



Creating A Single Global Electronic Market

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

OASIS/ebXML Registry Services Specification v1.0 DRAFT

OASIS/ebXML Registry Technical Committee

27 June 2001

1 Status of this Document

Distribution of this document is unlimited.

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

This version:

<http://www.oasis-open.org/committees/regrep/document/rsV1-0.doc>

Latest version:

<http://www.oasis-open.org/committees/regrep/documents/rsV1-0.doc>

28 **2 OASIS/ebXML Registry Technical Committee**

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

32

33 Nagwa Abdelghfour, Sun Microsystems
34 Nicholas Berry, Boeing
35 Kathryn Breining, Boeing
36 Lisa Carnahan, US NIST (TC Chair)
37 Dan Chang, IBM
38 Joseph M. Chiusano, LMI
39 Joe Dalman, Tie Commerce
40 Suresh Damodaran, Sterling Commerce
41 Vadim Draluk, BEA
42 John Evdemon, Vitria Technologies
43 Anne Fischer, Drummond Group
44 Sally Fuger, AIAG
45 Len Gallagher, NIST
46 Michael Joya, XMLGlobal
47 Una Kearns, Documentum
48 Kyu-Chul Lee, Chungnam National University
49 Megan MacMillan, Gartner Solista
50 Norbert Mikula, DataChannel
51 Joel Munter, Intel
52 Farrukh Najmi, Sun Microsystems
53 Joel Neu, Vitria Technologies
54 Sanjay Patil, IONA
55 Neal Smith, Chevron
56 Nikola Stojanovic, Encoda Systems Inc.
57 David Webber, XMLGlobal
58 Prasad Yendluri, webmethods
59 Yutaka Yoshida, Sun Microsystems

60

60	Table of Contents	
61	1 Status of this Document	1
62	2 ebXML participants	2
63	Table of Contents	3
64	Table of Tables	7
65	3 Introduction	8
66	3.1 Summary of Contents of Document	8
67	3.2 General Conventions	8
68	3.3 Audience	8
69	3.4 Related Documents.....	8
70	4 Design Objectives	9
71	4.1 Goals.....	9
72	4.2 Caveats and Assumptions.....	9
73	5 System Overview	9
74	5.1 What The ebXML Registry Does	9
75	5.2 How The ebXML Registry Works	9
76	5.2.1 Schema Documents Are Submitted.....	10
77	5.2.2 Business Process Documents Are Submitted	10
78	5.2.3 Seller's Collaboration Protocol Profile Is Submitted.....	10
79	5.2.4 Buyer Discovers The Seller	10
80	5.2.5 CPA Is Established.....	10
81	5.3 Where the Registry Services May Be Implemented	11
82	5.4 Implementation Conformance	11
83	5.4.1 Conformance as an ebXML Registry.....	11
84	5.4.2 Conformance as an ebXML Registry Client.....	11
85	6 Registry Architecture	12
86	6.1 ebXML Registry Profiles and Agreements	13
87	6.2 Client To Registry Communication Bootstrapping.....	13
88	6.3 Interfaces	14
89	6.4 Interfaces Exposed By The Registry	15
90	6.4.1 Synchronous and Asynchronous Responses	15
91	6.4.2 Interface RegistryService	15
92	6.4.3 Interface ObjectManager	16
93	6.4.4 Interface ObjectQueryManager	16
94	6.5 Interfaces Exposed By Registry Clients	17
95	6.5.1 Interface RegistryClient	17
96	6.6 Registry Response Class Hierarchy.....	18
97	7 Object Management Service	19
98	7.1 Life Cycle of a Repository Item	19

