery.

1332

### 1333 Examples

1334 A Registry client has registered an item and it has been assigned a URN identifier  
 1335 "urn:path:myitem". The client is now interested in all events since the beginning of the  
 1336 year that have impacted that item. The following query will return a set of  
 1337 AuditableEvent identifiers for all such events.

```

1338 <AuditableEventquery>
1339   <AuditableEventFilter>
1340     timestamp GE "2001-01-01" AND -- code by Clause, Section 8.2.10
1341     registryEntry EQUAL "urn:path:myitem"
1342   </AuditableEventFilter>
1343 </AuditableEventQuery>
  
```

1345

1346 A client company has many registered objects in the Registry. The Registry allows  
 1347 events submitted by other organizations to have an impact on your registered items,  
 1348 e.g. new classifications and new associations. The following query will return a set of  
 1349 identifiers for all auditable events, invoked by some other party, that had an impact on  
 1350 an item submitted by "myorg" and for which "myorg" is the responsible organization.

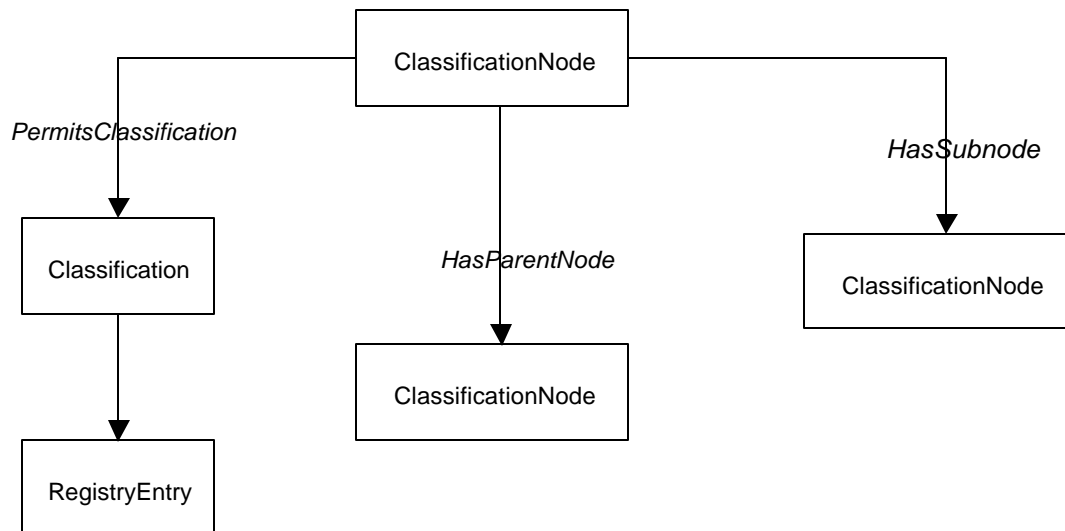
```

1351 <AuditableEventQuery>
1352   <RegistryEntryQuery>
1353
  
```

```
1354     <SubmittingOrganizationBranch>
1355         <OrganizationFilter>
1356             id EQUAL "urn:somepath:myorg"      -- code by Clause, Section 8.2.10
1357         </OrganizationFilter>
1358     </SubmittingOrganizationBranch>
1359     <ResponsibleOrganizationBranch>
1360         <OrganizationFilter>
1361             id EQUAL "urn:somepath:myorg"      -- code by Clause, Section 8.2.10
1362         </OrganizationFilter>
1363     </ResponsibleOrganizationBranch>
1364 </RegistryEntryQuery>
1365 <InvokedByBranch>
1366     <OrganizationQuery>
1367         <OrganizationFilter>
1368             id -EQUAL "urn:somepath:myorg"    -- code by Clause, Section 8.2.10
1369         </OrganizationFilter>
1370     </OrganizationQuery>
1371 </InvokedByBranch>
1372 </AuditableEventQuery>
1373
```

1373 **8.2.4 ClassificationNodeQuery**1374 **Purpose**

1375 To identify a set of classification node instances as the result of a query over selected  
 1376 registry metadata.

1377 **ebRIM Binding**1378 **Definition**

```

1379
1380 <!--ELEMENT ClassificationNodeQuery
1381   ( ClassificationNodeFilter?,
1382     PermitsClassificationBranch*,
1383     HasParentNode?,
1384     HasSubnode*           )>
1385
1386 <!--ELEMENT PermitsClassificationBranch
1387   ( ClassificationFilter?,
1388     RegistryEntryQuery?   )>
1389
1390 <!--ELEMENT HasParentNode
1391   ( ClassificationNodeFilter?,
1392     HasParentNode?        )>
1393
1394 <!--ELEMENT HasSubnode
1395   ( ClassificationNodeFilter?,
1396     HasSubnode*           )>
  
```

1397

1398



1399 **Semantic Rules**

- 1400 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry.  
1401 The following steps will eliminate instances in CN that do not satisfy the conditions of  
1402 the specified filters.
- 1403 a) If a ClassificationNodeFilter is not specified, or if CN is empty, then continue  
1404 below; otherwise, let x be a classification node in CN. If x does not satisfy the  
1405 ClassificationNodeFilter as defined in Section 8.2.9, then remove x from AE.
- 1406 b) If a PermitsClassificationBranch element is not specified, or if CN is empty, then  
1407 continue below; otherwise, let x be a remaining classification node in CN. If x is  
1408 not the target object of some Classification instance, then remove x from CN;  
1409 otherwise, treat each PermitsClassificationBranch element separately as follows:
- 1410 If no ClassificationFilter is specified within the PermitsClassificationBranch  
1411 element, then let CL be the set of all Classification instances that have x as the  
1412 target object; otherwise, let CL be the set of Classification instances that satisfy  
1413 the ClassificationFilter and have x as the target object. If CL is empty, then  
1414 remove x from CN. If no RegistryEntryQuery is specified within the  
1415 PermitsClassificationBranch element, then let RES be the set of all RegistryEntry  
1416 instances that are the source object of some classification instance in CL;  
1417 otherwise, let RE be the result set of the RegistryEntryQuery as defined in  
1418 Section 8.2.2 and let RES be the set of all instances in RE that are the source  
1419 object of some classification in CL. If RES is empty, then remove x from CN.
- 1420 c) If a HasParentNode element is not specified, or if CN is empty, then continue  
1421 below; otherwise, let x be a remaining classification node in CN and execute the  
1422 following paragraph with  $n=x$ .
- 1423 Let n be a classification node instance. If n does not have a parent node (i.e. if n  
1424 is a root node), then remove x from CN. Let p be the parent node of n. If a  
1425 ClassificationNodeFilter element is directly contained in HasParentNode and if p  
1426 does not satisfy the ClassificationNodeFilter, then remove x from CN.
- 1427 If another HasParentNode element is directly contained within this  
1428 HasParentNode element, then repeat the previous paragraph with  $n=p$ .
- 1429 d) If a HasSubnode element is not specified, or if CN is empty, then continue below;  
1430 otherwise, let x be a remaining classification node in CN. If x is not the parent  
1431 node of some ClassificationNode instance, then remove x from CN; otherwise,  
1432 treat each HasSubnode element separately and execute the following paragraph  
1433 with  $n = x$ .
- 1434 Let n be a classification node instance. If a ClassificationNodeFilter is not  
1435 specified within the HasSubnode element then let CNC be the set of all  
1436 classification nodes that have n as their parent node; otherwise, let CNC be the  
1437 set of all classification nodes that satisfy the ClassificationNodeFilter and have n  
1438 as their parent node. If CNC is empty then remove x from CN; otherwise, let y be  
1439 an element of CNC and continue with the next paragraph.

1440 If the HasSubnode element is terminal, i.e. if it does not directly contain another  
1441 HasSubnode element, then continue below; otherwise, repeat the previous  
1442 paragraph with the new HasSubnode element and with  $n = y$ .

1443 2. If CN is empty, then raise the warning: *classification node query result is empty*.

1444 3. Return CN as the result of the ClassificationNodeQuery.

1445

#### 1446 **Examples**

1447 A client application wishes to identify all classification nodes defined in the Registry that  
1448 are root nodes and have a name that contains the phrase “product code” or the phrase  
1449 “product type”. Note: By convention, if a classification node has no parent (i.e. is a root  
1450 node), then the parent attribute of that instance is set to null and is represented as a  
1451 literal by a zero length string.

```
1452  
1453 <ClassificationNodeQuery>  
1454   <ClassificationNodeFilter>  
1455     (name CONTAINS "product code" OR      -- code by Clause, Section 8.2.10  
1456      name CONTAINS "product type") AND  
1457     parent EQUAL ""  
1458   </ClassificationNodeFilter>  
1459 </ClassificationNodeQuery>
```

1460

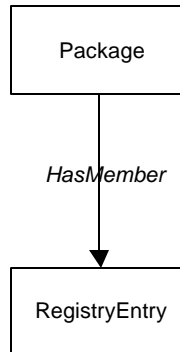
1461 A client application wishes to identify all of the classification nodes at the third level of a  
1462 classification scheme hierarchy. The client knows that the URN identifier for the root  
1463 node is “urn:ebxml:cs:myroot”. The following query identifies all nodes at the second  
1464 level under “myroot” (i.e. third level overall).

```
1465  
1466 <ClassificationNodeQuery>  
1467   <HasParentNode>  
1468     <HasParentNode>  
1469       <ClassificationNodeFilter>  
1470         id EQ "urn:ebxml:cs:myroot"  -- code by Clause, Section 8.2.10  
1471       </ClassificationNodeFilter>  
1472     </HasParentNode>  
1473   </HasParentNode>  
1474 </ClassificationNodeQuery>
```

1475

1475 **8.2.5 RegistryPackageQuery**1476 **Purpose**

1477 To identify a set of registry package instances as the result of a query over selected  
 1478 registry metadata.

1479 **ebRIM Binding**1480 **Definition**

```

1481
1482 <!ELEMENT RegistryPackageQuery
1483   ( PackageFilter?,
1484     HasMemberBranch*   )>
1485
1486 <!ELEMENT HasMemberBranch
1487   ( RegistryEntryQuery?   )>
  
```

1488

1489 **Semantic Rules**

1490 1. Let RP denote the set of all persistent Package instances in the Registry. The  
 1491 following steps will eliminate instances in RP that do not satisfy the conditions of the  
 1492 specified filters.

1493 a) If a PackageFilter is not specified, or if RP is empty, then continue below;  
 1494 otherwise, let x be a package instance in RP. If x does not satisfy the  
 1495 PackageFilter as defined in Section 8.2.9, then remove x from RP.

1496 b) If a HasMemberBranch element is not directly contained in the  
 1497 RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x  
 1498 be a remaining package instance in RP. If x is an empty package, then remove x  
 1499 from RP; otherwise, treat each HasMemberBranch element separately as  
 1500 follows:

1501

1502 If a RegistryEntryQuery element is not directly contained in the  
1503 HasMemberBranch element, then let PM be the set of all RegistryEntry instances  
1504 that are members of the package x; otherwise, let RE be the set of RegistryEntry  
1505 instances returned by the RegistryEntryQuery as defined in Section 8.2.2 and let  
1506 PM be the subset of RE that are members of the package x. If PM is empty, then  
1507 remove x from RP.

1508 2. If RP is empty, then raise the warning: *registry package query result is empty*.

1509 3. Return RP as the result of the RegistryPackageQuery.

1510

### 1511 Examples

1512 A client application wishes to identify all package instances in the Registry that contain  
1513 an Invoice extrinsic object as a member of the package.

1514

```
1515 <RegistryPackageQuery>
1516   <HasMemberBranch>
1517     <RegistryEntryQuery>
1518       <RegistryEntryFilter>
1519         objectType EQ "Invoice"      -- code by Clause, Section 8.2.10
1520       </RegistryEntryFilter>
1521     </RegistryEntryQuery>
1522   </HasMemberBranch>
1523 </RegistryPackageQuery>
1524
```

1525 A client application wishes to identify all package instances in the Registry that are not  
1526 empty.

1527

```
1528 <RegistryEntryQuery>
1529   <HasMemberBranch/>
1530 </RegistryEntryQuery>
1531
```

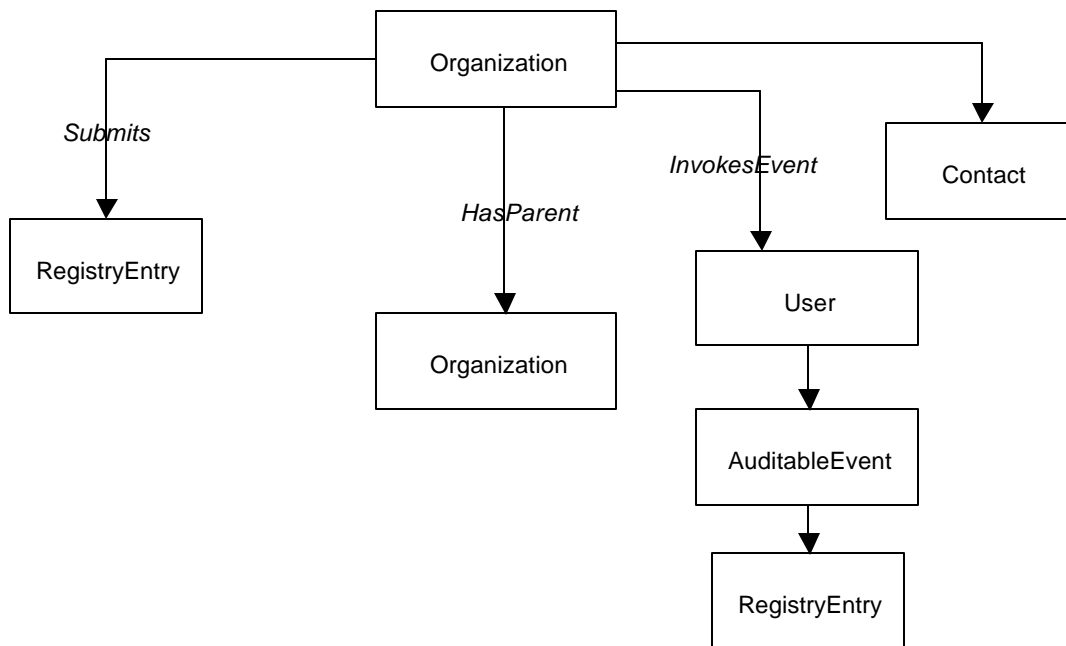
1532 A client application wishes to identify all package instances in the Registry that are  
1533 empty. Since the RegistryPackageQuery is not set up to do negations, clients will have  
1534 to do two separate RegistryPackageQuery requests, one to find all packages and  
1535 another to find all non-empty packages, and then do the set difference themselves.  
1536 Alternatively, they could do a more complex RegistryEntryQuery and check that the  
1537 packaging association between the package and its members is non-existent.

1538 Note: A registry package is an intrinsic RegistryEntry instance that is completely  
1539 determined by its associations with its members. Thus a RegistryPackageQuery can  
1540 always be re-specified as an equivalent RegistryEntryQuery using appropriate "Source"  
1541 and "Target" associations. However, the equivalent RegistryEntryQuery is often more  
1542 complicated to write.

1543

1543 **8.2.6 OrganizationQuery**1544 **Purpose**

1545 To identify a set of organization instances as the result of a query over selected registry  
 1546 metadata.

1547 **ebRIM Binding**

1548

1549 **Definition**

```

1550
1551 <!ELEMENT OrganizationQuery
1552   (   OrganizationFilter?,
1553       SubmitsRegistryEntry*,
1554       HasParentOrganization?,
1555       InvokesEventBranch*,
1556       ContactFilter           )>
1557
1558 <!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>
1559
1560 <!ELEMENT HasParentOrganization
1561   (   OrganizationFilter?,
1562       HasParentOrganization? )>
1563
1564 <!ELEMENT InvokesEventBranch
1565   (   UserFilter?,
1566       AuditableEventFilter?,
1567       RegistryEntryQuery? )>
  
```

1568 **Semantic Rules**

- 1569 1. Let ORG denote the set of all persistent Organization instances in the Registry. The  
1570 following steps will eliminate instances in ORG that do not satisfy the conditions of  
1571 the specified filters.
- 1572 a) If an OrganizationFilter element is not directly contained in the  
1573 OrganizationQuery element, or if ORG is empty, then continue below; otherwise,  
1574 let x be an organization instance in ORG. If x does not satisfy the  
1575 OrganizationFilter as defined in Section 8.2.9, then remove x from RP.
- 1576 b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery,  
1577 or if ORG is empty, then continue below; otherwise, consider each  
1578 SubmitsRegistryEntry element separately as follows:
- 1579 If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element,  
1580 then let RES be the set of all RegistryEntry instances that have been submitted  
1581 to the Registry by organization x; otherwise, let RE be the result of the  
1582 RegistryEntryQuery as defined in Section 8.2.2 and let RES be the set of all  
1583 instances in RE that have been submitted to the Registry by organization x. If  
1584 RES is empty, then remove x from ORG.
- 1585 c) If a HasParentOrganization element is not specified within the  
1586 OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute  
1587 the following paragraph with o = x:
- 1588 Let o be an organization instance. If an OrganizationFilter is not specified within  
1589 the HasParentOrganization and if o has no parent (i.e. if o is a root organization  
1590 in the Organization hierarchy), then remove x from ORG; otherwise, let p be the  
1591 parent organization of o. If p does not satisfy the OrganizationFilter, then remove  
1592 x from ORG.
- 1593 If another HasParentOrganization element is directly contained within this  
1594 HasParentOrganization element, then repeat the previous paragraph with o = p.
- 1595 d) If an InvokesEventBranch element is not specified within the OrganizationQuery,  
1596 or if ORG is empty, then continue below; otherwise, consider each  
1597 InvokesEventBranch element separately as follows:
- 1598 If an UserFilter is not specified, and if x is not the submitting organization of some  
1599 AuditableEvent instance, then remove x from ORG. If an AuditableEventFilter is  
1600 not specified, then let AE be the set of all AuditableEvent instances that have x  
1601 as the submitting organization; otherwise, let AE be the set of AuditableEvent  
1602 instances that satisfy the AuditableEventFilter and have x as the submitting  
1603 organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is  
1604 not specified in the InvokesEventBranch element, then let RES be the set of all  
1605 RegistryEntry instances associated with an event in AE; otherwise, let RE be the  
1606 result set of the RegistryEntryQuery, as specified in Section 8.2.2, and let RES  
1607 be the subset of RE of entries submitted by x. If RES is empty, then remove x  
1608 from ORG.

1609 e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG is  
1610 empty, then continue below; otherwise, consider each ContactFilter separately as  
1611 follows:

1612 Let CT be the set of Contact instances that satisfy the ContactFilter and are the  
1613 contacts for organization x. If CT is empty, then remove x from ORG.

1614 2. If ORG is empty, then raise the warning: *organization query result is empty*.

1615 3. Return ORG as the result of the OrganizationQuery.

1616

## 1617 Examples

1618 A client application wishes to identify a set of organizations, based in France, that have  
1619 submitted a PartyProfile extrinsic object this year.

```
1621 <OrganizationQuery>
1622   <OrganizationFilter>
1623     country EQUAL "France"           -- code by Clause, Section 8.2.10
1624   </OrganizationFilter>
1625   <SubmitsRegistryEntry>
1626     <RegistryEntryQuery>
1627       <RegistryEntryFilter>
1628         objectType EQUAL "CPP"      -- code by Clause, Section 8.2.10
1629       </RegistryEntryFilter>
1630       <HasAuditableEventBranch>
1631         <AuditableEventFilter>
1632           timestamp GE "2001-01-01" -- code by Clause, Section 8.2.10
1633         </AuditableEventFilter>
1634       </HasAuditableEventBranch>
1635     </RegistryEntryQuery>
1636   </SubmitsRegistryEntry>
1637 </OrganizationQuery>
```

1638

1639 A client application wishes to identify all organizations that have XYZ, Corporation as a  
1640 parent. The client knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is  
1641 no guarantee that subsidiaries of XYZ have a URN that uses the same format, so a full  
1642 query is required.

```
1644 <OrganizationQuery>
1645   <HasParentOrganization>
1646     <OrganizationFilter>
1647       id EQUAL "urn:ebxml:org:xyz"  -- code by Clause, Section 8.2.10
1648     </OrganizationFilter>
1649   </HasParentOrganization>
1650 </OrganizationQuery>
```

1651

## 1651 8.2.7 ReturnRegistryEntry

### 1652 Purpose

1653 To construct an XML document that contains selected registry metadata associated with  
 1654 the registry entries identified by a RegistryEntryQuery. NOTE: Initially, the  
 1655 RegistryEntryQuery could be the URN identifier for a single registry entry.

### 1656 Definition

```

1657
1658 <!ELEMENT ReturnRegistryEntry
1659   (
1660     RegistryEntryQuery,
1661     WithClassifications?,
1662     WithSourceAssociations?,
1663     WithTargetAssociations?,
1664     WithAuditableEvents?,
1665     WithExternalLinks?
1666   )>
1667
1668 <!ELEMENT WithClassifications ( ClassificationFilter? )>
1669 <!ELEMENT WithSourceAssociations ( AssociationFilter? )>
1670 <!ELEMENT WithTargetAssociations ( AssociationFilter? )>
1671 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>
1672 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
1673
1674 <!ELEMENT ReturnRegistryEntryResult
1675   (
1676     RegistryEntryMetadata*
1677   )>
1678
1679 <!ELEMENT RegistryEntryMetadata
1680   (
1681     RegistryEntry,
1682     Classification*,
1683     SourceAssociations?,
1684     TargetAssociations?,
1685     AuditableEvent*,
1686     ExternalLink*
1687   )>
1688
1689 <!ELEMENT SourceAssociations ( Association* )>
1690 <!ELEMENT TargetAssociations ( Association* )>
  
```

### 1685 Semantic Rules

- 1686 1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink  
 1687 elements contained in the ReturnRegistryEntryResult are defined by the ebXML  
 1688 Registry DTD specified in Appendix A.
- 1689 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in  
 1690 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let  
 1691 S be the set of warnings and errors returned. If any element in S is an error  
 1692 condition, then stop execution and return the same set of warnings and errors along  
 1693 with the ReturnRegistryEntryResult.



- 1694 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the  
1695 ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*.  
1696 Add this warning to the error list returned by the RegistryEntryQuery and return this  
1697 enhanced error list with the ReturnRegistryEntryResult.
- 1698 4. For each registry entry E referenced by an element of R, use the attributes of E to  
1699 create a new RegistryEntry element as defined in Appendix A. Then create a new  
1700 RegistryEntryMetadata element as defined above to be the parent element of that  
1701 RegistryEntry element.
- 1702 5. If no With option is specified, then the resulting RegistryEntryMetadata element has  
1703 no Classification, SourceAssociations, TargetAssociations, AuditableEvent, or  
1704 ExternalData subelements. The set of RegistryEntryMetadata elements, with the  
1705 Error list from the RegistryEntryQuery, is returned as the ReturnRegistryEntryResult.
- 1706 6. If WithClassifications is specified, then for each E in R do the following: If a  
1707 ClassificationFilter is not present, then let C be any classification instance linked to  
1708 E; otherwise, let C be a classification instance linked to E that satisfies the  
1709 ClassificationFilter (Section 8.2.9). For each such C, create a new Classification  
1710 element as defined in Appendix A. Add these Classification elements to their parent  
1711 RegistryEntryMetadata element.
- 1712 7. If WithSourceAssociations is specified, then for each E in R do the following: If an  
1713 AssociationFilter is not present, then let A be any association instance whose source  
1714 object is E; otherwise, let A be an association instance that satisfies the  
1715 AssociationFilter (Section 8.2.9) and whose source object is E. For each such A,  
1716 create a new Association element as defined in Appendix A. Add these Association  
1717 elements as subelements of the WithSourceAssociations and add that element to its  
1718 parent RegistryEntryMetadata element.
- 1719 8. If WithTargetAssociations is specified, then for each E in R do the following: If an  
1720 AssociationFilter is not present, then let A be any association instance whose target  
1721 object is E; otherwise, let A be an association instance that satisfies the  
1722 AssociationFilter (Section 8.2.9) and whose target object is E. For each such A,  
1723 create a new Association element as defined in Appendix A. Add these Association  
1724 elements as subelements of the WithTargetAssociations and add that element to its  
1725 parent RegistryEntryMetadata element.
- 1726 9. If WithAuditableEvents is specified, then for each E in R do the following: If an  
1727 AuditableEventFilter is not present, then let A be any auditable event instance linked  
1728 to E; otherwise, let A be any auditable event instance linked to E that satisfies the  
1729 AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent  
1730 element as defined in Appendix A. Add these AuditableEvent elements to their  
1731 parent RegistryEntryMetadata element.

- 1732 10. If WithExternalLinks is specified, then for each E in R do the following: If an  
1733 ExternalLinkFilter is not present, then let L be any external link instance linked to E;  
1734 otherwise, let L be any external link instance linked to E that satisfies the  
1735 ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink  
1736 element as defined in Appendix A. Add these ExternalLink elements to their parent  
1737 RegistryEntryMetadata element.
- 1738 11. If any warning or error condition results, then add the code and the message to the  
1739 RegistryResponse element that includes the RegistryEntryQueryResult.
- 1740 12. Return the set of RegistryEntryMetadata elements as the content of the  
1741 ReturnRegistryEntryResult.

1742

### 1743 Examples

1744 A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by  
1745 XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer wishes to  
1746 check on the current status of that DTD, especially if it has been superseded or  
1747 replaced, and get all of its current classifications. The following query request will return  
1748 an XML document with the registry entry for the existing DTD as the root, with all of its  
1749 classifications, and with associations to registry entries for any items that have  
1750 superseded or replaced it.

```
1751
1752 <ReturnRegistryEntry>
1753   <RegistryEntryQuery>
1754     <RegistryEntryFilter>
1755       id EQUAL "urn:com:xyz:po:325"           -- code by Clause, Section 8.2.10
1756     </RegistryEntryFilter>
1757   </RegistryEntryQuery>
1758   <WithClassifications/>
1759   <WithSourceAssociations>
1760     <AssociationFilter>                       -- code by Clause, Section 8.2.10
1761       associationType EQUAL "SupersededBy" OR
1762       associationType EQUAL "ReplacedBy"
1763     </AssociationFilter>
1764   </WithSourceAssociations>
1765 </ReturnRegistryEntry>
```

1766

1767 A client of the Registry registered an XML DTD several years ago and is now thinking of  
1768 replacing it with a revised version. The identifier for the existing DTD is  
1769 "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the  
1770 existing DTD. The client desires a list of all registered items that use the existing DTD  
1771 so they can assess the impact of an incompatible change. The following query returns  
1772 an XML document that is a list of all RegistryEntry elements that represent registered  
1773 items that use, contain, or extend the given DTD. The document also links each  
1774 RegistryEntry element in the list to an element for the identified association.

1775

```

1776
1777 <ReturnRegistryEntry>
1778   <RegistryEntryQuery>
1779     <SourceAssociationBranch>
1780       <AssociationFilter>           -- code by Clause, Section 8.2.10
1781         associationType EQUAL "Contains" OR
1782         associationType EQUAL "Uses" OR
1783         associationType EQUAL "Extends"
1784       </AssociationFilter>
1785     <RegistryEntryFilter>         -- code by Clause, Section 8.2.10
1786       id EQUAL "urn:xyz:dtd:po97"
1787     </RegistryEntryFilter>
1788   </SourceAssociationBranch>
1789 </RegistryEntryQuery>
1790 <WithSourceAssociations>
1791   <AssociationFilter>           -- code by Clause, Section 8.2.10
1792     associationType EQUAL "Contains" OR
1793     associationType EQUAL "Uses" OR
1794     associationType EQUAL "Extends"
1795   </AssociationFilter>
1796 </WithSourceAssociations>
1797 </ReturnRegistryEntry>

```

1798

1799 A user has been browsing the registry and has found a registry entry that describes a  
1800 package of core-components that should solve the user's problem. The package URN  
1801 identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to know what's in the package.  
1802 The following query returns an XML document with a registry entry for each member of  
1803 the package along with that member's Uses and HasMemberBranch associations.

```

1804
1805 <ReturnRegistryEntry>
1806   <RegistryEntryQuery>
1807     <TargetAssociationBranch>
1808       <AssociationFilter>           -- code by Clause, Section 8.2.10
1809         associationType EQUAL "HasMember"
1810       </AssociationFilter>
1811     <RegistryEntryFilter>         -- code by Clause, Section 8.2.10
1812       id EQUAL " urn:com:cc:pkg:ccstuff "
1813     </RegistryEntryFilter>
1814   </TargetAssociationBranch>
1815 </RegistryEntryQuery>
1816 <WithSourceAssociations>
1817   <AssociationFilter>           -- code by Clause, Section 8.2.10
1818     associationType EQUAL "HasMember" OR
1819     associationType EQUAL "Uses"
1820   </AssociationFilter>
1821 </WithSourceAssociations>
1822 </ReturnRegistryEntry>
1823

```

## 1823 8.2.8 ReturnRepositoryItem

### 1824 Purpose

1825 To construct an XML document that contains one or more repository items, and some  
 1826 associated metadata, by submitting a RegistryEntryQuery to the registry/repository that  
 1827 holds the desired objects. NOTE: Initially, the RegistryEntryQuery could be the URN  
 1828 identifier for a single registry entry.

### 1829 Definition

```

1830
1831 <!ELEMENT ReturnRepositoryItem
1832 ( RegistryEntryQuery,
1833   RecursiveAssociationOption?,
1834   WithDescription? )>
1835
1836 <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1837 <!ATTLIST RecursiveAssociationOption
1838   depthLimit CDATA #IMPLIED >
1839
1840 <!ELEMENT AssociationType EMPTY >
1841 <!ATTLIST AssociationType
1842   role CDATA #REQUIRED >
1843
1844 <!ELEMENT WithDescription EMPTY >
1845
1846 <!ELEMENT ReturnRepositoryItemResult
1847 ( RepositoryItem*)>
1848
1849 <!ELEMENT RepositoryItem
1850 ( ClassificationScheme
1851   | RegistryPackage
1852   | ExtrinsicObject
1853   | WithdrawnObject
1854   | ExternalLinkItem )>
1855 <!ATTLIST RepositoryItem
1856   identifier CDATA #REQUIRED
1857   name CDATA #REQUIRED
1858   contentURI CDATA #REQUIRED
1859   objectType CDATA #REQUIRED
1860   status CDATA #REQUIRED
1861   stability CDATA #REQUIRED
1862   description CDATA #IMPLIED >
1863
1864 <!ELEMENT ExtrinsicObject (#PCDATA) >
1865 <!ATTLIST ExtrinsicObject
1866   byteEncoding CDATA "Base64" >
1867
1868 <!ELEMENT WithdrawnObject EMPTY >
1869
1870 <!ELEMENT ExternalLinkItem EMPTY >
1871
1872
```

1873

1874 **Semantic Rules**

- 1875 1. If the RecursiveOption element is not present , then set Limit=0. If the  
1876 RecursiveOption element is present, interpret its depthLimit attribute as an integer  
1877 literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0  
1878 means that no recursion occurs. A Limit of -1 means that recursion occurs  
1879 indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive  
1880 integer, then stop execution and raise the exception: *invalid depth limit*; otherwise,  
1881 set Limit=N, where N is that positive integer. A Limit of N means that exactly N  
1882 recursive steps will be executed unless the process terminates prior to that limit.
- 1883 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned  
1884 as part of the ReturnRepositoryItemResult. Initially Result is empty. Semantic rules  
1885 4 through 10 determine the content of Result.
- 1886 3. If the WithDescription element is present, then set WSD="yes"; otherwise, set  
1887 WSD="no".
- 1888 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in  
1889 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let  
1890 S be the set of warnings and errors returned. If any element in S is an error  
1891 condition, then stop execution and return the same set of warnings and errors along  
1892 with the ReturnRepositoryItemResult.
- 1893 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from  
1894 R. After execution of these rules, if Depth is now equal to Limit, then return the  
1895 content of Result as the set of RepositoryItem elements in the  
1896 ReturnRepositoryItemResult element; otherwise, continue with Semantic Rule 8.
- 1897 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the  
1898 following:
- 1899 a) If E.contentURI references a repository item in this registry/repository, then  
1900 create a new RepositoryItem element, with values for its attributes derived as  
1901 specified in Semantic Rule 7.
- 1902 1) If E.objectType="ClassificationScheme", then put the referenced  
1903 ClassificationScheme DTD as the subelement of this RepositoryItem.  
1904 [NOTE: Requires DTD specification!]
- 1905 2) If E.objectType="RegistryPackage", then put the referenced  
1906 RegistryPackage DTD as the subelement of this RepositoryItem. [NOTE:  
1907 Requires DTD specification!]
- 1908 3) Otherwise, i.e., if the object referenced by E has an unknown internal  
1909 structure, then put the content of the repository item as the #PCDATA of a  
1910 new ExtrinsicObject subelement of this RepositoryItem.

- 1911        b) If E.objectURL references a registered object in some other registry/repository,  
1912        then create a new RepositoryItem element, with values for its attributes derived  
1913        as specified in Semantic Rule 7, and create a new ExternalLink element as the  
1914        subelement of this RepositoryItem.
- 1915        c) If E.objectURL is void, i.e. the object it would have referenced has been  
1916        withdrawn, then create a new RepositoryItem element, with values for its  
1917        attributes derived as specified in Semantic Rule 7, and create a new  
1918        WithdrawnObject element as the subelement of this RepositoryItem.
- 1919 7. Let E be a registry entry and let RO be the RepositoryItem element created in  
1920        Semantic Rule 6. Set the attributes of RO to the values derived from the  
1921        corresponding attributes of E. If WSD="yes", include the value of the description  
1922        attribute; otherwise, do not include it. Insert this new RepositoryItem element into the  
1923        Result set.
- 1924 8. Let R be defined as in Semantic Rule **Error! Reference source not found.**  
1925        Execute Semantic Rule 9 with Y as the set of RegistryEntry instances referenced by  
1926        R. Then continue with Semantic rule 10.
- 1927 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty  
1928        set of RegistryEntry instances. For each registry entry E in Y, and for each  
1929        AssociationType A of the RecursiveAssociationOption, do the following:
- 1930        a) Let Z be the set of target items E' linked to E under association instances having  
1931        E as the source object, E' as the target object, and A as the AssociationType.
- 1932        b) Add the elements of Z to NextLevel.
- 1933 10. Let X be the set of new registry entries that are in NextLevel but are not yet  
1934        represented in the Result set.
- 1935        Case:
- 1936        a) If X is empty, then return the content of Result as the set of RepositoryItem  
1937        elements in the ReturnRepositoryItemResult element.
- 1938        b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set.  
1939        When finished, add the elements of X to Y and set Depth=Depth+1. If Depth is  
1940        now equal to Limit, then return the content of Result as the set of RepositoryItem  
1941        elements in the ReturnRepositoryItemResult element; otherwise, repeat  
1942        Semantic Rules 9 and 10 with the new set Y of registry entries.
- 1943 11. If any exception, warning, or other status condition results during the execution of  
1944        the above, then return appropriate RegistryError elements in the RegistryResult  
1945        associated with the ReturnRepositoryItemResult element created in Semantic Rule 5  
1946        or Semantic Rule 10.

#### 1947 **Examples**

1948 A registry client has found a registry entry for a core-component item. The item's URN  
1949 identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many  
1950 other registered items. The client desires the collection of all items needed for a

1951 complete implementation of "goodthing". The following query returns an XML document  
1952 that is a collection of all needed items.

```
1953
1954 <ReturnRepositoryItem>
1955   <RegistryEntryQuery>
1956     <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1957       id EQUAL "urn:ebxml:cc:goodthing"
1958     </RegistryEntryFilter>
1959   </RegistryEntryQuery>
1960   <RecursiveAssociationOption>
1961     <AssociationType role="Uses" />
1962     <AssociationType role="ValidatesTo" />
1963   </RecursiveAssociationOption>
1964 </ReturnRepositoryItem>
1965
```

1966 A registry client has found a reference to a core-component routine  
1967 ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client knows  
1968 that all routines have a required association to its defining UML specification. The  
1969 following query returns both the routine and its UML specification as a collection of two  
1970 items in a single XML document.

```
1971
1972 <ReturnRepositoryItem>
1973   <RegistryEntryQuery>
1974     <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1975       id EQUAL "urn:ebxml:cc:rtn:nice87"
1976     </RegistryEntryFilter>
1977   </RegistryEntryQuery>
1978   <RecursiveAssociationOption depthLimit="1" >
1979     <AssociationType role="ValidatesTo" />
1980   </RecursiveAssociationOption>
1981 </ReturnRepositoryItem>
1982
```

1983 A user has been told that the 1997 version of the North American Industry Classification  
1984 System (NAICS) is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The  
1985 following query would retrieve the complete classification scheme, with all 1810 nodes,  
1986 as an XML document that validates to a classification scheme DTD.

```
1987
1988 <ReturnRepositoryItem>
1989   <RegistryEntryQuery>
1990     <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1991       id EQUAL "urn:nist:cs:naics-1997"
1992     </RegistryEntryFilter>
1993   </RegistryEntryQuery>
1994 </ReturnRepositoryItem>
```

1995

1996 Note: The ReturnRepositoryItemResult would include a single RepositoryItem that  
1997 consists of a ClassificationScheme document whose content is determined by the URL  
1998 <ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt>.