99	7.2	RegistryObject Attributes.....	20
100	7.3	The Submit Objects Protocol.....	20
101	7.3.1	Universally Unique ID Generation	21
102	7.3.2	ID Attribute And Object References.....	21
103	7.3.3	Sample SubmitObjectsRequest.....	22
104	7.4	The Add Slots Protocol	24
105	7.5	The Remove Slots Protocol.....	24
106	7.6	The Approve Objects Protocol	25
107	7.7	The Deprecate Objects Protocol	26
108	7.8	The Remove Objects Protocol	26
109	7.8.1	Deletion Scope DeleteRepositoryItemOnly	27
110	7.8.2	Deletion Scope DeleteAll.....	27
111	8	Object Query Management Service	27
112	8.1	Browse and Drill Down Query Support.....	28
113	8.1.1	Get Root Classification Nodes Request	28
114	8.1.2	Get Classification Tree Request.....	29
115	8.1.3	Get Classified Objects Request.....	29
116	8.1.3.1	Get Classified Objects Request Example	30
117	8.2	Filter Query Support.....	32
118	8.2.1	FilterQuery.....	34
119	8.2.2	RegistryEntryQuery	36
120	8.2.3	AuditableEventQuery.....	42
121	8.2.4	ClassificationNodeQuery	45
122	8.2.5	RegistryPackageQuery.....	48
123	8.2.6	OrganizationQuery	50
124	8.2.7	ReturnRegistryEntry	53
125	8.2.8	ReturnRepositoryItem.....	57
126	8.2.9	Registry Filters	61
127	8.2.10	XML Clause Constraint Representation	65
128	8.3	SQL Query Support.....	69
129	8.3.1	SQL Query Syntax Binding To [ebRIM].....	69
130	8.3.1.1	Interface and Class Binding	69
131	8.3.1.2	Accessor Method To Attribute Binding.....	70
132	8.3.1.3	Primitive Attributes Binding	70
133	8.3.1.4	Reference Attribute Binding	70
134	8.3.1.5	Complex Attribute Binding.....	70
135	8.3.1.6	Collection Attribute Binding.....	71
136	8.3.2	Semantic Constraints On Query Syntax.....	71
137	8.3.3	SQL Query Results.....	71
138	8.3.4	Simple Metadata Based Queries	72
139	8.3.5	RegistryEntry Queries	72
140	8.3.6	Classification Queries.....	72
141	8.3.6.1	Identifying ClassificationNodes	72
142	8.3.6.2	Getting Root Classification Nodes	72
143	8.3.6.3	Getting Children of Specified ClassificationNode.....	73

144	8.3.6.4	Getting Objects Classified By a ClassificationNode	73
145	8.3.6.5	Getting ClassificationNodes That Classify an Object...73	
146	8.3.7	Association Queries.....	73
147	8.3.7.1	Getting All Association With Specified Object As Its Source	
148		73	
149	8.3.7.2	Getting All Association With Specified Object As Its Target	
150		74	
151	8.3.7.3	Getting Associated Objects Based On Association Attributes	
152		74	
153	8.3.7.4	Complex Association Queries	74
154	8.3.8	Package Queries.....	74
155	8.3.8.1	Complex Package Queries	74
156	8.3.9	ExternalLink Queries	74
157	8.3.9.1	Complex ExternalLink Queries.....	75
158	8.3.10	Audit Trail Queries.....	75
159	8.4	Ad Hoc Query Request/Response	75
160	8.5	Content Retrieval	76
161	8.5.1	Identification Of Content Payloads	76
162	8.5.2	GetContentResponse Message Structure	76
163	8.6	Query And Retrieval: Typical Sequence	77
164	9	Registry Security.....	78
165	9.1	Integrity of Registry Content.....	79
166	9.1.1	Message Payload Signature.....	79
167	9.2	Authentication	79
168	9.2.1	Message Header Signature.....	79
169	9.3	Confidentiality.....	80
170	9.3.1	On-the-wire Message Confidentiality.....	80
171	9.3.2	Confidentiality of Registry Content	80
172	9.4	Authorization	80
173	9.4.1	Pre-defined Roles For Registry Users	80
174	9.4.2	Default Access Control Policies.....	80
175	Appendix A	ebXML Registry DTD Definition.....	81
176	Appendix B	Interpretation of UML Diagrams.....	92
177	B.1	UML Class Diagram	92
178	B.2	UML Sequence Diagram.....	93
179	Appendix C	SQL Query.....	93
180	C.1	SQL Query Syntax Specification	93
181	C.2	Non-Normative BNF for Query Syntax Grammar	94
182	C.3	Relational Schema For SQL Queries	95
183	Appendix D	Non-normative Content Based Ad Hoc Queries	102
184	D.1.1	Automatic Classification of XML Content.....	102
185	D.1.2	Index Definition.....	102
186	D.1.3	Example Of Index Definition.....	103