1999

## 1999 8.2.9 Registry Filters

### 2000 Purpose

2001 To identify a subset of the set of all persistent instances of a given registry class.

### 2002 Definition

2003

2004 <!ELEMENT ObjectFilter ( Clause )>

2005

2006 <!ELEMENT RegistryEntryFilter ( Clause )>

2007

2008 <!ELEMENT IntrinsicObjectFilter ( Clause )>

2009

2010 <!ELEMENT ExtrinsicObjectFilter ( Clause )>

2011

2012 <!ELEMENT PackageFilter ( Clause )>

2013

2014 <!ELEMENT OrganizationFilter ( Clause )>

2015

2016 <!ELEMENT ContactFilter ( Clause )>

2017

2018 <!ELEMENT ClassificationNodeFilter ( Clause )>

2019

2020 <!ELEMENT AssociationFilter ( Clause )>

2021

2022 <!ELEMENT ClassificationFilter ( Clause )>

2023

2024 <!ELEMENT ExternalLinkFilter ( Clause )>

2025

2026 <!ELEMENT ExternalIdentifierFilter ( Clause )>

2027

2028 <!ELEMENT SlotFilter ( Clause )>

2029

2030 <!ELEMENT AuditableEventFilter ( Clause )>

2031

2032 <!ELEMENT UserFilter ( Clause )>

2033

### 2034 Semantic Rules

2035 1. The Clause element is defined in Section 8.2.10, Clause.

2036 2. For every ObjectFilter XML element, the leftArgument attribute of any containing  
2037 SimpleClause shall identify a public attribute of the RegistryObject UML class  
2038 defined in [ebRIM]. If not, raise exception: *object attribute error*. The ObjectFilter  
2039 returns a set of identifiers for RegistryObject instances whose attribute values  
2040 evaluate to *True* for the Clause predicate.

2041 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any  
2042 containing SimpleClause shall identify a public attribute of the RegistryEntry UML  
2043 class defined in [ebRIM].



- 2044 If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a  
2045 set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*  
2046 for the Clause predicate.
- 2047 4. For every IntrinsicObjectFilter XML element, the leftArgument attribute of any  
2048 containing SimpleClause shall identify a public attribute of the IntrinsicObject UML  
2049 class defined in [ebRIM]. If not, raise exception: *intrinsic object attribute error*. The  
2050 IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject instances whose  
2051 attribute values evaluate to *True* for the Clause predicate.
- 2052 5. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any  
2053 containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML  
2054 class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The  
2055 ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose  
2056 attribute values evaluate to *True* for the Clause predicate.
- 2057 6. For every PackageFilter XML element, the leftArgument attribute of any containing  
2058 SimpleClause shall identify a public attribute of the Package UML class defined in  
2059 [ebRIM]. If not, raise exception: *package attribute error*. The PackageFilter returns a  
2060 set of identifiers for Package instances whose attribute values evaluate to *True* for  
2061 the Clause predicate.
- 2062 7. For every OrganizationFilter XML element, the leftArgument attribute of any  
2063 containing SimpleClause shall identify a public attribute of the Organization or  
2064 PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization*  
2065 *attribute error*. The OrganizationFilter returns a set of identifiers for Organization  
2066 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2067 8. For every ContactFilter XML element, the leftArgument attribute of any containing  
2068 SimpleClause shall identify a public attribute of the Contact or PostalAddress UML  
2069 class defined in [ebRIM]. If not, raise exception: *contact attribute error*. The  
2070 ContactFilter returns a set of identifiers for Contact instances whose attribute values  
2071 evaluate to *True* for the Clause predicate.
- 2072 9. For every ClassificationNodeFilter XML element, the leftArgument attribute of any  
2073 containing SimpleClause shall identify a public attribute of the ClassificationNode  
2074 UML class defined in [ebRIM]. If not, raise exception: *classification node attribute*  
2075 *error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode  
2076 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2077 10. For every AssociationFilter XML element, the leftArgument attribute of any  
2078 containing SimpleClause shall identify a public attribute of the Association UML  
2079 class defined in [ebRIM]. If not, raise exception: *association attribute error*. The  
2080 AssociationFilter returns a set of identifiers for Association instances whose attribute  
2081 values evaluate to *True* for the Clause predicate.

- 2082 11. For every ClassificationFilter XML element, the leftArgument attribute of any  
 2083 containing SimpleClause shall identify a public attribute of the Classification UML  
 2084 class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The  
 2085 ClassificationFilter returns a set of identifiers for Classification instances whose  
 2086 attribute values evaluate to *True* for the Clause predicate.
- 2087 12. For every ExternalLinkFilter XML element, the leftArgument attribute of any  
 2088 containing SimpleClause shall identify a public attribute of the ExternalLink UML  
 2089 class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The  
 2090 ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose  
 2091 attribute values evaluate to *True* for the Clause predicate.
- 2092 13. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any  
 2093 containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML  
 2094 class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The  
 2095 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances  
 2096 whose attribute values evaluate to *True* for the Clause predicate.
- 2097 14. For every SlotFilter XML element, the leftArgument attribute of any containing  
 2098 SimpleClause shall identify a public attribute of the Slot UML class defined in  
 2099 [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of  
 2100 identifiers for Slot instances whose attribute values evaluate to *True* for the Clause  
 2101 predicate.
- 2102 15. For every AuditableEventFilter XML element, the leftArgument attribute of any  
 2103 containing SimpleClause shall identify a public attribute of the AuditableEvent UML  
 2104 class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The  
 2105 AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose  
 2106 attribute values evaluate to *True* for the Clause predicate.
- 2107 16. For every UserFilter XML element, the leftArgument attribute of any containing  
 2108 SimpleClause shall identify a public attribute of the User UML class defined in  
 2109 [ebRIM]. If not, raise exception: *auditable identity attribute error*. The UserFilter  
 2110 returns a set of identifiers for User instances whose attribute values evaluate to *True*  
 2111 for the Clause predicate.

2112

2113 **Example**

2114 The following is a complete example of RegistryEntryQuery combined with Clause  
 2115 expansion of RegistryEntryFilter to return a set of RegistryEntry instances whose  
 2116 objectType attribute is "CPP" and whose status attribute is "Approved".

```

2117 <RegistryEntryQuery>
2118   <RegistryEntryFilter>
2119     <Clause>
2120       <CompoundClause   connectivePredicate="And" >
2121         <Clause>
2122           <SimpleClause leftArgument="objectType" >
2123             <StringClause stringPredicate="equal" >CPP</StringClause>
2124

```

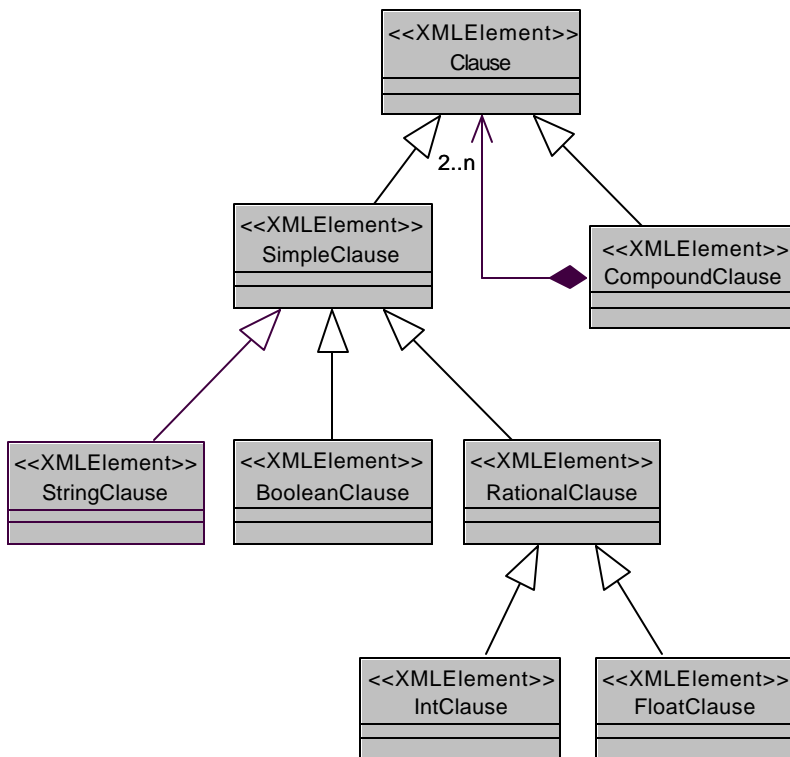
```
2125         </SimpleClause>
2126     </Clause>
2127 <Clause>
2128     <SimpleClause leftArgument="status" >
2129 <StringClause stringPredicate="equal" >Approved</StringClause>
2130     </SimpleClause>
2131 </Clause>
2132 </CompoundClause>
2133 </Clause>
2134 </RegistryEntryFilter>
2135 </RegistryEntryQuery>
2136
2137
```

2137 **8.2.10 XML Clause Constraint Representation**2138 **Purpose**

2139 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate*  
 2140 *Clauses*. Predicate Clauses are utilized to formally define the constraint mechanism,  
 2141 and are referred to simply as **Clauses** in this specification.

2142 **Conceptual UML Diagram**

2143 The following is a conceptual diagram outlining the Clause base structure. It is  
 2144 expressed in UML for visual depiction.



2145

2146

**Figure 19: The Clause base structure**

2147 **Semantic Rules**

2148 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate -  
 2149 RightArgument" format to form a *Clause*. There are two types of Clauses:  
 2150 *SimpleClauses* and *CompoundClauses*.

2151 SimpleClauses

2152 A SimpleClause always defines the leftArgument as a text string, sometimes referred to  
 2153 as the *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be  
 2154 extended. SimpleClause is extended to support BooleanClause, StringClause, and  
 2155 RationalClause (abstract).

2156 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a  
 2157 boolean. StringClause defines the predicate as an enumerated attribute of appropriate  
 2158 string-compare operations and a right argument as the element's text data. Rational  
 2159 number support is provided through a common RationalClause providing an  
 2160 enumeration of appropriate rational number compare operations, which is further  
 2161 extended to IntClause and FloatClause, each with appropriate signatures for the right  
 2162 argument.

### 2163 CompoundClauses

2164 A CompoundClause contains two or more Clauses (Simple or Compound) and a  
 2165 connective predicate. This provides for arbitrarily complex Clauses to be formed.

2166

### 2167 **Definition**

2168

```
2169 <!ELEMENT Clause ( SimpleClause | CompoundClause )>
```

2170

```
2171 <!ELEMENT SimpleClause
```

```
2172 ( BooleanClause | RationalClause | StringClause )>
```

```
2173 <!ATTLIST SimpleClause
```

```
2174 leftArgument CDATA #REQUIRED >
```

2175

```
2176 <!ELEMENT CompoundClause ( Clause, Clause+ )>
```

```
2177 <!ATTLIST CompoundClause
```

```
2178 connectivePredicate ( And | Or ) #REQUIRED>
```

2179

```
2180 <!ELEMENT BooleanClause EMPTY >
```

```
2181 <!ATTLIST BooleanClause
```

```
2182 booleanPredicate ( True | False ) #REQUIRED>
```

2183

```
2184 <!ELEMENT RationalClause ( IntClause | FloatClause )>
```

```
2185 <!ATTLIST RationalClause
```

```
2186 logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
```

2187

```
2188 <!ELEMENT IntClause ( #PCDATA )
```

```
2189 <!ATTLIST IntClause
```

```
2190 e-dtype NMTOKEN #FIXED 'int' >
```

2191

```
2192 <!ELEMENT FloatClause ( #PCDATA )>
```

```
2193 <!ATTLIST FloatClause
```

```
2194 e-dtype NMTOKEN #FIXED 'float' >
```

2195

```
2196 <!ELEMENT StringClause ( #PCDATA )>
```

```
2197 <!ATTLIST StringClause
```

```
2198 stringPredicate
```

```
2199 ( contains | -contains |
```

```
2200 startswith | -startswith |
```

2201           equal | -equal  
2202           endswith | -endswith ) #REQUIRED >

2203

## 2204 **Examples**

### 2205 **Simple BooleanClause: "Smoker" = True**

2206  
2207       <?xml version="1.0" encoding="UTF-8"?>  
2208       <!DOCTYPE Clause SYSTEM "Clause.dtd" >  
2209       <Clause>  
2210         <SimpleClause leftArgument="Smoker">  
2211           <BooleanClause booleanPredicate="True"/>  
2212         </SimpleClause>  
2213       </Clause>  
2214

### 2215 **Simple StringClause: "Smoker" contains "mo"**

2216  
2217       <?xml version="1.0" encoding="UTF-8"?>  
2218       <!DOCTYPE Clause SYSTEM "Clause.dtd" >  
2219       <Clause>  
2220         <SimpleClause leftArgument="Smoker">  
2221           <StringClause stringcomparepredicate="contains">  
2222             mo  
2223           </StringClause>  
2224         </SimpleClause>  
2225       </Clause>

2226

### 2227 **Simple IntClause: "Age" >= 7**

2228  
2229       <?xml version="1.0" encoding="UTF-8"?>  
2230       <!DOCTYPE Clause SYSTEM "Clause.dtd" >  
2231       <Clause>  
2232         <SimpleClause leftArgument="Age">  
2233           <RationalClause logicalPredicate="GE">  
2234             <IntClause e-dtype="int">7</IntClause>  
2235           </RationalClause>  
2236         </SimpleClause>  
2237       </Clause>  
2238

### 2239 **Simple FloatClause: "Size" = 4.3**

2240  
2241       <?xml version="1.0" encoding="UTF-8"?>  
2242       <!DOCTYPE Clause SYSTEM "Clause.dtd" >  
2243       <Clause>  
2244         <SimpleClause leftArgument="Size">  
2245           <RationalClause logicalPredicate="E">  
2246             <FloatClause e-dtype="float">4.3</FloatClause>  
2247         </RationalClause>

```
2248     </SimpleClause>
2249 </Clause>
```

2250

2251 **Compound with two Simples ("Smoker" = False)AND("Age" =< 45))**

```
2252
2253 <?xml version="1.0" encoding="UTF-8"?>
2254 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2255 <Clause>
2256   <CompoundClause connectivePredicate="And">
2257     <Clause>
2258       <SimpleClause leftArgument="Smoker">
2259         <BooleanClause booleanPredicate="False"/>
2260       </SimpleClause>
2261     </Clause>
2262     <Clause>
2263       <SimpleClause leftArgument="Age">
2264         <RationalClause logicalPredicate="EL">
2265           <IntClause e-dtype="int">45</IntClause>
2266         </RationalClause>
2267       </SimpleClause>
2268     </Clause>
2269   </CompoundClause>
2270 </Clause>
```

2271

2272 **Coumpound with one Simple and one Compound**2273 **( ("Smoker" = False)And(("Age" =< 45)Or("American"=True)) )**

```
2274
2275 <?xml version="1.0" encoding="UTF-8"?>
2276 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2277 <Clause>
2278   <CompoundClause connectivePredicate="And">
2279     <Clause>
2280       <SimpleClause leftArgument="Smoker">
2281         <BooleanClause booleanPredicate="False"/>
2282       </SimpleClause>
2283     </Clause>
2284     <Clause>
2285       <CompoundClause connectivePredicate="Or">
2286         <Clause>
2287           <SimpleClause leftArgument="Age">
2288             <RationalClause logicalPredicate="EL">
2289               <IntClause e-dtype="int">45</IntClause>
2290             </RationalClause>
2291           </SimpleClause>
2292         </Clause>
2293         <Clause>
2294           <SimpleClause leftArgument="American">
2295             <BooleanClause booleanPredicate="True"/>
2296           </SimpleClause>
2297         </Clause>
```

```
2298         </CompoundClause>
2299     </Clause>
2300 </CompoundClause>
2301 </Clause>
```

## 2302 **8.3 SQL Query Support**

2303 The Registry may optionally support an SQL based query capability that is designed for  
2304 Registry clients that demand more complex query capability. The optional SQLQuery  
2305 element in the AdhocQueryRequest allows a client to submit complex SQL queries  
2306 using a declarative query language.

2307 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper  
2308 subset of the “SELECT” statement of Entry level SQL defined by ISO/IEC 9075:1992,  
2309 Database Language SQL [SQL], extended to include `<sql invoked routines>`  
2310 (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-  
2311 defined routines defined in template form in Appendix D.3. The exact syntax of the  
2312 Registry query language is defined by the BNF grammar in D.1.

2313 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement  
2314 to use relational databases in a Registry implementation.

### 2315 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

2316 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed  
2317 relational schema defined in Appendix D.3. The relational schema is an algorithmic  
2318 binding to [ebRIM] as described in the following sections.

#### 2319 **8.3.1.1 Interface and Class Binding**

2320 A subset of the Interface and class names defined in [ebRIM] map to table names that  
2321 may be queried by an SQL query. Appendix D.3 defines the names of the ebRIM  
2322 interfaces and classes that may be queried by an SQL query.

2323 The algorithm used to define the binding of [ebRIM] classes to table definitions in  
2324 Appendix D.3 is as follows:

- 2325 • Only those classes and interfaces that have concrete instances are mapped to  
2326 relational tables. This results in intermediate interfaces in the inheritance  
2327 hierarchy, such as RegistryObject and IntrinsicObject, to not map to SQL tables.  
2328 An exception to this rule is RegistryEntry, which is defined next.
- 2329 • A special view called RegistryEntry is defined to allow SQL queries to be made  
2330 against RegistryEntry instances. This is the only interface defined in [ebRIM] that  
2331 does not have concrete instances but is queryable by SQL queries.
- 2332 • The names of relational tables are the same as the corresponding [ebRIM] class  
2333 or interface name. However, the name binding is case insensitive.



2334       • Each [ebRIM] class or interface that maps to a table in Appendix D.3 includes  
2335       column definitions in Appendix D.3 where the column definitions are based on a  
2336       subset of attributes defined for that class or interface in [ebRIM]. The attributes  
2337       that map to columns include the inherited attributes for the [ebRIM] class or  
2338       interface. Comments in Appendix D.3 indicate which ancestor class or interface  
2339       contributed which column definitions.

2340       An SQLException against a table not defined in Appendix D.3 may raise an error condition:  
2341       InvalidQueryException.

2342       The following sections describe the algorithm for mapping attributes of [ebRIM] to  
2343       SQLcolumn definitions.

#### 2344       **8.3.1.2    Accessor Method To Attribute Binding**

2345       Most of the [ebRIM] interfaces methods are simple get methods that map directly to  
2346       attributes. For example the getName method on RegistryObject maps to a name  
2347       attribute of type String. Each get method in [ebRIM] defines the exact attribute name  
2348       that it maps to in the interface definitions in [ebRIM].

#### 2349       **8.3.1.3    Primitive Attributes Binding**

2350       Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the  
2351       same way as column names in SQL. Again the exact attribute names are defined in the  
2352       interface definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is  
2353       case insensitive. It is therefore valid for a query to contain attribute names that do not  
2354       exactly match the case defined in [ebRIM].