187	D.1.4 Proposed XML Definition	103
188	D.1.5 Example of Automatic Classification.....	103
189	Appendix E Security Implementation Guideline.....	103
190	E.1 Authentication	104
191	E.2 Authorization	104
192	E.3 Registry Bootstrap.....	104
193	E.4 Content Submission – Client Responsibility	104
194	E.5 Content Submission – Registry Responsibility	104
195	E.6 Content Delete/Deprecate – Client Responsibility.....	105
196	E.7 Content Delete/Deprecate – Registry Responsibility.....	105
197	Appendix F Native Language Support (NLS).....	105
198	F.1 Definitions	105
199	F.1.1 Coded Character Set (CCS):.....	105
200	F.1.2 Character Encoding Scheme (CES):.....	105
201	F.1.3 Character Set (charset):	106
202	F.2 NLS And Request / Response Messages	106
203	F.3 NLS And Storing of RegistryEntry	106
204	F.3.1 Character Set of <i>RegistryEntry</i>	106
205	F.3.2 Language Information of <i>RegistryEntry</i>	106
206	F.4 NLS And Storing of Repository Items.....	107
207	F.4.1 Character Set of Repository Items.....	107
208	F.4.2 Language information of repository item	107
209	Appendix G Terminology Mapping	107
210	References	109
211	Disclaimer	110
212	Contact Information	111
213	Copyright Statement	112
214	Table of Figures	
215	Figure 1: Registry Architecture Supports Flexible Topologies.....	12
216	Figure 2: ebXML Registry Interfaces.....	14
217	Figure 3: Registry Reponse Class Hierarchy	18
218	Figure 4: Life Cycle of a Repository Item	20
219	Figure 5: Submit Objects Sequence Diagram	20
220	Figure 7: Add Slots Sequence Diagram	24
221	Figure 8: Remove Slots Sequence Diagram	25
222	Figure 9: Approve Objects Sequence Diagram	25

223	Figure 10: Deprecate Objects Sequence Diagram	26
224	Figure 11: Remove Objects Sequence Diagram	27
225	Figure 12: Get Root Classification Nodes Sequence Diagram.....	28
226	Figure 14: Get Classification Tree Sequence Diagram	29
227	Figure 16: A Sample Geography Classification.....	30
228	Figure 17: Get Classified Objects Sequence Diagram	31
229	Figure 19: Example ebRIM Binding.....	32
230	Figure 20: The Clause base structure	65
231	Figure 21: Submit Ad Hoc Query Sequence Diagram	75
232	Figure 23: Typical Query and Retrieval Sequence.....	78

233 **Table of Tables**

234	Table 1: Terminology Mapping Table	108
235		
236		

236 **3 Introduction**

237 **3.1 Summary of Contents of Document**

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

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

243 **3.2 General Conventions**

244 The following conventions are used throughout this document:

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

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

256 **3.3 Audience**

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

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

261 **3.4 Related Documents**

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

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

268 **4 Design Objectives**

269 **4.1 Goals**

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

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

275 **4.2 Caveats and Assumptions**

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

279 It is assumed that:

- 280 1. Interoperability requirements dictate that the ebXML Message Services
281 Specification is used between an ebXML Registry and an ebXML Registry
282 Client. The use of other communication means is not precluded; however, in
283 those cases interoperability cannot be assumed. Other communication means
284 are outside the scope of this specification.
- 285 2. All access to the Registry content is exposed via the interfaces defined for the
286 Registry Services.
- 287 3. The Registry makes use of a Repository for storing and retrieving persistent
288 information required by the Registry Services. This is an implementation detail
289 that will not be discussed further in this specification.