#### 2355       **8.3.1.4    Reference Attribute Binding**

2356       A few of the [ebRIM] interface methods return references to instances of interfaces or  
2357       classes defined by [ebRIM]. For example, the getAccessControlPolicy method of the  
2358       RegistryObject class returns a reference to an instance of an AccessControlPolicy  
2359       object.

2360       In such cases the reference maps to the `id` attribute for the referenced object. The  
2361       name of the resulting column is the same as the attribute name in [ebRIM] as defined by  
2362       8.3.1.3. The data type for the column is UUID as defined in Appendix D.3.

2363       When a reference attribute value holds a null reference, it maps to a null value in the  
2364       SQL binding and may be tested with the `<null specification>` as defined by [SQL].

2365       Reference attribute binding is a special case of a primitive attribute mapping.

#### 2366       **8.3.1.5    Complex Attribute Binding**

2367       A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead  
2368       they are of a complex type as defined by an entity class in [ebRIM]. Examples include  
2369       attributes of type TelephoneNumber, Contact, PersonName etc. in interface  
2370       Organization and class Contact.

2371 The SQL query schema algorithmically maps such complex attributes as multiple  
2372 primitive attributes within the parent table. The mapping simply flattens out the entity  
2373 class attributes within the parent table. The attribute name for the flattened attributes  
2374 are composed of a concatenation of attribute names in the reference chain. For example  
2375 Organization has a contact attribute of type Contact. Contact has an address attribute of  
2376 type PostalAddress. PostalAddress has a String attribute named city. This city attribute  
2377 will be named contact\_address\_city.

#### 2378 **8.3.1.6 Collection Attribute Binding**

2379 A few of the [ebRIM] interface methods return a collection of references to instances of  
2380 interfaces or classes defined by [ebRIM]. For example, the getPackages method of the  
2381 ManagedObject class returns a Collection of references to instances of Packages that  
2382 the object is a member of.

2383 Such collection attributes in [ebRIM] classes have been mapped to stored procedures in  
2384 Appendix D.3 such that these stored procedures return a collection of `id` attribute  
2385 values. The returned value of these stored procedures can be treated as the result of a  
2386 table sub-query in SQL.

2387 These stored procedures may be used as the right-hand-side of an SQL IN clause to  
2388 test for membership of an object in such collections of references.

#### 2389 **8.3.2 Semantic Constraints On Query Syntax**

2390 This section defines simplifying constraints on the query syntax that cannot be  
2391 expressed in the BNF for the query syntax. These constraints must be applied in the  
2392 semantic analysis of the query.

- 2393 1. Class names and attribute names must be processed in a case insensitive manner.
- 2394 2. The syntax used for stored procedure invocation must be consistent with the syntax  
2395 of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 2396 3. For this version of the specification, the SQL select column list consists of exactly  
2397 one column, and must always be `t.id`, where `t` is a table reference in the FROM  
2398 clause.

#### 2399 **8.3.3 SQL Query Results**

2400 The results of an SQL query is always an ObjectRefList as defined by the  
2401 AdHocQueryResponse in 8.4. This means the result of an SQL query is always a  
2402 collection of references to instances of a sub-class of the RegistryObject interface in  
2403 [ebRIM]. This is reflected in a semantic constraint that requires that the SQL select  
2404 column specified must always be an `id` column in a table in Appendix D.3 for this  
2405 version of the specification.

### 2406 **8.3.4 Simple Metadata Based Queries**

2407 The simplest form of an SQL query is based upon metadata attributes specified for a  
2408 single class within [ebRIM]. This section gives some examples of simple metadata  
2409 based queries.

2410 For example, to get the collection of ExtrinsicObjects whose name contains the word  
2411 'Acme' and that have a version greater than 1.3, the following query predicates must be  
2412 supported:

```
2413  
2414 SELECT id FROM ExtrinsicObject WHERE name LIKE '%Acme%' AND  
2415     majorVersion >= 1 AND  
2416     (majorVersion >= 2 OR minorVersion > 3);
```

2417 Note that the query syntax allows for conjugation of simpler predicates into more  
2418 complex queries as shown in the simple example above.

### 2419 **8.3.5 RegistryEntry Queries**

2420 Given the central role played by the RegistryEntry interface in ebRIM, the schema for  
2421 the SQL query defines a special view called RegistryEntry that allows doing a  
2422 polymorphic query against all RegistryEntry instances regardless of their actual  
2423 concrete type or table name.

2424 The following example is the same as Section 8.3.4 except that it is applied against all  
2425 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will  
2426 include id for all qualifying RegistryEntry instances whose name contains the word  
2427 'Acme' and that have a version greater than 1.3.

```
2428 SELECT id FROM RegistryEntry WHERE name LIKE '%Acme%' AND  
2429     objectType = 'ExtrinsicObject' AND  
2430     majorVersion >= 1 AND  
2431     (majorVersion >= 2 OR minorVersion > 3);
```

### 2432 **8.3.6 Classification Queries**

2433 This section describes the various classification related queries that must be supported.

#### 2434 **8.3.6.1 Identifying ClassificationNodes**

2435 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they  
2436 may also be identified as a path attribute that specifies an XPATH expression [XPT]  
2437 from a root classification node to the specified classification node in the XML document  
2438 that would represent the ClassificationNode tree including the said ClassificationNode.

#### 2439 **8.3.6.2 Getting Root Classification Nodes**

2440 To get the collection of root ClassificationNodes the following query predicate must be  
2441 supported:

```
2442 SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

2443 The above query returns all ClassificationNodes that have their parent attribute set to  
 2444 null. Note that the above query may also specify a predicate on the name if a specific  
 2445 root ClassificationNode is desired.

### 2446 **8.3.6.3 Getting Children of Specified ClassificationNode**

2447 To get the children of a ClassificationNode given the ID of that node the following style  
 2448 of query must be supported:

```
2449 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

2450 The above query returns all ClassificationNodes that have the node specified by <id> as  
 2451 their parent attribute.

### 2452 **8.3.6.4 Getting Objects Classified By a ClassificationNode**

2453 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the  
 2454 following style of query must be supported:

```
2455 SELECT id FROM ExtrinsicObject
2456 WHERE
2457   id IN (SELECT classifiedObject FROM Classification
2458         WHERE
2459           classificationNode IN (SELECT id FROM ClassificationNode
2460                                WHERE path = '/Geography/Asia/Japan'))
2461 AND
2462   id IN (SELECT classifiedObject FROM Classification
2463         WHERE
2464           classificationNode IN (SELECT id FROM ClassificationNode
2465                                WHERE path = '/Industry/Automotive'))
```

2467 The above query gets the collection of ExtrinsicObjects that are classified by the  
 2468 Automotive Industry and the Japan Geography. Note that according to the semantics  
 2469 defined for GetClassifiedObjectsRequest, the query will also contain any objects that  
 2470 are classified by descendents of the specified ClassificationNodes.

### 2471 **8.3.6.5 Getting ClassificationNodes That Classify an Object**

2472 To get the collection of ClassificationNodes that classify a specified Object the following  
 2473 style of query must be supported:

```
2474 SELECT id FROM ClassificationNode
2475 WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

## 2476 **8.3.7 Association Queries**

2477 This section describes the various Association related queries that must be supported.

### 2478 **8.3.7.1 Getting All Association With Specified Object As Its Source**

2479 To get the collection of Associations that have the specified Object as its source, the  
 2480 following query must be supported:

```
2481 SELECT id FROM Association WHERE sourceObject = <id>
```

### 2482 **8.3.7.2 Getting All Association With Specified Object As Its Target**

2483 To get the collection of Associations that have the specified Object as its target, the  
2484 following query must be supported:

```
2485 SELECT id FROM Association WHERE targetObject = <id>
```

### 2486 **8.3.7.3 Getting Associated Objects Based On Association Attributes**

2487 To get the collection of Associations that have specified Association attributes, the  
2488 following queries must be supported:

2489 Select Associations that have the specified name.

```
2490 SELECT id FROM Association WHERE name = <name>
```

2491 Select Associations that have the specified source role name.

```
2492 SELECT id FROM Association WHERE sourceRole = <roleName>
```

2493 Select Associations that have the specified target role name.

```
2494 SELECT id FROM Association WHERE targetRole = <roleName>
```

2495 Select Associations that have the specified association type, where association type is a  
2496 string containing the corresponding field name described in [ebRIM].

```
2497 SELECT id FROM Association WHERE  
2498 associationType = <associationType>
```

### 2499 **8.3.7.4 Complex Association Queries**

2500 The various forms of Association queries may be combined into complex predicates.  
2501 The following query selects Associations from an object with a specified id, that have  
2502 the sourceRole "buysFrom" and targetRole "sellsTo":

```
2503 SELECT id FROM Association WHERE  
2504 sourceObject = <id> AND  
2505 sourceRole = 'buysFrom' AND  
2506 targetRole = 'sellsTo'
```

### 2507 **8.3.8 Package Queries**

2508 To find all Packages that a specified ExtrinsicObject belongs to, the following query is  
2509 specified:

```
2510 SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

#### 2511 **8.3.8.1 Complex Package Queries**

2512 The following query gets all Packages that a specified object belongs to, that are not  
2513 deprecated and where name contains "RosettaNet."

```
2514 SELECT id FROM Package WHERE  
2515 id IN (RegistryEntry_packages(<id>)) AND  
2516 name LIKE '%RosettaNet%' AND  
2517 status <> 'Deprecated'
```

### 2518 **8.3.9 ExternalLink Queries**

2519 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query  
2520 is specified:

```
2521 SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

2522 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following  
 2523 query is specified:

```
2524 SELECT id From ExtrinsicObject WHERE id IN (RegistryEntry_linkedObjects(<id>))
```

### 2525 8.3.9.1 Complex ExternalLink Queries

2526 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to,  
 2527 that contain the word 'legal' in their description and have a URL for their externalURI.

```
2528 SELECT id FROM ExternalLink WHERE  

  2529 id IN (RegistryEntry_externalLinks(<id>)) AND  

  2530 description LIKE '%legal%' AND  

  2531 externalURI LIKE '%http://%'
```

### 2532 8.3.10 Audit Trail Queries

2533 To get the complete collection of AuditableEvent objects for a specified ManagedObject,  
 2534 the following query is specified:

```
2535 SELECT id FROM AuditableEvent WHERE registryEntry = <id>
```

## 2536 8.4 Ad Hoc Query Request/Response

2537 A client submits an ad hoc query to the ObjectQueryManager by sending an  
 2538 AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a  
 2539 query in one of the supported Registry query mechanisms.

2540 The ObjectQueryManager sends an AdhocQueryResponse either synchronously or  
 2541 asynchronously back to the client. The AdhocQueryResponse returns a collection of  
 2542 objects whose element type is in the set of element types represented by the leaf nodes  
 2543 of the RegistryEntry hierarchy in [ebRIM].

2544



2545

2546

Figure 20: Submit Ad Hoc Query Sequence Diagram



2547 For details on the schema for the business documents shown in this process refer to  
2548 Appendix A.

## 2549 8.5 Content Retrieval

2550 A client retrieves content via the Registry by sending the GetContentRequest to the  
2551 ObjectQueryManager. The GetContentRequest specifies a list of Object references for  
2552 Objects that need to be retrieved. The ObjectQueryManager returns the specified  
2553 content by sending a GetContentResponse message to the ObjectQueryManagerClient  
2554 interface of the client. If there are no errors encountered, the GetContentResponse  
2555 message includes the specified content as additional payloads within the message. In  
2556 addition to the GetContentResponse payload, there is one additional payload for each  
2557 content that was requested. If there are errors encountered, the RegistryResponse  
2558 payload includes an error and there are no additional content specific payloads.

### 2559 8.5.1 Identification Of Content Payloads

2560 Since the GetContentResponse message may include several repository items as  
2561 additional payloads, it is necessary to have a way to identify each payload in the  
2562 message. To facilitate this identification, the Registry must do the following:

- 2563 • Use the ID for each RegistryEntry instance that describes the repository item as  
2564 the DocumentLabel element in the DocumentReference for that object in the  
2565 Manifest element of the ebXMLHeader.

### 2566 8.5.2 GetContentResponse Message Structure

2567 The following message fragment illustrates the structure of the GetContentResponse  
2568 Message that is returning a Collection of CPPs as a result of a GetContentRequest that  
2569 specified the IDs for the requested objects. Note that the ID for each object retrieved in  
2570 the message as additional payloads is used as its DocumentLabel in the Manifest of the  
2571 ebXMLHeader.

```
2572 ...
2573 --PartBoundary
2574 ...
2575 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2576 ...
2577   <eb:Service eb:type="ebXMLRegistry">ObjectManager</eb:Service>
2578   <eb:Action>submitObjects</eb:Action>
2579 ...
2580 </eb:MessageHeader>
2581 ...
2582 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2583   <eb:Reference xlink:href="cid:registryentries@example.com" ...>
2584     <eb:Description xml:lang="en-us">XML instances that are parameters for the particular
2585     Registry Interface / Method. These are RIM structures that don't include repository items, just a
2586     reference - contentURI to them.</eb:Description>
2587   </eb:Reference>
2588   <eb:Reference xlink:href="cid:cpp1@example.com" ...>
```

```

2592     <eb:Description xml:lang="en-us">XML instance of CPP 1. This is a repository
2593 item.</eb:Description>
2594     </eb:Reference>
2595     <eb:Reference xlink:href="cid:cpp2@example.com" ...>
2596     <eb:Description xml:lang="en-us">XML instance of CPP 2. This is a repository
2597 item.</eb:Description>
2598     </eb:Reference>
2599 </eb:Manifest>
2600
2601 --PartBoundary
2602 Content-ID: registryentries@example.com
2603 Content-Type: text/xml
2604 ...
2605 <?xml version="1.0" encoding="UTF-8"?>
2606 <RootElement>
2607 <SubmitObjectsRequest>
2608   <RegistryEntryList>
2609     <ExtrinsicObject ... contentURI="cid:cppl@example.com" .../>
2610     <ExtrinsicObject ... contentURI="cid:cpp2@example.com" .../>
2611   </RegistryEntryList>
2612 </SubmitObjectsRequest>
2613 </RootElement>
2614 --PartBoundary
2615 Content-ID: cppl@example.com
2616 Content-Type: text/xml
2617 ...
2618 <CPP>
2619 ...
2620 </CPP>
2621
2622 --PartBoundary
2623 Content-ID: cpp2@example.com
2624 Content-Type: text/xml
2625 ...
2626 <CPP>
2627 ...
2628 </CPP>
2629
2630 --PartBoundary--
2631

```

2632

## 2633 8.6 Query And Retrieval: Typical Sequence

2634 The following diagram illustrates the use of both browse/drilldown and ad hoc queries  
 2635 followed by a retrieval of content that was selected by the queries.





2636

2637

Figure 21: Typical Query and Retrieval Sequence

2638

## 9 Registry Security

2639

2640

2641

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

2642

2643

2644

2645

2646

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.

## 2647 **9.1 Integrity of Registry Content**

2648 It is assumed that most business registries do not have the resources to validate the  
2649 veracity of the content submitted to them. The minimal integrity that the Registry must  
2650 provide is to ensure that content submitted by a Submitting Organization (SO) is  
2651 maintained in the Registry without any tampering either *en-route* or *within* the Registry.  
2652 Furthermore, the Registry must make it possible to identify the SO for any Registry  
2653 content unambiguously.

### 2654 **9.1.1 Message Payload Signature**

2655 Integrity of Registry content requires that all submitted content must be signed by the  
2656 Registry client as defined by [SEC]. The signature on the submitted content ensures  
2657 that:

- 2658 • The content has not been tampered with en-route or within the Registry.
- 2659 • The content's veracity can be ascertained by its association with a specific  
2660 submitting organization

## 2661 **9.2 Authentication**

2662 The Registry must be able to authenticate the identity of the Principal associated with  
2663 client requests. *Authentication* is required to identify the ownership of content as well as  
2664 to identify what "privileges" a Principal can be assigned with respect to the specific  
2665 objects in the Registry.

2666 The Registry must perform Authentication on a per request basis. From a security point  
2667 of view, all messages are independent and there is no concept of a session  
2668 encompassing multiple messages or conversations. Session support may be added as  
2669 an optimization feature in future versions of this specification.

2670 The Registry must implement a credential-based authentication mechanism based on  
2671 digital certificates and signatures. The Registry uses the certificate DN from the  
2672 signature to authenticate the user.

### 2673 **9.2.1 Message Header Signature**

2674 Message headers may be signed by the sending ebXML Messaging Service as defined  
2675 by [SEC]. Since this specification is not yet finalized, this version does not require that  
2676 the message header be signed. In the absence of a message header signature, the  
2677 payload signature is used to authenticate the identity of the requesting client.

## 2678 9.3 Confidentiality

### 2679 9.3.1 On-the-wire Message Confidentiality

2680 It is suggested but not required that message payloads exchanged between clients and  
2681 the Registry be encrypted during transmission. Payload encryption must abide by any  
2682 restrictions set forth in [SEC].

### 2683 9.3.2 Confidentiality of Registry Content

2684 In the current version of this specification, there are no provisions for confidentiality of  
2685 Registry content. All content submitted to the Registry may be discovered and read by  
2686 *any* client. Therefore, the Registry must be able to decrypt any submitted content after it  
2687 has been received and prior to storing it in its repository. This implies that the Registry  
2688 and the client have an a priori agreement regarding encryption algorithm, key exchange  
2689 agreements, etc. This service is not addressed in this specification.

## 2690 9.4 Authorization

2691 The Registry must provide an authorization mechanism based on the information model  
2692 defined in [ebRIM]. In this version of the specification the authorization mechanism is  
2693 based on a default Access Control Policy defined for a pre-defined set of roles for  
2694 Registry users. Future versions of this specification will allow for custom Access Control  
2695 Policies to be defined by the Submitting Organization.

### 2696 9.4.1 Pre-defined Roles For Registry Users

2697 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.

### 2698 9.4.2 Default Access Control Policies

2699 The Registry must create a default AccessControlPolicy object that grants the default  
2700 permissions to Registry users based upon their assigned role.

2701 The following table defines the Permissions granted by the Registry to the various pre-  
2702 defined roles for Registry users based upon the default AccessControlPolicy.

2703

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).

2704

2705 The following list summarizes the default role-based AccessControlPolicy:

- 2706 • The Registry must implement the default AccessControlPolicy and associate it  
2707 with all Objects in the Registry
- 2708 • Anyone can publish content, but needs to be authenticated
- 2709 • Anyone can access the content without requiring authentication
- 2710 • The ContentOwner has access to all methods for Registry Objects owned by  
2711 them
- 2712 • The RegistryAdministrator has access to all methods on all Registry Objects
- 2713 • Unauthenticated clients can access all read-only (getXXX) methods
- 2714 • At the time of content submission, the Registry must assign the default  
2715 ContentOwner role to the Submitting Organization (SO) as authenticated by the  
2716 credentials in the submission message. In the current version of this  
2717 specification, it will be the DN as identified by the certificate
- 2718 • Clients that browse the Registry need not use certificates. The Registry must  
2719 assign the default RegistryGuest role to such clients.

## 2720 **Appendix A Web Service Architecture**

### 2721 **A.1 WSDL Terminology Primer**

2722 WSDL provides the ability to describe a web service in abstract as well as with concrete  
2723 bindings to specific protocols.

2724 In WSDL an abstract service consists of one or more `port types` or end-points.  
2725 Each port type consists of a collection of `operations`. Each operation is defined in  
2726 terms of `messages` that define what data is exchanged as part of that operation. Each  
2727 message is typically defined in terms of elements within an XML Schema definition.

2728 An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an  
 2729 abstract service is bound to a specific protocol by providing a binding definition for  
 2730 each abstract port type that defines additional protocols specific details.

2731 Finally, a concrete service definition is defined as a collection of ports, where each  
 2732 port is simply adds address information such as a URL for each concrete port.

## 2733 A.2 Registry Service Abstract Specification