290 **5 System Overview**

291 **5.1 What The ebXML Registry Does**

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

297 **5.2 How The ebXML Registry Works**

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

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

308 **5.2.1 Schema Documents Are Submitted**

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

313 **5.2.2 Business Process Documents Are Submitted**

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

318 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

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

323 **5.2.4 Buyer Discovers The Seller**

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

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

330 **5.2.5 CPA Is Established**

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

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

337 **5.3 Where the Registry Services May Be Implemented**

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

340 **5.4 Implementation Conformance**

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

348 **5.4.1 Conformance as an ebXML Registry**

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

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

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

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

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

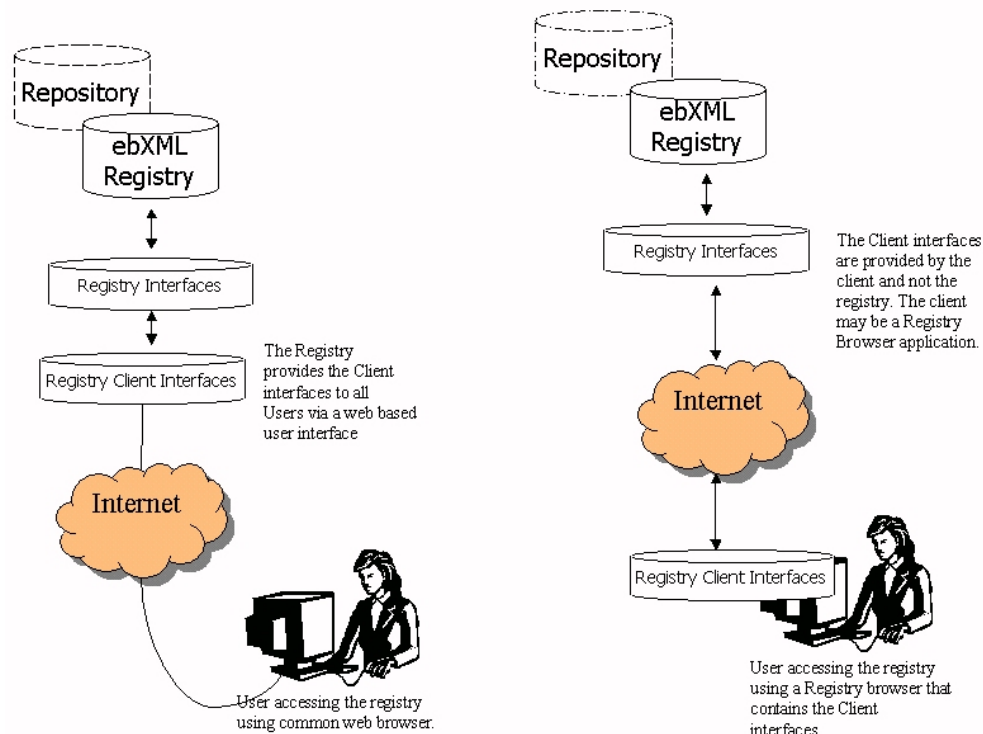
364 6 Registry Architecture

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

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

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

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



384

385

Figure 1: Registry Architecture Supports Flexible Topologies

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

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

393 Examples include:

- 394 • an ebXML Registry of Registries that serves as a centralized registration point;
- 395 • cooperative ebXML Registries, where registries register with each other in a
396 federation;
- 397 • registration of ebXML Registries with other Registry systems that act as white
398 pages or yellow pages. The document [ebXML-UDDI] provides an example of
399 ebXML Registries being discovered through a system of emerging white/yellow
400 pages known as UDDI.

401 **6.1 ebXML Registry Profiles and Agreements**

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

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

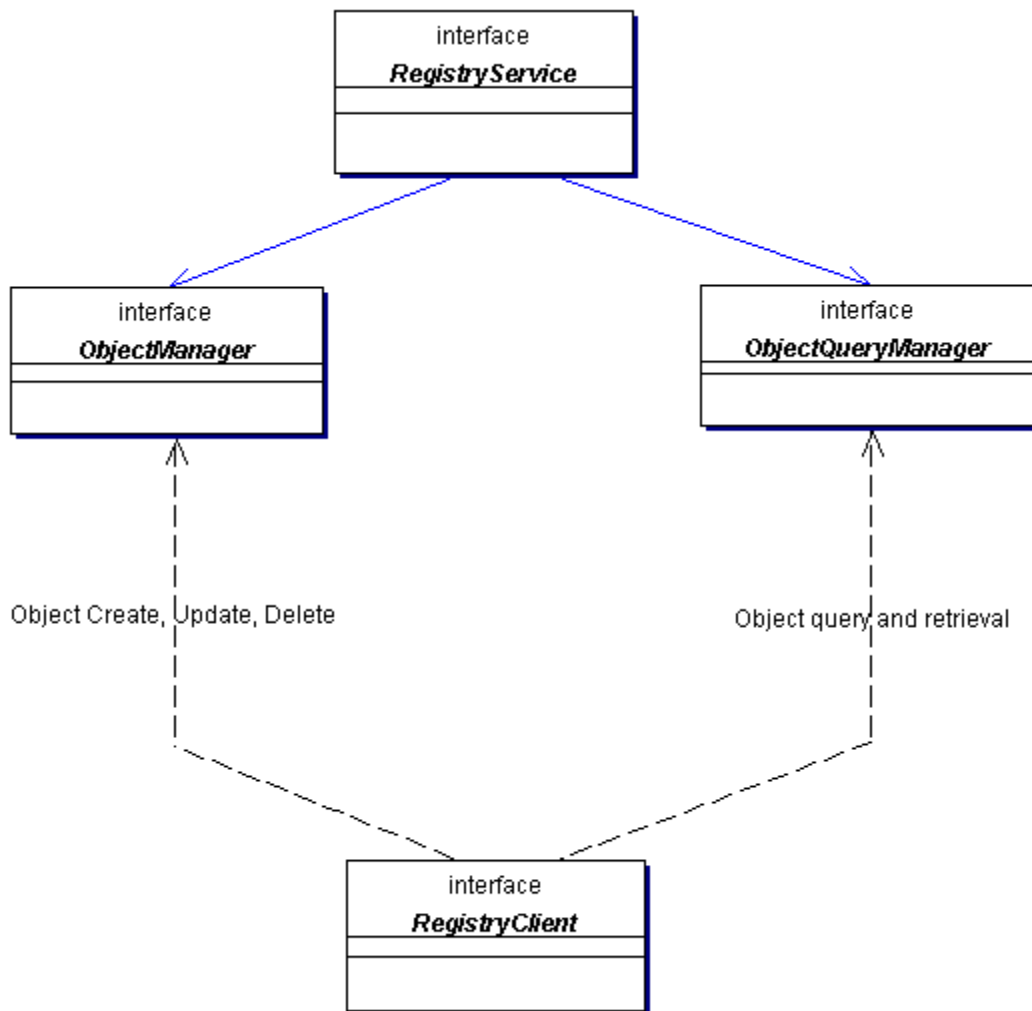
411 The CPA between clients and the Registry should describe the interfaces that the
412 Registry and the client expose to each other for Registry-specific interactions. These
413 interfaces are described in Figure 2 and subsequent sections. The definition of the
414 Registry CPP template and a Registry Client CPP template are beyond the scope of this
415 document.

416 **6.2 Client To Registry Communication Bootstrapping**

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

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

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



429

430

Figure 2: ebXML Registry Interfaces

431 6.3 Interfaces

432 This specification defines the interfaces exposed by both the Registry (Section 6.4) and
 433 the Registry Client (Section 6.5). Figure 2 shows the relationship between the
 434 interfaces and the mapping of specific Registry interfaces with specific Registry Client
 435 interfaces.

436 6.4 Interfaces Exposed By The Registry

437 When using the ebXML Messaging Services Specification, ebXML Registry Services
438 elements correspond to Messaging Services elements as follows:

- 439 • The value of the Service element in the MessageHeader is an ebXML Registry
440 Service interface name (e.g., “ObjectManager”). The type attribute of the Service
441 element should have a value of “ebXMLRegistry”.
- 442 • The value of the Action element in the MessageHeader is an ebXML Registry
443 Service method name (e.g., “submitObjects”).

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

447 6.4.1 Synchronous and Asynchronous Responses

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

- 449 • Asynchronous response
 - 450 ○ MessageHeader only;
 - 451 ○ No registry response element (e.g., AdHocQueryResponse and
452 GetContentResponse).
- 453 • Synchronous response
 - 454 ○ MessageHeader;
 - 455 ○ Registry response element including
 - 456 ▪ a status attribute (success or failure)
 - 457 ▪ an optional ebXML Error.

458 The ebXML Registry implements the following interfaces as its services (Registry
459 Services).

460 6.4.2 Interface RegistryService

461 _____
462 This is the principal interface implemented by the Registry. It provides the methods that
463 are used by the client to discover service-specific interfaces implemented by the
464 Registry.
465 _____

Method Summary of RegistryService

ObjectManager	getObjectManager () Returns the ObjectManager interface implemented by
-------------------------------	---

	the Registry service.
ObjectQueryManager	getObjectQueryManager () Returns the ObjectQueryManager interface implemented by the Registry service.

466 6.4.3 Interface ObjectManager

467

468 This is the interface exposed by the Registry Service that implements the Object life
469 cycle management functionality of the Registry. Its methods are invoked by the Registry
470 Client. For example, the client may use this interface to submit objects, to classify and
471 associate objects and to deprecate and remove objects. For this specification the
472 semantic meaning of submit, classify, associate, deprecate and remove is found in
473 [ebRIM].

474

Method Summary of ObjectManager

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

475 6.4.4 Interface ObjectQueryManager

476

477 This is the interface exposed by the Registry that implements the Object Query
478 management service of the Registry. Its methods are invoked by the Registry Client.

479 For example, the client may use this interface to perform browse and drill down queries
 480 or ad hoc queries on registry content.

481

Method Summary of ObjectQueryManager

RegistryResponse	getClassificationTree (GetClassificationTreeRequest req) Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.
RegistryResponse	getClassifiedObjects (GetClassifiedObjectsRequest req) Returns a collection of references to RegistryEntries classified under specified ClassificationItem.
RegistryResponse	getContent () 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.
RegistryResponse	getRootClassificationNodes (GetRootClassificationNodesRequest req) Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.
RegistryResponse	submitAdhocQuery (AdhocQueryRequest req) Submit an ad hoc query request.