```

2734 <?xml version = "1.0" encoding = "UTF-8"?>
2735 <definitions name = "ObjectQueryManager" targetNamespace = "http://www.oasis-
2736 open.org/ebxml/registry/1.1/services/Registry.wsdl" xmlns:tns = "http://www.oasis-
2737 open.org/ebxml/registry/1.1/services/Registry.wsdl" xmlns:xsd1 = "http://www.oasis-
2738 open.org/ebxml/registry/1.1/schemas/Registry.xsd" xmlns:soap =
2739 "http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wSDL = "http://schemas.xmlsoap.org/wsdl/" xmlns: =
2740 "http://schemas.xmlsoap.org/wsdl/" xmlns = "http://schemas.xmlsoap.org/wsdl/">
2741   <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
2742 "http://schemas.xmlsoap.org/wsdl/ file:///c:/jsews/ebxmlrr-spec/misc/schema/wsdl.xsd"-->
2743   <documentation>$Header: /jse/ebxmlrr-spec/misc/services/Registry.wsdl,v 1.7 2001/08/13
2744 01:47:29 najmi Exp $
2745 This is the the normative abstract WSDL service definition for the OASIS ebXML Registry
2746 services.</documentation>
2747   <!--Ensure that namespace matches targetNamespace of file being imported?-->
2748   <import namespace = "http://www.oasis-open.org/ebxml/registry/1.1/schemas/Registry.xsd"
2749 location = "http://www.oasis-open.org/ebxml/registry/1.1/schemas/Registry.xsd"/>
2750   <!-- Commonly re-used Messages -->
2751
2752   <message name = "RegistryResponse">
2753     <part name = "RegistryResponse" element = "xsd1:RegistryResponse"/>
2754     <!--This part is optional and contains the mime/multipart containing content for a
2755 GetContentRequest-->
2756
2757     <part name = "content" type = "xsd:any"/>
2758   </message>
2759   <!-- Messages used by ObjectManager -->
2760
2761   <message name = "GetClassificationTreeRequest">
2762     <part name = "GetClassificationTreeRequest" element =
2763 "xsd1:GetClassificationTreeRequest"/>
2764   </message>
2765   <message name = "GetClassifiedObjectsRequest">
2766     <part name = "GetClassifiedObjectsRequest" element =
2767 "xsd1:GetClassifiedObjectsRequest"/>
2768   </message>
2769   <message name = "GetContentsRequest">
2770     <part name = "GetContentsRequest" element = "xsd1:GetContentsRequest"/>
2771   </message>
2772   <message name = "GetRootClassificationNodesRequest">
2773     <part name = "GetRootClassificationNodesRequest" element =
2774 "xsd1:GetRootClassificationNodesRequest"/>
2775   </message>
2776   <message name = "SubmitAdhocQueryRequest">
2777     <part name = "SubmitAdhocQueryRequest" element = "xsd1:SubmitAdhocQueryRequest"/>
2778   </message>
2779   <!-- Messages used by ObjectManager -->
2780   <message name = "AddSlotsRequest">
2781     <part name = "AddSlotsRequest" element = "xsd1:AddSlotsRequest"/>
2782   </message>
2783   <message name = "ApproveObjectsRequest">
2784     <part name = "ApproveObjectsRequest" element = "xsd1:ApproveObjectsRequest"/>
2785   </message>
2786   <message name = "DeprecateObjectsRequest">
2787     <part name = "DeprecateObjectsRequest" element = "xsd1:DeprecateObjectsRequest"/>
2788   </message>
2789   <message name = "RemoveObjectsRequest">

```

```

2790         <part name = "RemoveObjectsRequest" element = "xsd1:RemoveObjectsRequest" />
2791     </message>
2792     <message name = "RemoveSlotsRequest">
2793         <part name = "RemoveSlotsRequest" element = "xsd1:RemoveSlotsRequest" />
2794     </message>
2795     <message name = "SubmitObjectsRequest">
2796         <part name = "SubmitObjectsRequest" element = "xsd1:SubmitObjectsRequest" />
2797         <!--This part is the mime/multipart containing content-->
2798         <part name = "content" type = "xsd:any" />
2799     </message>
2800     <portType name = "ObjectQueryManagerPortType">
2801         <documentation>Maps to the ObjectQueryManager interface of Registry Services
2802 spec.</documentation>
2803         <operation name = "getClassificationTree">
2804             <input message = "tns:GetClassificationTreeRequest" />
2805             <output message = "tns:RegistryResponse" />
2806         </operation>
2807         <operation name = "getClassifiedObjects">
2808             <input message = "tns:GetClassifiedObjectsRequest" />
2809             <output message = "tns:RegistryResponse" />
2810         </operation>
2811         <operation name = "getContents">
2812             <input message = "tns:GetContentsRequest" />
2813             <output message = "tns:RegistryResponse" />
2814         </operation>
2815         <operation name = "getRootClassificationNodes">
2816             <input message = "tns:GetRootClassificationNodesRequest" />
2817             <output message = "tns:RegistryResponse" />
2818         </operation>
2819         <operation name = "submitAdhocQuery">
2820             <input message = "tns:SubmitAdhocQueryRequest" />
2821             <output message = "tns:RegistryResponse" />
2822         </operation>
2823     </portType>
2824     <portType name = "ObjectManagerPortType">
2825         <documentation>Maps to the ObjectManager interface of Registry Services
2826 spec.</documentation>
2827         <operation name = "addSlots">
2828             <input message = "tns:AddSlotsRequest" />
2829             <output message = "tns:RegistryResponse" />
2830         </operation>
2831         <operation name = "approveObjectsRequest">
2832             <input message = "tns:ApproveObjectsRequest" />
2833             <output message = "tns:RegistryResponse" />
2834         </operation>
2835         <operation name = "deprecateObjectsRequest">
2836             <input message = "tns:DeprecateObjectsRequest" />
2837             <output message = "tns:RegistryResponse" />
2838         </operation>
2839         <operation name = "removeObjectsRequest">
2840             <input message = "tns:RemoveObjectsRequest" />
2841             <output message = "tns:RegistryResponse" />
2842         </operation>
2843         <operation name = "removeSlotsRequest">
2844             <input message = "tns:RemoveSlotsRequest" />
2845             <output message = "tns:RegistryResponse" />
2846         </operation>
2847         <operation name = "submitObjectsRequest">
2848             <input message = "tns:SubmitObjectsRequest" />
2849             <output message = "tns:RegistryResponse" />
2850         </operation>
2851     </portType>
2852 </definitions>

```

### 2853 A.3 Registry Service SOAP Binding

```

2854 <?xml version = "1.0" encoding = "UTF-8"?>
2855 <definitions name = "RegistryServiceSOAPBinding" targetNamespace = "http://www.oasis-

```

```

2856 open.org/ebxml/registry/1.1/services/RegistrySOAPBinding.wsdl" xmlns:tns = "http://www.oasis-
2857 open.org/ebxml/registry/1.1/services/RegistrySOAPBinding.wsdl" xmlns:registry =
2858 "http://www.oasis-open.org/ebxml/registry/1.1/services/Registry.wsdl" xmlns:soap =
2859 "http://schemas.xmlsoap.org/wsdl/soap/" xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/" xmlns: =
2860 "http://schemas.xmlsoap.org/wsdl/" xmlns:mime = "http://schemas.xmlsoap.org/wsdl/mime/" xmlns =
2861 "http://schemas.xmlsoap.org/wsdl/">
2862 <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
2863 "http://schemas.xmlsoap.org/wsdl/ file:///C:/jsews/ebxmlrr-spec/misc/schema/wsdl.xsd"-->
2864 <documentation>$Header: /jse/ebxmlrr-spec/misc/services/RegistrySOAPBinding.wsdl,v 1.8
2865 2001/08/13 01:47:29 najmi Exp $
2866 This is the the normative concrete SOAP/HTTP binding for the OASIS ebXML Registry
2867 services.</documentation>
2868 <!--Import the definition of the abstract OASIS ebXML Registry services-->
2869 <import namespace = "http://www.oasis-open.org/ebxml/registry/1.1/services/Registry.wsdl"
2870 location = "http://www.oasis-open.org/ebxml/registry/1.1/services/Registry.wsdl"/>
2871 <!--The SOAP bindings to the abstract services follow-->
2872 <binding name = "ObjectQueryManagerSoapBinding" type = "tns:ObjectQueryManagerPortType">
2873 <!--
2874 transport attribute below specifies use of http transport for SOAP binding.
2875 Currently this is the only normative definition of transport for SOAP binding.
2876 -->
2877
2878 <soap:binding style = "document" transport =
2879 "http://schemas.xmlsoap.org/soap/http"/>
2880 <operation name = "getClassificationTree">
2881 <soap:operation soapAction =
2882 "uri:oasis:ebxml:registry:services:ObjectQueryManager:getClassificationTree"/>
2883 <input>
2884 <soap:body use = "literal"/>
2885 </input>
2886 <output>
2887 <soap:body use = "literal"/>
2888 </output>
2889 </operation>
2890 <operation name = "getClassifiedObjects">
2891 <soap:operation soapAction =
2892 "uri:oasis:ebxml:registry:services:ObjectQueryManager:getClassifiedObjects"/>
2893 <input>
2894 <soap:body use = "literal"/>
2895 </input>
2896 <output>
2897 <soap:body use = "literal"/>
2898 </output>
2899 </operation>
2900 <operation name = "getContent">
2901 <soap:operation soapAction =
2902 "uri:oasis:ebxml:registry:services:ObjectQueryManager:getContent"/>
2903 <input>
2904 <soap:body use = "literal"/>
2905 </input>
2906 <output>
2907 <mime:multipartRelated>
2908 <mime:part>
2909 <soap:body parts = "RegistryResponse" use =
2910 "literal"/>
2911 </mime:part>
2912 <mime:part>
2913 <mime:content part = "content" type = "*/*/>
2914 </mime:part>
2915 </mime:multipartRelated>
2916 </output>
2917 </operation>
2918 <operation name = "getRootClassificationNodes">
2919 <soap:operation soapAction =
2920 "uri:oasis:ebxml:registry:services:ObjectQueryManager:getRootClassificationNodes"/>
2921 <input>
2922 <soap:body use = "literal"/>
2923 </input>
2924 <output>
2925

```

```

2926         <soap:body use = "literal"/>
2927     </output>
2928 </operation>
2929 <operation name = "submitAdhocQuery">
2930     <soap:operation soapAction =
2931 "uri:oasis:ebxml:registry:services:ObjectQueryManager:submitAdhocQueries"/>
2932     <input>
2933         <soap:body use = "literal"/>
2934     </input>
2935 <output>
2936     <soap:body use = "literal"/>
2937 </output>
2938 </operation>
2939 </binding>
2940 <binding name = "ObjectManagerSoapBinding" type = "tns:ObjectManagerPortType">
2941     <soap:binding style = "document" transport =
2942 "http://schemas.xmlsoap.org/soap/http"/>
2943     <operation name = "addSlots">
2944         <soap:operation soapAction =
2945 "uri:oasis:ebxml:registry:services:ObjectManager:addSlots"/>
2946         <input>
2947             <soap:body use = "literal"/>
2948         </input>
2949         <output>
2950             <soap:body use = "literal"/>
2951         </output>
2952     </operation>
2953     <operation name = "approveObjects">
2954         <soap:operation soapAction =
2955 "uri:oasis:ebxml:registry:services:ObjectManager:approveObjects"/>
2956         <input>
2957             <soap:body use = "literal"/>
2958         </input>
2959         <output>
2960             <soap:body use = "literal"/>
2961         </output>
2962     </operation>
2963     <operation name = "deprecateObjects">
2964         <!--Need undeprecateObjects?!-->
2965         <soap:operation soapAction =
2966 "uri:oasis:ebxml:registry:services:ObjectManager:deprecateObjects"/>
2967         <input>
2968             <soap:body use = "literal"/>
2969         </input>
2970         <output>
2971             <soap:body use = "literal"/>
2972         </output>
2973     </operation>
2974 </binding>
2975 <operation name = "removeObjects">
2976     <soap:operation soapAction =
2977 "uri:oasis:ebxml:registry:services:ObjectQueryManager:removeObjects"/>
2978     <input>
2979         <soap:body use = "literal"/>
2980     </input>
2981     <output>
2982         <soap:body use = "literal"/>
2983     </output>
2984 </operation>
2985 <operation name = "removeSlots">
2986     <soap:operation soapAction =
2987 "uri:oasis:ebxml:registry:services:ObjectQueryManager:removeSlots"/>
2988     <input>
2989         <soap:body use = "literal"/>
2990     </input>
2991     <output>
2992         <soap:body use = "literal"/>
2993     </output>
2994 </operation>
2995 </operation>

```



```

2996         <operation name = "submitObjects">
2997             <soap:operation soapAction =
2998 "uri:oasis:ebxml:registry:services:ObjectQueryManager:submitObjects"/>
2999             <input>
3000                 <mime:multipartRelated>
3001                     <mime:part>
3002                         <soap:body parts = "SubmitObjectsRequest" use =
3003 "literal"/>
3004                     </mime:part>
3005                     <mime:part>
3006                         <mime:content part = "content" type = "*/"/>
3007                     </mime:part>
3008                 </mime:multipartRelated>
3009             </input>
3010             <output>
3011                 <soap:body use = "literal"/>
3012             </output>
3013         </operation>
3014
3015     </binding>
3016     <!--The concrete services bound to the SOAP binding follows-->
3017
3018     <service name = "RegistryService">
3019         <documentation>The ObjectQueryManager service of OASIS ebXML registry version
3020 1.1</documentation>
3021         <port name = "ObjectQueryManager" binding = "tns:ObjectQueryManagerSOAPBinding">
3022             <soap:address location = "http://your_URL_to_your_ObjectQueryManager"/>
3023         </port>
3024         <port name = "ObjectManager" binding = "tns:ObjectManagerSOAPBinding">
3025             <soap:address location = "http://your_URL_to_your_ObjectQueryManager"/>
3026         </port>
3027     </service>
3028 </definitions>

```

## 3029 **Appendix B ebXML Registry DTD Definition**

3030 The following is the definition for the various ebXML Message payloads described in  
3031 this document.

```

3032 <?xml version="1.0" encoding="UTF-8"?>
3033 <!-- Begin information model mapping. -->
3034
3035 <!--
3036 ObjectAttributes are attributes from the RegistryObject interface in ebRIM.
3037
3038 id may be empty. If specified it may be in urn:uuid format or be in some
3039 arbitrary format. If id is empty registry must generate globally unique id.
3040
3041 If id is provided and in proper UUID syntax (starts with urn:uuid:)
3042 registry will honour it.
3043
3044 If id is provided and is not in proper UUID syntax then it is used for
3045 linkage within document and is ignored by the registry. In this case the
3046 registry generates a UUID for id attribute.
3047
3048 id must not be null when object is being retrieved from the registry.
3049 -->
3050 <!ENTITY % ObjectAttributes "
3051     id          ID          #IMPLIED
3052     name        CDATA      #IMPLIED
3053 
```

```
3054     description CDATA #IMPLIED
3055 ">
3056
3057 <!--
3058 Use as a proxy for an Object that is in the registry already.
3059 Specifies the id attribute of the object in the registry as its id attribute.
3060 id attribute in ObjectAttributes is exactly the same syntax and semantics as
3061 id attribute in RegistryObject.
3062 -->
3063 <!ELEMENT ObjectRef EMPTY>
3064 <!ATTLIST ObjectRef
3065     id ID #IMPLIED
3066 >
3067
3068 <!ELEMENT ObjectRefList (ObjectRef)*>
3069
3070 <!--
3071 RegistryEntryAttributes are attributes from the RegistryEntry interface
3072 in ebRIM.
3073 It inherits ObjectAttributes
3074 -->
3075 <!ENTITY % RegistryEntryAttributes " %ObjectAttributes;
3076     majorVersion      CDATA '1'
3077     minorVersion      CDATA '0'
3078     status             CDATA #IMPLIED
3079     userVersion       CDATA #IMPLIED
3080     stability          CDATA 'Dynamic'
3081     expirationDate    CDATA #IMPLIED">
3082
3083 <!ELEMENT RegistryEntry (SlotList?)>
3084 <!ATTLIST RegistryEntry
3085     %RegistryEntryAttributes; >
3086 <!ELEMENT Value (#PCDATA)>
3087 <!ELEMENT ValueList (Value*)>
3088 <!ELEMENT Slot (ValueList?)>
3089 <!ATTLIST Slot
3090     name CDATA #REQUIRED
3091     slotType CDATA #IMPLIED
3092 >
3093 <!ELEMENT SlotList (Slot*)>
3094
3095 <!--
3096 ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.
3097 It inherits RegistryEntryAttributes
3098 -->
3099
3100
3101 <!ELEMENT ExtrinsicObject EMPTY >
3102 <!ATTLIST ExtrinsicObject
3103     %RegistryEntryAttributes;
3104     contentURI CDATA #REQUIRED
3105     mimeType CDATA #IMPLIED
3106     objectType CDATA #REQUIRED
3107     opaque (true | false) "false"
3108 >
3109
```