482 6.5 Interfaces Exposed By Registry Clients

483 An ebXML Registry client implements the following interface.

484 6.5.1 Interface RegistryClient

485

486 This is the principal interface implemented by a Registry client. The client provides this
 487 interface when creating a connection to the Registry. It provides the methods that are
 488 used by the Registry to deliver asynchronous responses to the client. Note that a client
 489 need not provide a RegistryClient interface if the [CPA] between the client and the
 490 registry does not support asynchronous responses.

491 The registry sends all asynchronous responses to operations to the onResponse
 492 method.

493

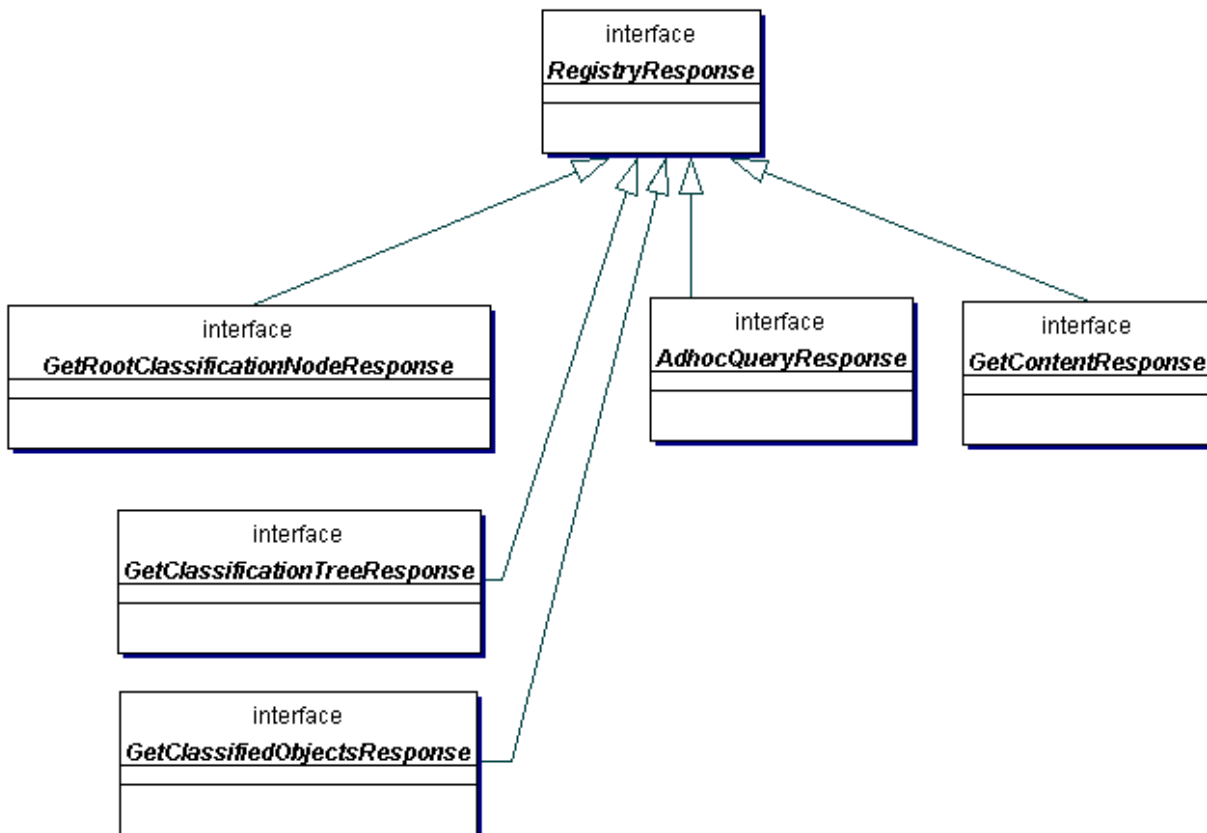
Method Summary of RegistryClient

void	onResponse (RegistryResponse resp) Notifies client of the response sent by registry to previously submitted request.
------	---

494

495 6.6 Registry Response Class Hierarchy

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



498

499

Figure 3: Registry Reponse Class Hierarchy

500

501

501

502

504 7 Object Management Service

505

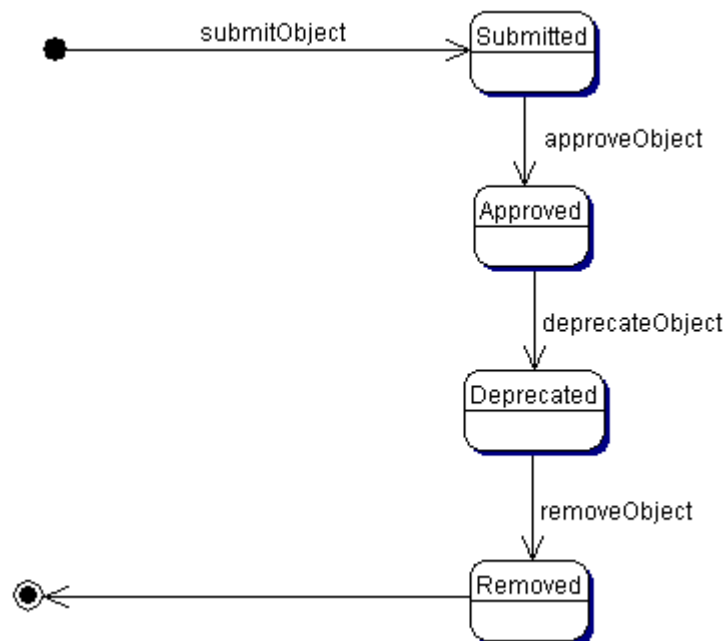
506 This section defines the ObjectManagement service of the Registry. The Object
507 Management Service is a sub-service of the Registry service. It provides the
508 functionality required by RegistryClients to manage the life cycle of repository items
509 (e.g. XML documents required for ebXML business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

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

516 7.1 Life Cycle of a Repository Item

517 The main purpose of the ObjectManagement service is to manage the life cycle of
518 repository items.

519 Figure 4 shows the typical life cycle of a repository item. Note that the current version of
520 this specification does not support Object versioning. Object versioning will be added in
521 a future version of this specification.



522

523

Figure 4: Life Cycle of a Repository Item**524 7.2 RegistryObject Attributes**

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

531 7.3 The Submit Objects Protocol

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

536



537

538

Figure 5: Submit Objects Sequence Diagram

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

541 The SubmitObjectRequest message includes a RegistrEntryList element.

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

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

548 In the event of success, the registry sends a RegistryResponse with a status of
549 “success” back to the client. In the event of failure, the registry sends a
550 RegistryResponse with a status of “failure” back to the client.

551 **7.3.1 Universally Unique ID Generation**

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

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

556 This id is usually generated by the registry. The `id` attribute for submitted objects may
557 optionally be supplied by the client. If the client supplies the `id` and it conforms to the
558 format of a URN that specifies a DCE 128 bit UUID then the registry assumes that the
559 client wishes to specify the `id` for the object. In this case, the registry must honor a
560 client-supplied `id` and use it as the `id` attribute of the object in the registry. If the `id` is
561 found by the registry to not be globally unique, the registry must raise the error
562 condition: `InvalidIdError`.

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

567 **7.3.2 ID Attribute And Object References**

568 The `id` attribute of an object may be used by other objects to reference the first object.
569 Such references are common both within the `SubmitObjectsRequest` as well as within
570 the registry. Within a `SubmitObjectsRequest`, the `id` attribute may be used to refer to an
571 object within the `SubmitObjectsRequest` as well as to refer to an object within the
572 registry. An object in the `SubmitObjectsRequest` that needs to be referred to within the
573 request document may be assigned an `id` by the submitter so that it can be referenced
574 within the request. The submitter may give the object a proper uuid URN, in which case
575 the `id` is permanently assigned to the object within the registry. Alternatively, the
576 submitter may assign an arbitrary `id` (not a proper uuid URN) as long as the `id` is unique
577 within the request document. In this case the `id` serves as a linkage mechanism within
578 the request document but must be ignored by the registry and replaced with a registry
579 generated `id` upon submission.

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

584 7.3.3 Sample SubmitObjectsRequest

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

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

```

591 <?xml version = "1.0" encoding = "UTF-8"?>
592 <!DOCTYPE SubmitObjectsRequest SYSTEM "file:///home/najmi/Registry.dtd">
593
594 <SubmitObjectsRequest>
595   <RegistryEntryList>
596
597     <!--
598     The following 3 objects package specified ExtrinsicObject in specified
599     Package, where both the Package and the ExtrinsicObject are
600     being submitted
601     -->
602     <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
603     <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
604       objectType = "CPP" name = "Widget Profile"
605       description = "ACME's profile for selling widgets"/>
606     <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
607       sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>
608
609     <!--
610     The following 3 objects package specified ExtrinsicObject in specified Package,
611     Where the Package is being submitted and the ExtrinsicObject is
612     already in registry
613     -->
614     <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
615     <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
616     <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
617       associationType = "Packages" sourceObject = "acmePackage2"
618       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
619
620     <!--
621     The following 3 objects package specified ExtrinsicObject in specified Package,
622     where the Package and the ExtrinsicObject are already in registry
623     -->
624     <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
625     <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
626     <!-- id is unspecified implying that registry must create a uuid for this object -->
627     <Association associationType = "Packages"
628       sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
629       targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
630
631     <!--
632     The following 3 objects externally link specified ExtrinsicObject using
633     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
634     are being submitted
635     -->
636     <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
637     <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
  