```
3110
3111 <!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
3112
3113 <!-- Leaf classes that reflect the concrete classes in ebRIM -->
3114 <!ELEMENT RegistryEntryList
3115 (Association | Classification | ClassificationNode | Package |
3116 ExternalLink | ExternalIdentifier | Organization |
3117 ExtrinsicObject | ObjectRef)*>
3118
3119 <!--
3120 An ExternalLink specifies a link from a RegistryEntry and an external URI
3121 -->
3122 <!ELEMENT ExternalLink EMPTY>
3123 <!ATTLIST ExternalLink
3124     %IntrinsicObjectAttributes;
3125     externalURI CDATA #IMPLIED
3126 >
3127
3128 <!--
3129 An ExternalIdentifier provides an identifier for a RegistryEntry
3130
3131 The value is the value of the identifier (e.g. the social security number)
3132 -->
3133 <!ELEMENT ExternalIdentifier EMPTY>
3134 <!ATTLIST ExternalIdentifier
3135     %IntrinsicObjectAttributes;
3136     value CDATA #REQUIRED
3137 >
3138
3139 <!--
3140 An Association specifies references to two previously submitted
3141 registry entrys.
3142
3143 The sourceObject is id of the sourceObject in association
3144 The targetObject is id of the targetObject in association
3145 -->
3146 <!ELEMENT Association EMPTY>
3147 <!ATTLIST Association
3148     %IntrinsicObjectAttributes;
3149     sourceRole CDATA #IMPLIED
3150     targetRole CDATA #IMPLIED
3151     associationType CDATA #REQUIRED
3152     bidirection (true | false) "false"
3153     sourceObject IDREF #REQUIRED
3154     targetObject IDREF #REQUIRED
3155 >
3156
3157 <!--
3158 A Classification specifies references to two registry entrys.
3159
3160 The classifiedObject is id of the Object being classified.
3161 The classificationNode is id of the ClassificationNode classyng the object
3162 -->
3163 <!ELEMENT Classification EMPTY>
3164 <!ATTLIST Classification
3165     %IntrinsicObjectAttributes;
```

```
3166         classifiedObject IDREF #REQUIRED
3167         classificationNode IDREF #REQUIRED
3168     >
3169
3170     <!--
3171     A Package is a named collection of objects.
3172     -->
3173     <!ELEMENT Package EMPTY>
3174     <!ATTLIST Package
3175         %IntrinsicObjectAttributes;
3176     >
3177
3178     <!-- Attributes inherited by various types of telephone number elements -->
3179     <!ENTITY % TelephoneNumberAttributes " areaCode CDATA #REQUIRED
3180         contryCode CDATA #REQUIRED
3181         extension CDATA #IMPLIED
3182         number CDATA #REQUIRED
3183         url CDATA #IMPLIED">
3184     <!ELEMENT TelephoneNumber EMPTY>
3185     <!ATTLIST TelephoneNumber
3186         %TelephoneNumberAttributes;
3187     >
3188     <!ELEMENT FaxNumber EMPTY>
3189     <!ATTLIST FaxNumber
3190         %TelephoneNumberAttributes;
3191     >
3192
3193     <!ELEMENT PagerNumber EMPTY>
3194     <!ATTLIST PagerNumber
3195         %TelephoneNumberAttributes;
3196     >
3197
3198     <!ELEMENT MobileTelephoneNumber EMPTY>
3199     <!ATTLIST MobileTelephoneNumber
3200         %TelephoneNumberAttributes;
3201     >
3202     <!-- PostalAddress -->
3203     <!ELEMENT PostalAddress EMPTY>
3204     <!ATTLIST PostalAddress
3205         city CDATA #REQUIRED
3206         country CDATA #REQUIRED
3207         postalCode CDATA #REQUIRED
3208         state CDATA #IMPLIED
3209         street CDATA #REQUIRED
3210     >
3211     <!-- PersonName -->
3212     <!ELEMENT PersonName EMPTY>
3213     <!ATTLIST PersonName
3214         firstName CDATA #REQUIRED
3215         middleName CDATA #IMPLIED
3216         lastName CDATA #REQUIRED
3217     >
3218
3219     <!-- Organization -->
3220     <!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
3221     <!ATTLIST Organization
```

```
3222     %IntrinsicObjectAttributes;
3223     parent IDREF #IMPLIED
3224     primaryContact IDREF #REQUIRED
3225 >
3226
3227 <!ELEMENT User (PersonName, PostalAddress, TelephoneNumber,
3228               MobileTelephoneNumber?,
3229               FaxNumber?, PagerNumber?)>
3230 <!ATTLIST User
3231     %ObjectAttributes;
3232     organization IDREF #IMPLIED
3233     email CDATA #IMPLIED
3234     url CDATA #IMPLIED
3235 >
3236
3237 <!ELEMENT AuditableEvent EMPTY>
3238 <!ATTLIST AuditableEvent
3239     %ObjectAttributes;
3240     eventType CDATA #REQUIRED
3241     registryEntry IDREF #REQUIRED
3242     timestamp CDATA #REQUIRED
3243     user IDREF #REQUIRED
3244 >
3245
3246 <!--
3247 ClassificationNode is used to submit a Classification tree to the Registry.
3248
3249 parent is the id to the parent node. code is an optional code value for a
3250                               ClassificationNode
3251 often defined by an external taxonomy (e.g. NAICS)
3252 -->
3253 <!ELEMENT ClassificationNode EMPTY>
3254 <!ATTLIST ClassificationNode
3255     %IntrinsicObjectAttributes;
3256     parent IDREF #IMPLIED
3257     code CDATA #IMPLIED
3258 >
3259
3260 <!--
3261 End information model mapping.
3262
3263 Begin Registry Services Interface
3264
3265 <!ELEMENT RequestAcceptedResponse EMPTY>
3266 <!ATTLIST RequestAcceptedResponse
3267     xml:lang NMTOKEN #REQUIRED
3268 >
3269 <!--
3270
```

3271 The SubmitObjectsRequest allows one to submit a list of RegistryEntry  
3272 elements. Each RegistryEntry element provides metadata for a single submitted  
3273 object. Note that the repository item being submitted is in a separate  
3274 document that is not in this DTD. The ebXML Messaging Services Specification  
3275 defines packaging, for submission, of the metadata of a repository item with  
3276 the repository item itself. The value of the contentURI attribute of the  
3277 ExtrinsicObject element must be the same as the xlink:href attribute within  
3278 the Reference element within the Manifest element of the MessageHeader.  
3279 -->  
3280 <!ELEMENT SubmitObjectsRequest (RegistryEntryList)>  
3281 <!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>  
3282 <!-- Only need name in Slot within SlotList -->  
3283 <!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>  
3284 <!--  
3285 The ObjectRefList is the list of  
3286 refs to the registry entrys being approved.  
3287 -->  
3288 <!ELEMENT ApproveObjectsRequest (ObjectRefList)>  
3289 <!--  
3290 The ObjectRefList is the list of  
3291 refs to the registry entrys being deprecated.  
3292 -->  
3293 <!ELEMENT DeprecateObjectsRequest (ObjectRefList)>  
3294 <!--  
3295 The ObjectRefList is the list of  
3296 refs to the registry entrys being removed  
3297 -->  
3298 <!ELEMENT RemoveObjectsRequest (ObjectRefList)>  
3299 <!ATTLIST RemoveObjectsRequest  
3300       deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"  
3301 >  
3302 <!ELEMENT GetRootClassificationNodesRequest EMPTY>  
3303 <!--  
3304 The namePattern follows SQL-92 syntax for the pattern specified in  
3305 LIKE clause. It allows for selecting only those root nodes that match  
3306 the namePattern. The default value of '\*' matches all root nodes.  
3307 -->  
3308 <!ATTLIST GetRootClassificationNodesRequest  
3309       namePattern CDATA "\*" >  
3310 >  
3311 <!--  
3312 The response includes one or more ClassificationNodes  
3313 -->  
3314 <!ELEMENT GetRootClassificationNodesResponse ( ClassificationNode+ )>  
3315 <!--  
3316 Get the classification tree under the ClassificationNode specified parentRef.  
3317  
3318 If depth is 1 just fetch immediate child  
3319 nodes, otherwise fetch the descendant tree upto the specified depth level.  
3320 If depth is 0 that implies fetch entire sub-tree  
3321 -->  
3322 <!ELEMENT GetClassificationTreeRequest EMPTY>  
3323 <!ATTLIST GetClassificationTreeRequest  
3324       parent CDATA #REQUIRED  
3325       depth CDATA "1" >  
3326 >

```
3327 <!--
3328 The response includes one or more ClassificationNodes which includes only
3329 immediate ClassificationNode children nodes if depth attribute in
3330 GetClassificationTreeRequest was 1, otherwise the decendent nodes
3331 upto specified depth level are returned.
3332 -->
3333 <!ELEMENT GetClassificationTreeResponse ( ClassificationNode+ )>
3334 <!--
3335 Get refs to all registry entrys that are classified by all the
3336 ClassificationNodes specified by ObjectRefList.
3337 Note this is an implicit logical AND operation
3338 -->
3339 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
3340 <!--
3341 objectType attribute can specify the type of objects that the registry
3342 client is interested in, that is classified by this ClassificationNode.
3343 It is a String that matches a choice in the type attribute of
3344                               ExtrinsicObject.
3345 The default value of '*' implies that client is interested in all types
3346 of registry entrys that are classified by the specified ClassificationNode.
3347 -->
3348 <!--
3349 The response includes a RegistryEntryList which has zero or more
3350 RegistryEntrys that are classified by the ClassificationNodes
3351 specified in the ObjectRefList in GetClassifiedObjectsRequest.
3352 -->
3353 <!ELEMENT GetClassifiedObjectsResponse ( RegistryEntryList )>
3354 <!--
3355 An Ad hoc query request specifies a query string as defined by [RS] in the
3356                               queryString attribute
3357 -->
3358 <!ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |
3359                               ReturnRepositoryItem | SQLQuery)>
3360 <!ELEMENT SQLQuery (#PCDATA)>
3361 <!--
3362 The response includes a RegistryEntryList which has zero or more
3363 RegistryEntrys that match the query specified in AdhocQueryRequest.
3364 -->
3365 <!ELEMENT AdhocQueryResponse
3366 ( RegistryEntryList |
3367   FilterQueryResult |
3368   ReturnRegistryEntryResult |
3369   ReturnRepositoryItemResult )>
3370 <!--
3371 Gets the actual content (not metadata) specified by the ObjectRefList
3372 -->
3373 <!ELEMENT GetContentRequest (ObjectRefList)>
3374 <!--
3375 The GetObjectsResponse will have no sub-elements if there were no errors.
3376 The actual contents will be in the other payloads of the message.
3377 -->
3378 <!ELEMENT GetContentResponse EMPTY >
3379 <!--
3380 Describes the capability profile for the registry and what optional features
3381 are supported
3382 -->
```

```
3383 <!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
3384 <!ATTLIST RegistryProfile
3385     version CDATA #REQUIRED
3386 >
3387
3388 <!ELEMENT OptionalFeaturesSupported EMPTY>
3389 <!ATTLIST OptionalFeaturesSupported
3390     sqlQuery (true | false) "false"
3391     xQuery (true | false) "false"
3392 >
3393 <!-- Begin FilterQuery DTD -->
3394 <!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
3395     ClassificationNodeQuery |
3396     RegistryPackageQuery |
3397     OrganizationQuery)>
3398 <!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
3399     AuditableEventQueryResult |
3400     ClassificationNodeQueryResult |
3401     RegistryPackageQueryResult |
3402     OrganizationQueryResult)>
3403 <!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
3404 <!ELEMENT RegistryEntryView EMPTY>
3405 <!ATTLIST RegistryEntryView
3406     objectURN CDATA #REQUIRED
3407     contentURI CDATA #IMPLIED
3408     objectID CDATA #IMPLIED
3409 >
3410 <!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
3411 <!ELEMENT AuditableEventView EMPTY>
3412 <!ATTLIST AuditableEventView
3413     objectID CDATA #REQUIRED
3414     timestamp CDATA #REQUIRED
3415 >
3416 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
3417 <!ELEMENT ClassificationNodeView EMPTY>
3418 <!ATTLIST ClassificationNodeView
3419     objectURN CDATA #REQUIRED
3420     contentURI CDATA #IMPLIED
3421     objectID CDATA #IMPLIED
3422 >
3423 <!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
3424 <!ELEMENT RegistryPackageView EMPTY>
3425 <!ATTLIST RegistryPackageView
3426     objectURN CDATA #REQUIRED
3427     contentURI CDATA #IMPLIED
3428     objectID CDATA #IMPLIED
3429 >
3430 <!ELEMENT OrganizationQueryResult (OrganizationView*)>
3431 <!ELEMENT OrganizationView EMPTY>
3432 <!ATTLIST OrganizationView
3433     orgURN CDATA #REQUIRED
3434     objectID CDATA #IMPLIED
3435 >
3436
3437 <!ELEMENT RegistryEntryQuery
3438     ( RegistryEntryFilter?,
```



```
3439     SourceAssociationBranch*,
3440     TargetAssociationBranch*,
3441     HasClassificationBranch*,
3442     SubmittingOrganizationBranch?,
3443     ResponsibleOrganizationBranch?,
3444     ExternalIdentifierFilter*,
3445     ExternalLinkFilter*,
3446     SlotFilter*,
3447     HasAuditableEventBranch*           )>
3448
3449 <!ELEMENT SourceAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3450 <!ELEMENT TargetAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3451 <!ELEMENT HasClassificationBranch (ClassificationFilter?,
3452                                   ClassificationNodeFilter?)>
3453 <!ELEMENT SubmittingOrganizationBranch (OrganizationFilter?, ContactFilter?)>
3454 <!ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
3455                                         ContactFilter?)>
3456 <!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
3457                                    OrganizationFilter?)>
3458 <!ELEMENT AuditableEventQuery
3459   (AuditableEventFilter?, RegistryEntryQuery*, InvokedByBranch? )>
3460
3461 <!ELEMENT InvokedByBranch
3462   ( UserFilter?, OrganizationQuery? )>
3463
3464 <!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
3465                                   PermitsClassificationBranch*,
3466                                   HasParentNode?, HasSubnode*)>
3467 <!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
3468                                       RegistryEntryQuery?)>
3469 <!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
3470 <!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
3471 <!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
3472 <!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
3473 <!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
3474                              HasParentOrganization?,
3475                              InvokesEventBranch*,
3476                              ContactFilter*)>
3477 <!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
3478 <!ELEMENT HasParentOrganization (OrganizationFilter?,
3479                                   HasParentOrganization?)>
3480 <!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
3481                               RegistryEntryQuery?)>
3482 <!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
3483                               WithSourceAssociations?,
3484                               WithTargetAssociations?,
3485                               WithAuditableEvents?,
3486                               WithExternalLinks?)>
3487 <!ELEMENT WithClassifications (ClassificationFilter?)>
3488 <!ELEMENT WithSourceAssociations (AssociationFilter?)>
3489 <!ELEMENT WithTargetAssociations (AssociationFilter?)>
3490 <!ELEMENT WithAuditableEvents (AuditableEventFilter?)>
3491 <!ELEMENT WithExternalLinks (ExternalLinkFilter?)>
3492 <!ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*)>
```

```
3493 <!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
3494                               SourceAssociations?,
3495                               TargetAssociations?,
3496                               AuditableEvent*, ExternalLink*)>
3497 <!ELEMENT SourceAssociations (Association*)>
3498 <!ELEMENT TargetAssociations (Association*)>
3499 <!ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
3500                               RecursiveAssociationOption?,
3501                               WithDescription?)>
3502 <!ELEMENT RecursiveAssociationOption (AssociationType+)>
3503 <!ATTLIST RecursiveAssociationOption
3504     depthLimit CDATA #IMPLIED
3505 >
3506 <!ELEMENT AssociationType EMPTY>
3507 <!ATTLIST AssociationType
3508     role CDATA #REQUIRED
3509 >
3510 <!ELEMENT WithDescription EMPTY>
3511 <!ELEMENT ReturnRepositoryItemResult (RepositoryItem*)>
3512 <!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject
3513                           | ExternalLink)>
3514 <!ATTLIST RepositoryItem
3515     identifier CDATA #REQUIRED
3516     name CDATA #REQUIRED
3517     contentURI CDATA #REQUIRED
3518     objectType CDATA #REQUIRED
3519     status CDATA #REQUIRED
3520     stability CDATA #REQUIRED
3521     description CDATA #IMPLIED
3522 >
3523 <!ELEMENT RegistryPackage EMPTY>
3524 <!ELEMENT WithdrawnObject EMPTY>
3525 <!ELEMENT ExternalLinkItem EMPTY>
3526 <!ELEMENT ObjectFilter (Clause)>
3527 <!ELEMENT RegistryEntryFilter (Clause)>
3528 <!ELEMENT IntrinsicObjectFilter (Clause)>
3529 <!ELEMENT ExtrinsicObjectFilter (Clause)>
3530 <!ELEMENT PackageFilter (Clause)>
3531 <!ELEMENT OrganizationFilter (Clause)>
3532 <!ELEMENT ContactFilter (Clause)>
3533 <!ELEMENT ClassificationNodeFilter (Clause)>
3534 <!ELEMENT AssociationFilter (Clause)>
3535 <!ELEMENT ClassificationFilter (Clause)>
3536 <!ELEMENT ExternalLinkFilter (Clause)>
3537 <!ELEMENT SlotFilter (Clause)>
3538 <!ELEMENT ExternalIdentifierFilter (Clause)>
3539 <!ELEMENT AuditableEventFilter (Clause)>
3540 <!ELEMENT UserFilter (Clause)>
3541
3542 <!--
3543 The following lines define the XML syntax for Clause.
3544 -->
3545 <!ELEMENT Clause (SimpleClause | CompoundClause)>
3546 <!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
3547 <!ATTLIST SimpleClause
3548     leftArgument CDATA #REQUIRED
```

```
3549 >
3550 <!ELEMENT CompoundClause (Clause, Clause+)>
3551 <!ATTLIST CompoundClause
3552     connectivePredicate (And | Or) #REQUIRED
3553 >
3554 <!ELEMENT BooleanClause EMPTY>
3555 <!ATTLIST BooleanClause
3556     booleanPredicate (true | false) #REQUIRED
3557 >
3558 <!ELEMENT RationalClause (IntClause | FloatClause)>
3559 <!ATTLIST RationalClause
3560     logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
3561 >
3562 <!ELEMENT IntClause (#PCDATA)>
3563 <!ATTLIST IntClause
3564     e-dtype NMTOKEN #FIXED "int"
3565 >
3566 <!ELEMENT FloatClause (#PCDATA)>
3567 <!ATTLIST FloatClause
3568     e-dtype NMTOKEN #FIXED "float"
3569 >
3570 <!ELEMENT StringClause (#PCDATA)>
3571 <!ATTLIST StringClause
3572     stringPredicate
3573     (contains | -contains |
3574      startswith | -startswith |
3575      equal | -equal |
3576      endswith | -endswith) #REQUIRED
3577 >
3578 <!-- End FilterQuery DTD -->
3579
3580 <!-- Begin RegistryError definition -->
3581 <!-- The RegistryErrorList is derived from the ErrorList element from the
3582     ebXML Message Service Specification -->
3583 <!ELEMENT RegistryErrorList ( RegistryError+ )>
3584 <!ATTLIST RegistryErrorList
3585     highestSeverity ( Warning | Error ) 'Warning' >
3586
3587 <!ELEMENT RegistryError (#PCDATA) >
3588 <!ATTLIST RegistryError
3589     codeContext CDATA #REQUIRED
3590     errorCode CDATA #REQUIRED
3591     severity ( Warning | Error ) 'Warning'
3592     location CDATA #IMPLIED
3593     xml:lang NMTOKEN #IMPLIED>
3594
3595 <!ELEMENT RegistryResponse
3596 (( AdhocQueryResponse |
3597  GetContentResponse |
3598  GetClassificationTreeResponse |
3599  GetClassifiedObjectsResponse |
3600  GetRootClassificationNodesResponse )? ,
3601  RegistryErrorList? )>
3602 <!ATTLIST RegistryResponse
3603     status (success | failure ) #REQUIRED >
3604
```

```
3605 <!-- The contrived root node -->
3606
3607 <!ELEMENT RootElement
3608   ( SubmitObjectsRequest |
3609     ApproveObjectsRequest |
3610     DeprecateObjectsRequest |
3611     RemoveObjectsRequest |
3612     GetRootClassificationNodesRequest |
3613     GetClassificationTreeRequest |
3614     GetClassifiedObjectsRequest |
3615     AdhocQueryRequest |
3616     GetContentRequest |
3617     AddSlotsRequest |
3618     RemoveSlotsRequest |
3619     RegistryResponse |
3620     RegistryProfile) >
3621
3622 <!ELEMENT Href   (#PCDATA )>
3623
3624 <!ELEMENT XMLDocumentErrorLocn (DocumentId , Xpath )>
3625
3626 <!ELEMENT DocumentId  (#PCDATA )>
3627
3628 <!ELEMENT Xpath  (#PCDATA)>
```

## 3629 **Appendix C**

### 3630 **Interpretation of UML Diagrams**

3631 This section describes in *abstract terms* the conventions used to define ebXML  
3632 business process description in UML.

#### 3633 **C.1 UML Class Diagram**

3634 A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP])  
3635 required to implement an ebXML Registry Services and clients. See on page 20 for an  
3636 example. The UML class diagram contains:

- 3637
- 3638 1. A collection of UML interfaces where each interface represents a Service  
3639 Interface for a Registry service.
  - 3640 2. Tabular description of methods on each interface where each method represents  
3641 an Action (as defined by [ebCPP]) within the Service Interface representing the  
3642 UML interface.
  - 3643 3. Each method within a UML interface specifies one or more parameters, where  
3644 the type of each method argument represents the ebXML message type that is  
3645 exchanged as part of the Action corresponding to the method. Multiple  
3646 arguments imply multiple payload documents within the body of the  
3647 corresponding ebXML message.

## 3648 C.2 UML Sequence Diagram

3649 A UML sequence diagram is used to specify the business protocol representing the  
3650 interactions between the UML interfaces for a Registry specific ebXML business  
3651 process. A UML sequence diagram provides the necessary information to determine the  
3652 sequencing of messages, request to response association as well as request to error  
3653 response association as described by [ebCPP].

3654 Each sequence diagram shows the sequence for a specific conversation protocol as  
3655 method calls from the requestor to the responder. Method invocation may be  
3656 synchronous or asynchronous based on the UML notation used on the arrow-head for  
3657 the link. A half arrow-head represents asynchronous communication. A full arrow-head  
3658 represents synchronous communication.

3659 Each method invocation may be followed by a response method invocation from the  
3660 responder to the requestor to indicate the ResponseName for the previous Request.  
3661 Possible error response is indicated by a conditional response method invocation from  
3662 the responder to the requestor. See **Error! Reference source not found.** on page  
3663 **Error! Bookmark not defined.** for an example.

## 3664 Appendix D SQL Query

### 3665 D.1 SQL Query Syntax Specification

3666 This section specifies the rules that define the SQL Query syntax as a subset of  
3667 SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in  
3668 [SQL/PSM]. The SQL query syntax conforms to the <query specification>, modulo  
3669 the restrictions identified below:

- 3670 1. A <select list> may contain at most one <select sublist>.
- 3671 2. In a <select list> must be is a single column whose data type is UUID, from the  
3672 table in the <from clause>.
- 3673 3. A <derived column> may not have an <as clause>.
- 3674 4. <table expression> does not contain the optional <group by clause> and <having  
3675 clause> clauses.
- 3676 5. A <table reference> can only consist of <table name> and <correlation name>.
- 3677 6. A <table reference> does not have the optional AS between <table name> and  
3678 <correlation name>.
- 3679 7. There can only be one <table reference> in the <from clause>.
- 3680 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in  
3681 predicate> allows for the right hand side of the <in predicate> to be limited to a  
3682 restricted <query specification> as defined above.

- 3683 9. A <search condition> within the <where clause> may not include a <query  
3684 expression>.
- 3685 10. The SQL query syntax allows for the use of <sql invoked routines>  
3686 invocation from [SQL/PSM] as the RHS of the <in predicate>.

## 3687 D.2 Non-Normative BNF for Query Syntax Grammar

3688 The following BNF exemplifies the grammar for the registry query syntax. It is provided  
3689 here as an aid to implementors. Since this BNF is not based on [SQL] it is provided as  
3690 non-normative syntax. For the normative syntax rules see Appendix D.1.

```

3691 /*****
3692 * The Registry Query (Subset of SQL-92) grammar starts here
3693 *****/
3694 RegistryQuery = SQLSelect [ ";" ]
3695
3696 SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
3697
3698 SQLSelectCols = ID
3699
3700 SQLTableList = SQLTableRef
3701
3702 SQLTableRef = ID
3703
3704 SQLWhere = "WHERE" SQLOrExpr
3705
3706 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
3707
3708 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *
3709
3710 SQLNotExpr = [ "NOT" ] SQLCompareExpr
3711
3712 SQLCompareExpr =
3713   (SQLColRef "IS") SQLIsClause
3714   | SQLSumExpr [ SQLCompareExprRight ]
3715
3716 SQLCompareExprRight =
3717   SQLLikeClause
3718   | SQLInClause
3719   | SQLCompareOp SQLSumExpr
3720
3721 SQLCompareOp =
3722   "="
3723   | "<>"
3724   | ">"
3725   | ">="
3726   | "<"
3727   | "<="
3728
3729 SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
3730
3731 SQLLValueList = SQLLValueElement ( "," SQLLValueElement ) *
3732
3733 SQLLValueElement = "NULL" | SQLSelect
3734
3735 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
3736
3737 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
3738
3739
3740
3741
3742

```

```

3743 SQLPattern = STRING_LITERAL
3744
3745 SQLLiteral =
3746     STRING_LITERAL
3747     | INTEGER_LITERAL
3748     | FLOATING_POINT_LITERAL
3749
3750 SQLColRef = SQLLvalue
3751
3752 SQLLvalue = SQLLvalueTerm
3753
3754 SQLLvalueTerm = ID ( "." ID ) *
3755
3756 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr ) *
3757
3758 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr ) *
3759
3760 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
3761
3762 SQLTerm = "(" SQLOrExpr ")"
3763     | SQLColRef
3764     | SQLLiteral
3765
3766 INTEGER_LITERAL = ([ "0"-"9" ] ) +
3767
3768 FLOATING_POINT_LITERAL =
3769     ([ "0"-"9" ] ) + "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3770     | "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3771     | ([ "0"-"9" ] ) + EXPONENT
3772     | ([ "0"-"9" ] ) + ( EXPONENT ) ?
3773
3774 EXPONENT = [ "e", "E" ] ( [ "+" , "-" ] ) ? ([ "0"-"9" ] ) +
3775
3776 STRING_LITERAL: "'" (~[ "'" ] ) * ( '"' (~[ '"' ] ) * ) * "'"
3777
3778 ID = ( <LETTER> ) + ( "_" | "$" | "#" | <DIGIT> | <LETTER> ) *
3779 LETTER = [ "A"-"Z", "a"-"z" ]
3780 DIGIT = [ "0"-"9" ]

```

### 3781 D.3 Relational Schema For SQL Queries

```

3782
3783 --SQL Load file for creating the ebXML Registry tables
3784
3785
3786 --Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92
3787
3788
3789 CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
3790 CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
3791 CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
3792
3793 CREATE TYPE UUID UNDER ShortName FINAL;
3794 CREATE TYPE URI UNDER LongName FINAL;
3795
3796 CREATE TABLE ExtrinsicObject (
3797
3798     --RegistryObject Attributes
3799     id                                UUID PRIMARY KEY NOT NULL,
3800     name                              LongName,
3801     description                        FreeFormText,
3802     accessControlPolicy                UUID NOT NULL,
3803
3804     --Versionable attributes
3805     majorVersion                      INT DEFAULT 0 NOT NULL,
3806     minorVersion                      INT DEFAULT 1 NOT NULL,
3807
3808     --RegistryEntry attributes

```

```

3809     status                                INT DEFAULT 0 NOT NULL,
3810     userVersion                            ShortName,
3811     stability                               INT          DEFAULT 0 NOT NULL,
3812     expirationDate                         TIMESTAMP,
3813
3814 --ExtrinsicObject attributes
3815     contentURI                             URI,
3816     mimeType                               ShortName,
3817     objectType                             INT DEFAULT 0 NOT NULL,
3818     opaque                                 BOOLEAN DEFAULT false NOT NULL
3819
3820 );
3821
3822 CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
3823 --Must return a collection of UUIDs for related RegistryEntry instances
3824 }
3825
3826 CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
3827 --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
3828 --Collection must be in ascending order by timestamp
3829 }
3830
3831 CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
3832 --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
3833 }
3834
3835 CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
3836 --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
3837 }
3838
3839 CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
3840 --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
3841 }
3842
3843 CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
3844 --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
3845 }
3846
3847 CREATE TABLE Package (
3848
3849 --RegistryObject Attributes
3850     id                                    UUID PRIMARY KEY NOT NULL,
3851     name                                  LongName,
3852     description                           FreeFormText,
3853     accessControlPolicy                   UUID NOT NULL,
3854
3855 --Versionable attributes
3856     majorVersion                          INT DEFAULT 0 NOT NULL,
3857     minorVersion                          INT DEFAULT 1 NOT NULL,
3858
3859 --RegistryEntry attributes
3860     status                                INT DEFAULT 0 NOT NULL,
3861     userVersion                            ShortName,
3862     stability                               INT          DEFAULT 0 NOT NULL,
3863     expirationDate                         TIMESTAMP,
3864
3865 --Package attributes
3866 );
3867
3868 CREATE PROCEDURE Package_memberbjects(packageId) {
3869 --Must return a collection of UUIDs for RegistryEntrys that are memebers of this Package.
3870 }
3871
3872 CREATE TABLE ExternalLink (
3873
3874 --RegistryObject Attributes
3875     id                                    UUID PRIMARY KEY NOT NULL,
3876     name                                  LongName,
3877     description                           FreeFormText,
3878     accessControlPolicy                   UUID NOT NULL,

```



```

3879
3880 --Versionable attributes
3881     majorVersion           INT DEFAULT 0 NOT NULL,
3882     minorVersion          INT DEFAULT 1 NOT NULL,
3883
3884 --RegistryEntry attributes
3885     status                  INT DEFAULT 0 NOT NULL,
3886     userVersion            ShortName,
3887     stability              INT     DEFAULT 0 NOT NULL,
3888     expirationDate        TIMESTAMP,
3889
3890 --ExternalLink attributes
3891     externalURI            URI NOT NULL
3892 );
3893
3894 CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
3895 --Must return a collection of UUIDs for objects in this relationship
3896 }
3897
3898 CREATE TABLE ExternalIdentifier (
3899
3900 --RegistryObject Attributes
3901     id                     UUID PRIMARY KEY NOT NULL,
3902     name                   LongName,
3903     description            FreeFormText,
3904     accessControlPolicy    UUID NOT NULL,
3905
3906 --Versionable attributes
3907     majorVersion          INT DEFAULT 0 NOT NULL,
3908     minorVersion          INT DEFAULT 1 NOT NULL,
3909
3910 --RegistryEntry attributes
3911     status                INT DEFAULT 0 NOT NULL,
3912     userVersion           ShortName,
3913     stability              INT     DEFAULT 0 NOT NULL,
3914     expirationDate        TIMESTAMP,
3915
3916 --ExternalIdentifier attributes
3917     value                 ShortName NOT NULL
3918 );
3919
3920
3921 --A SlotValue row represents one value of one slot in some
3922 --RegistryEntry
3923 CREATE TABLE SlotValue (
3924
3925 --RegistryObject Attributes
3926     registryEntry         UUID     PRIMARY KEY NOT NULL,
3927
3928 --Slot attributes
3929     name                  LongName NOT NULL PRIMARY KEY NOT NULL,
3930     value                 ShortName NOT NULL
3931 );
3932
3933 CREATE TABLE Association (
3934 --RegistryObject Attributes
3935     id                     UUID PRIMARY KEY NOT NULL,
3936     name                   LongName,
3937     description            FreeFormText,
3938     accessControlPolicy    UUID NOT NULL,
3939
3940 --Versionable attributes
3941     majorVersion          INT DEFAULT 0 NOT NULL,
3942     minorVersion          INT DEFAULT 1 NOT NULL,
3943
3944 --RegistryEntry attributes
3945     status                INT DEFAULT 0 NOT NULL,
3946     userVersion           ShortName,
3947     stability              INT     DEFAULT 0 NOT NULL,
3948     expirationDate        TIMESTAMP,

```

```

3949
3950 --Association attributes
3951     associationType          INT NOT NULL,
3952     bidirectional           BOOLEAN DEFAULT false NOT NULL,
3953     sourceObject            UUID NOT NULL,
3954     sourceRole              ShortName,
3955     label                   ShortName,
3956     targetObject            UUID NOT NULL,
3957     targetRole              ShortName
3958 );
3959
3960 --Classification is currently identical to Association
3961 CREATE TABLE Classification (
3962 --RegistryObject Attributes
3963     id                      UUID PRIMARY KEY NOT NULL,
3964     name                    LongName,
3965     description              FreeFormText,
3966     accessControlPolicy     UUID NOT NULL,
3967
3968 --Versionable attributes
3969     majorVersion            INT DEFAULT 0 NOT NULL,
3970     minorVersion            INT DEFAULT 1 NOT NULL,
3971
3972 --RegistryEntry attributes
3973     status                  INT DEFAULT 0 NOT NULL,
3974     userVersion             ShortName,
3975     stability               INT     DEFAULT 0 NOT NULL,
3976     expirationDate          TIMESTAMP,
3977
3978 --Classification attributes. Assumes not derived from Association
3979     sourceObject            UUID NOT NULL,
3980     targetObject            UUID NOT NULL,
3981 );
3982
3983
3984 CREATE TABLE ClassificationNode (
3985 --RegistryObject Attributes
3986     id                      UUID PRIMARY KEY NOT NULL,
3987     name                    LongName,
3988     description              FreeFormText,
3989     accessControlPolicy     UUID NOT NULL,
3990
3991 --Versionable attributes
3992     majorVersion            INT DEFAULT 0 NOT NULL,
3993     minorVersion            INT DEFAULT 1 NOT NULL,
3994
3995 --RegistryEntry attributes
3996     status                  INT DEFAULT 0 NOT NULL,
3997     userVersion             ShortName,
3998     stability               INT     DEFAULT 0 NOT NULL,
3999     expirationDate          TIMESTAMP,
4000
4001 --ClassificationNode attributes
4002     parent                  UUID,
4003     path                    VARCHAR(512) NOT NULL,
4004     code                    ShortName
4005 );
4006
4007 CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
4008 --Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
4009 }
4010
4011 --Begin Registry Audit Trail tables
4012
4013 CREATE TABLE AuditableEvent (
4014 --RegistryObject Attributes
4015     id                      UUID PRIMARY KEY NOT NULL,
4016     name                    LongName,
4017     description              FreeFormText,
4018     accessControlPolicy     UUID NOT NULL,

```

```

4019
4020 --AuditableEvent attributes
4021 user                                UUID,
4022 eventType                            INT DEFAULT 0 NOT NULL,
4023 registryEntry                        UUID NOT NULL,
4024 timestamp                            TIMESTAMP NOT NULL,
4025 );
4026
4027
4028 CREATE TABLE User (
4029 --RegistryObject Attributes
4030 id                                    UUID PRIMARY KEY NOT NULL,
4031 name                                  LongName,
4032 description                            FreeFormText,
4033 accessControlPolicy                    UUID NOT NULL,
4034
4035 --User attributes
4036 organization                            UUID NOT NULL
4037
4038 --address attributes flattened
4039 address_city                            ShortName,
4040 address_country                        ShortName,
4041 address_postalCode                    ShortName,
4042 address_state                          ShortName,
4043 address_street                        ShortName,
4044
4045 email                                  ShortName,
4046
4047 --fax attribute flattened
4048 fax_areaCode                            VARCHAR(4) NOT NULL,
4049 fax_countryCode                        VARCHAR(4),
4050 fax_extension                            VARCHAR(8),
4051 fax_umber                                VARCHAR(8) NOT NULL,
4052 fax_url                                  URI
4053
4054 --mobilePhone attribute flattened
4055 mobilePhone_areaCode                    VARCHAR(4) NOT NULL,
4056 mobilePhone_countryCode                VARCHAR(4),
4057 mobilePhone_extension                    VARCHAR(8),
4058 mobilePhone_umber                        VARCHAR(8) NOT NULL,
4059 mobilePhone_url                          URI
4060
4061 --name attribute flattened
4062 name_firstName                            ShortName,
4063 name_middleName                          ShortName,
4064 name_lastName                             ShortName,
4065
4066 --pager attribute flattened
4067 pager_areaCode                            VARCHAR(4) NOT NULL,
4068 pager_countryCode                        VARCHAR(4),
4069 pager_extension                            VARCHAR(8),
4070 pager_umber                                VARCHAR(8) NOT NULL,
4071 pager_url                                  URI
4072
4073 --telephone attribute flattened
4074 telephone_areaCode                        VARCHAR(4) NOT NULL,
4075 telephone_countryCode                    VARCHAR(4),
4076 telephone_extension                      VARCHAR(8),
4077 telephone_umber                            VARCHAR(8) NOT NULL,
4078 telephone_url                              URI,
4079
4080 url                                        URI,
4081
4082 );
4083
4084 CREATE TABLE Organization (
4085 --RegistryObject Attributes
4086 id                                    UUID PRIMARY KEY NOT NULL,
4087 name                                  LongName,
4088

```

```

4089     description                                FreeFormText,
4090     accessControlPolicy                        UUID NOT NULL,
4091
4092 --Versionable attributes
4093     majorVersion                               INT DEFAULT 0 NOT NULL,
4094     minorVersion                              INT DEFAULT 1 NOT NULL,
4095
4096 --RegistryEntry attributes
4097     status                                     INT DEFAULT 0 NOT NULL,
4098     userVersion                               ShortName,
4099     stability                                 INT     DEFAULT 0 NOT NULL,
4100     expirationDate                           TIMESTAMP,
4101
4102 --Organization attributes
4103
4104 --Organization.address attribute flattened
4105     address_city                               ShortName,
4106     address_country                           ShortName,
4107     address_postalCode                        ShortName,
4108     address_state                             ShortName,
4109     address_street                            ShortName,
4110
4111 --primary contact for Organization, points to a User.
4112 --Note many Users may belong to the same Organization
4113     contact                                    UUID NOT NULL,
4114
4115 --Organization.fax attribute falttened
4116     fax_areaCode                              VARCHAR(4) NOT NULL,
4117     fax_countryCode                           VARCHAR(4),
4118     fax_extension                             VARCHAR(8),
4119     fax_umber                                 VARCHAR(8) NOT NULL,
4120     fax_url                                   URI,
4121
4122 --Organization.parent attribute
4123     parent                                    UUID,
4124
4125 --Organization.telephone attribute falttened
4126     telephone_areaCode                       VARCHAR(4) NOT NULL,
4127     telephone_countryCode                    VARCHAR(4),
4128     telephone_extension                       VARCHAR(8),
4129     telephone_umber                           VARCHAR(8) NOT NULL,
4130     telephone_url                             URI
4131 );
4132
4133
4134 --Note that the ebRIM security view is not visible through the public query mechanism
4135 --in the current release
4136
4137
4138 --The RegistryEntry View allows polymorphic queries over all ebRIM classes derived
4139 --from RegistryEntry
4140
4141 CREATE VIEW RegistryEntry (
4142 --RegistryObject Attributes
4143     id,
4144     name,
4145     description,
4146     accessControlPolicy,
4147
4148 --Versionable attributes
4149     majorVersion,
4150     minorVersion,
4151
4152 --RegistryEntry attributes
4153     status,
4154     userVersion,
4155     stability,
4156     expirationDate
4157 ) AS

```

```
4159 SELECT
4160 --RegistryObject Attributes
4161 id,
4162 name,
4163 description,
4164 accessControlPolicy,
4165
4166 --Versionable attributes
4167 majorVersion,
4168 minorVersion,
4169
4170 --RegistryEntry attributes
4171 status,
4172 userVersion,
4173 stability,
4174 expirationDate
4175
4176 FROM ExtrinsicObject
4177 UNION
4178
4179 SELECT
4180 --RegistryObject Attributes
4181 id,
4182 name,
4183 description,
4184 accessControlPolicy,
4185
4186 --Versionable attributes
4187 majorVersion,
4188 minorVersion,
4189
4190 --RegistryEntry attributes
4191 status,
4192 userVersion,
4193 stability,
4194 expirationDate
4195 FROM (Registry)Package
4196 UNION
4197
4198 SELECT
4199 --RegistryObject Attributes
4200 id,
4201 name,
4202 description,
4203 accessControlPolicy,
4204
4205 --Versionable attributes
4206 majorVersion,
4207 minorVersion,
4208
4209 --RegistryEntry attributes
4210 status,
4211 userVersion,
4212 stability,
4213 expirationDate
4214 FROM ClassificationNode;
```

4215

## 4216 **Appendix E Non-normative Content Based Ad Hoc Queries**

4217 The Registry SQL query capability supports the ability to search for content based not  
4218 only on metadata that catalogs the content but also the data contained within the  
4219 content itself. For example it is possible for a client to submit a query that searches for  
4220 all Collaboration Party Profiles that define a role named “seller” within a RoleName  
4221 element in the CPP document itself. Currently content-based query capability is  
4222 restricted to XML content.

### 4223 **E.1.1 Automatic Classification of XML Content**

4224 Content-based queries are indirectly supported through the existing classification  
4225 mechanism supported by the Registry.

4226 A submitting organization may define logical indexes on any XML schema or DTD when  
4227 it is submitted. An instance of such a logical index defines a link between a specific  
4228 attribute or element node in an XML document tree and a ClassificationNode in a  
4229 classification scheme within the registry.

4230 The registry utilizes this index to automatically classify documents that are instances of  
4231 the schema at the time the document instance is submitted. Such documents are  
4232 classified according to the data contained within the document itself.

4233 Such automatically classified content may subsequently be discovered by clients using  
4234 the existing classification-based discovery mechanism of the Registry and the query  
4235 facilities of the ObjectQueryManager.

4236 [Note] This approach is conceptually similar to the way databases support  
4237 indexed retrieval. DBAs define indexes on tables in the schema. When  
4238 data is added to the table, the data gets automatically indexed.

### 4239 **E.1.2 Index Definition**

4240 This section describes how the logical indexes are defined in the SubmittedObject  
4241 element defined in the Registry DTD. The complete Registry DTD is specified in  
4242 Appendix A.

4243 A SubmittedObject element for a schema or DTD may define a collection of  
4244 ClassificationIndexes in a ClassificationIndexList optional element. The  
4245 ClassificationIndexList is ignored if the content being submitted is not of the SCHEMA  
4246 objectType.

4247 The ClassificationIndex element inherits the attributes of the base class RegistryObject  
4248 in [ebRIM]. It then defines specialized attributes as follows:

- 4249 1. classificationNode: This attribute references a specific ClassificationNode by its  
4250 ID.

- 4251 2. contentIdentifier: This attribute identifies a specific data element within the  
 4252 document instances of the schema using an XPATH expression as defined by  
 4253 [XPT].

### 4254 **E.1.3 Example Of Index Definition**

4255 To define an index that automatically classifies a CPP based upon the roles defined  
 4256 within its RoleName elements, the following index must be defined on the CPP schema  
 4257 or DTD:

```
4258 <ClassificationIndex
4259     classificationNode='id-for-role-classification-scheme'
4260     contentIdentifier='/Role//RoleName'
4261 />
```

### 4262 **E.1.4 Proposed XML Definition**

```
4263 <!--
4264 A ClassificationIndexList is specified on ExtrinsicObjects of objectType
4265 'Schema' to define an automatic Classification of instance objects of the
4266 schema using the specified classificationNode as parent and a
4267 ClassificationNode created or selected by the object content as selected by
4268 the contentIdentifier
4269 -->
4270 <!ELEMENT ClassificationIndex EMPTY>
4271 <!ATTLIST ClassificationIndex
4272     %ObjectAttributes;
4273     classificationNode IDREF #REQUIRED
4274     contentIdentifier CDATA #REQUIRED
4275 >
4276
4277 <!-- ClassificationIndexList contains new ClassificationIndexes -->
4278 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

### 4279 **E.1.5 Example of Automatic Classification**

4280 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the  
 4281 CPP is submitted it will automatically be classified by two ClassificationNodes named  
 4282 “buyer” and “seller” that are both children of the ClassificationNode (e.g. a node named  
 4283 Role) specified in the classificationNode attribute of the ClassificationIndex. Note that if  
 4284 either of the two ClassificationNodes named “buyer” and “seller” did not previously exist,  
 4285 the ObjectManager would automatically create these ClassificationNodes.

## 4286 **Appendix F Security Implementation Guideline**

4287 This section provides a suggested blueprint for how security processing may be  
 4288 implemented in the Registry. It is meant to be illustrative not prescriptive. Registries  
 4289 may choose to have different implementations as long as they support the default  
 4290 security roles and authorization rules described in this document.

## 4291 **F.1 Authentication**

- 4292 1. As soon as a message is received, the first work is the authentication. A principal  
4293 object is created.
- 4294 2. If the message is signed, it is verified (including the validity of the certificate) and the  
4295 DN of the certificate becomes the identity of the principal. Then the Registry is  
4296 searched for the principal and if found, the roles and groups are filled in.
- 4297 3. If the message is not signed, an empty principal is created with the role  
4298 RegistryGuest. This step is for symmetry and to decouple the rest of the processing.
- 4299 4. Then the message is processed for the command and the objects it will act on.

## 4300 **F.2 Authorization**

4301 For every object, the access controller will iterate through all the AccessControlPolicy  
4302 objects with the object and see if there is a chain through the permission objects to  
4303 verify that the requested method is permitted for the Principal. If any of the permission  
4304 objects which the object is associated with has a common role, or identity, or group with  
4305 the principal, the action is permitted.

## 4306 **F.3 Registry Bootstrap**

4307 When a Registry is newly created, a default Principal object should be created with the  
4308 identity of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any  
4309 message signed by the Registry Admin will get all the privileges.

4310 When a Registry is newly created, a singleton instance of AccessControlPolicy is  
4311 created as the default AccessControlPolicy. This includes the creation of the necessary  
4312 Permission instances as well as the Privileges and Privilege attributes.

## 4313 **F.4 Content Submission – Client Responsibility**

4314 The Registry client has to sign the contents before submission – otherwise the content  
4315 will be rejected.

## 4316 **F.5 Content Submission – Registry Responsibility**

- 4317 1. Like any other request, the client will be first authenticated. In this case, the Principal  
4318 object will get the DN from the certificate.
- 4319 2. As per the request in the message, the RegistryEntry will be created.
- 4320 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.



- 4321 4. If a principal with the identity of the SO is not available, an identity object with the  
4322 SO's DN is created
- 4323 5. A principal with this identity is created

## 4324 **F.6 Content Delete/Deprecate – Client Responsibility**

4325 The Registry client has to sign the payload (not entire message) before submission, for  
4326 authentication purposes; otherwise, the request will be rejected

## 4327 **F.7 Content Delete/Deprecate – Registry Responsibility**

- 4328 1. Like any other request, the client will be first authenticated. In this case, the Principal  
4329 object will get the DN from the certificate. As there will be a principal with this identity  
4330 in the Registry, the Principal object will get all the roles from that object
- 4331 2. As per the request in the message (delete or deprecate), the appropriate method in  
4332 the RegistryObject class will be accessed.
- 4333 3. The access controller performs the authorization by iterating through the Permission  
4334 objects associated with this object via the singleton default AccessControlPolicy.
- 4335 4. If authorization succeeds then the action will be permitted. Otherwise an error  
4336 response is sent back with a suitable AuthorizationException error message.

## 4337 **Appendix G Native Language Support (NLS)**

### 4338 **G.1 Definitions**

4339 Although this section discusses only character set and language, the following terms  
4340 have to be defined clearly.  
4341

#### 4342 **G.1.1 Coded Character Set (CCS):**

4343 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130].  
4344 Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.  
4345

#### 4346 **G.1.2 Character Encoding Scheme (CES):**

4347 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of  
4348 CES are ISO-2022, UTF-8.

### 4349 **G.1.3 Character Set (charset):**

4350 charset is a set of rules for mapping from a sequence of octets to a sequence of  
4351 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-  
4352 KR.

4353  
4354 A list of registered character sets can be found at [IANA].

### 4355 **G.2 NLS And Request / Response Messages**

4356 For the accurate processing of data in both registry client and registry services, it is  
4357 essential to know which character set is used. Although the body part of the transaction  
4358 may contain the charset in xml encoding declaration, registry client and registry services  
4359 shall specify charset parameter in MIME header when they use text/xml. Because as  
4360 defined in [RFC 3023], if a text/xml entity is received with the charset parameter  
4361 omitted, MIME processors and XML processors MUST use the default charset value of  
4362 "us-ascii".

4363  
4364 Ex. Content-Type: text/xml; charset=ISO-2022-JP

4365  
4366 Also, when an application/xml entity is used, the charset parameter is optional, and  
4367 registry client and registry services must follow the requirements in Section 4.3.3 of  
4368 [REC-XML] which directly address this contingency.

4369  
4370 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

### 4371 **G.3 NLS And Storing of RegistryEntry**

4372 This section provides NLS guidelines on how a registry should store **RegistryEntry**  
4373 instances.

#### 4374 **G.3.1 Character Set of RegistryEntry**

4375 This is basically an implementation issue because the actual character set that the  
4376 **RegistryEntry** is stored with, does not affect the interface. However, it is highly  
4377 recommended to use UTF-16 or UTF-8 for covering various languages.

#### 4378 **G.3.2 Language Information of RegistryEntry**

4379 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]). If the  
4380 xml:lang attribute is specified, then the registry may use that language code as the  
4381 value of a special Slot with name **language** and slotType of **nls** in the **RegistryEntry**.  
4382 The value must be compliant to [RFC 1766]. Slots are defined in [ebRIM].

## 4383 G.4 NLS And Storing of Repository Items

4384 This section provides NLS guidelines on how a registry should store repository items.

### 4385 G.4.1 Character Set of Repository Items

4386 Unlike the character set of **RegistryEntry**, the charset of a repository item must be  
 4387 preserved as it is originally specified in the transaction. The registry may use a special  
 4388 Slot with name **repositoryItemCharset**, and sloType of **nls** for the **RegistryEntry** for  
 4389 storing the charset of the corresponding repository item. Value must be the one defined  
 4390 in [RFC 2277], [RFC 2278]. The **repositoryItemCharset** is optional because not all  
 4391 repository items require it.

### 4392 G.4.2 Language information of repository item

4393 Specifying only character set is not enough to tell which language is used in the  
 4394 repository item. A registry may use a special Slot with name **repositoryItemLang**, and  
 4395 sloType of **nls** to store that information. This attribute is optional because not all  
 4396 repository items require it. Value must be compliant to [RFC 1766]

4397  
 4398 This document currently specifies only the method of sending the information of  
 4399 character set and language, and how it is stored in a registry. However, the language  
 4400 information may be used as one of the query criteria, such as retrieving only DTD  
 4401 written in French. Furthermore, a language negotiation procedure, like registry client is  
 4402 asking a favorite language for messages from registry services, could be another  
 4403 functionality for the future revision of this document.

## 4404 Appendix H Terminology Mapping

4405 While every attempt has been made to use the same terminology used in other works  
 4406 there are some terminology differences.

4407 The following table shows the terminology mapping between this specification and that  
 4408 used in other specifications and working groups.

4409

This Document	OASIS	ISO 11179
“repository item”	RegisteredObject	
RegistryEntry	RegistryEntry	Administered Component
ExternalLink	RelatedData	N/A
Object.id	regEntryId, orgId, etc.	
ExtrinsicObject.uri	objectURL	
ExtrinsicObject.objectType	defnSource, objectType	

RegistryEntry.name	commonName	
Object.description	shortDescription, Description	
ExtrinsicObject.mimeType	objectType="mime" fileType="<mime type>"	
Versionable.majorVersion	userVersion only	
Versionable.minorVersion	userVersion only	
RegistryEntry.status	registrationStatus	

4410

**Table 1: Terminology Mapping Table**

4411

## 4411 **References**

- 4412 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 4413 [GLS] ebXML Glossary, [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)
- 4414 [TA] ebXML Technical Architecture
- 4415 [http://www.ebxml.org/specdrafts/ebXML\\_TA\\_v1.0.pdf](http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf)
- 4416 [OAS] OASIS Information Model
- 4417 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 4418 [ISO] ISO 11179 Information Model
- 4419 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 4420
- 4421 [ebRIM] ebXML Registry Information Model
- 4422 [http://www.ebxml.org/project\\_teams/registry/private/registryInfoModelv0.54.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf)
- 4423 [ebBPM] ebXML Business Process Specification Schema
- 4424 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 4425 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 4426 [http://www.ebxml.org/project\\_teams/trade\\_partner/private/](http://www.ebxml.org/project_teams/trade_partner/private/)
- 4427 [ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
- 4428 <http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html>
- 4429 [CTB] Context table informal document from Core Components
- 4430 [ebMS] ebXML Messaging Service Specification, Version 0.21
- 4431 [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)
- 4432 [ERR] ebXML TRP Error Handling Specification
- 4433 [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)
- 4434 [SEC] ebXML Risk Assessment Technical Report, Version 3.6
- 4435 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 4436 [XPT] XML Path Language (XPath) Version 1.0
- 4437 <http://www.w3.org/TR/xpath>
- 4438 [SQL] Structured Query Language (FIPS PUB 127-2)
- 4439 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 4440
- 4441 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 4442 (SQL/PSM) [ISO/IEC 9075-4:1996]

- 4443  
4444 [IANA] IANA (Internet Assigned Numbers Authority).  
4445 Official Names for Character Sets, ed. Keld Simonsen et al.  
4446 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>  
4447  
4448 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:  
4449 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.  
4450 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>  
4451  
4452 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:  
4453 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.  
4454 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>  
4455  
4456 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:  
4457 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.  
4458 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>  
4459  
4460 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:  
4461 XML Media Types, ed. M. Murata. 2001.  
4462 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>  
4463  
4464 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second  
4465 Edition)  
4466 <http://www.w3.org/TR/REC-xml>  
4467  
4468 [UUID] DCE 128 bit Universal Unique Identifier  
4469 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
4470 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>  
4471  
4472 [WSDL]W3C Note. Web Services Description Language (WSDL) 1.1  
4473 <http://www.w3.org/TR/wsdl>  
4474 [SOAP11]W3C Note. Simple Object Access Protocol, May 2000,  
4475 <http://www.w3.org/TR/SOAP>  
4476 [SOAPAt]W3C Note: SOAP with Attachments, Dec 2000,  
4477 <http://www.w3.org/TR/SOAP-attachments>  
4478

## 4479 Disclaimer

4480 The views and specification expressed in this document are those of the authors and  
4481 are not necessarily those of their employers. The authors and their employers  
4482 specifically disclaim responsibility for any problems arising from correct or incorrect  
4483 implementation or use of this design.  
4484

4484 **Contact Information**

## 4485 Team Leader

4486 Name: Lisa Carnahan  
4487 Company:  
4488 Street:  
4489 City, State, Postal Code:  
4490 Country: USA  
4491 Phone:  
4492 Email: lisa.carnahan@????

4493

## 4494 Vice Team Lead

4495 Name: Yutaka Yoshida  
4496 Company: Sun Microsystems  
4497 Street: 901 San Antonio Road, MS UMPK17-102  
4498 City, State, Postal Code: Palo Alto, CA 94303  
4499 Country: USA  
4500 Phone: 650.786.5488  
4501 Email: Yutaka.Yoshida@eng.sun.com

4502

## 4503 Editor

4504 Name: Anne A. Fischer  
4505 Company: Drummond Group, Inc.  
4506 Street: 4700 Bryant Irvin Ct., Suite 303  
4507 City, State, Postal Code: Fort Worth, Texas 76107-7645  
4508 Country: USA  
4509 Phone: 817-371-2367  
4510 Email: anne@drummondgroup.com

4511

4512

## 4512 **Copyright Statement**

4513 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

4514

4515 This document and translations of it may be copied and furnished to others, and  
4516 derivative works that comment on or otherwise explain it or assist in its implementation  
4517 MAY be prepared, copied, published and distributed, in whole or in part, without  
4518 restriction of any kind, provided that the above copyright notice and this paragraph are  
4519 included on all such copies and derivative works. However, this document itself MAY  
4520 not be modified in any way, such as by removing the copyright notice or references to  
4521 ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other  
4522 than English.

4523

4524 The limited permissions granted above are perpetual and will not be revoked by ebXML  
4525 or its successors or assigns.

4526

4527 This document and the information contained herein is provided on an  
4528 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
4529 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
4530 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
4531 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
4532 PURPOSE.