```

```
639     name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
640 <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
641     sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
642
643 <!--
644 The following 2 objects externally link specified ExtrinsicObject using specified
645 ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
646 is already in registry. Note that the targetObject points to an ObjectRef in a
647 previous line
648 -->
649 <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
650 <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
651     associationType = "ExternallyLinks" sourceObject = "acmeLink2"
652     targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
653
654 <!--
655 The following 2 objects externally identify specified ExtrinsicObject using specified
656 ExternalIdentifier, where the ExternalIdentifier is being submitted and the
657 ExtrinsicObject is already in registry. Note that the targetObject points to an
658 ObjectRef in a previous line
659 -->
660 <ExternalIdentifier id = "acmeDUNSID" name = "DUNS" description = "DUNS ID for ACME"
661     value = "13456789012"/>
662 <Association id = "acmeDUNSID-alreadySubmittedCPP-Assoc"
663     associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSID"
664     targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
665
666 <!--
667 The following show submission of a brand new classification scheme in its entirety
668 -->
669 <ClassificationNode id = "geographyNode" name = "Geography"
670     description = "The Geography scheme example from Registry Services Spec" />
671 <ClassificationNode id = "asiaNode" name = "Asia"
672     description = "The Asia node under the Geography node" parent="geographyNode" />
673 <ClassificationNode id = "japanNode" name = "Japan"
674     description="The Japan node under the Asia node" parent="asiaNode" />
675 <ClassificationNode id = "koreaNode" name = "Korea"
676     description="The Korea node under the Asia node" parent="asiaNode" />
677 <ClassificationNode id = "europeNode" name = "Europe"
678     description = "The Europe node under the Geography node" parent="geographyNode" />
679 <ClassificationNode id = "germanyNode" name = "Germany"
680     description="The Germany node under the Asia node" parent="europeNode" />
681 <ClassificationNode id = "northAmericaNode" name = "North America"
682     description = "The North America node under the Geography node"
683     parent="geographyNode" />
684 <ClassificationNode id = "usNode" name = "US"
685     description="The US node under the Asia node" parent="northAmericaNode" />
686
687 <!--
688 The following show submission of a Automotive sub-tree of ClassificationNodes that
689 gets added to an existing classification scheme named 'Industry'
690 that is already in the registry
691 -->
692 <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
693 <ClassificationNode id = "automotiveNode" name = "Automotive"
694     description = "The Automotive sub-tree under Industry scheme"
695     parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
696 <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
697     description = "The Parts Supplier node under the Automotive node"
698     parent="automotiveNode" />
699 <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
700     description = "The Engine Supplier node under the Automotive node"
701     parent="automotiveNode" />
702
703 <!--
704 The following show submission of 2 Classifications of an object that is already in
705 the registry using 2 ClassificationNodes. One ClassificationNode
706 is being submitted in this request (Japan) while the other is already in the registry.
707 -->
708 <Classification id = "japanClassification"
```

```

709     description = "Classifies object by /Geography/Asia/Japan node"
710     classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
711     classificationNode="japanNode" />
712   <Classification id = "classificationUsingExistingNode"
713     description = "Classifies object using a node in the registry"
714     classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
715     classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
716   <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />
717
718
719   </RegistryEntryList>
720 </SubmitObjectsRequest>

```

7.4 The Add Slots Protocol

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



725

726

Figure 7: Add Slots Sequence Diagram

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

7.5 The Remove Slots Protocol

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

732



733

734

Figure 8: Remove Slots Sequence Diagram

735 In the event of success, the registry sends a RegistryResponse with a status of
 736 “success” back to the client. In the event of failure, the registry sends a
 737 RegistryResponse with a status of “failure” back to the client.

738 7.6 The Approve Objects Protocol

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



743

744

Figure 9: Approve Objects Sequence Diagram

745 In the event of success, the registry sends a RegistryResponse with a status of
 746 “success” back to the client. In the event of failure, the registry sends a
 747 RegistryResponse with a status of “failure” back to the client.
 748 For details on the schema for the business documents shown in this process refer to
 749 Appendix A.

750 7.7 The Deprecate Objects Protocol

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



756

757

Figure 10: Deprecate Objects Sequence Diagram

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

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

763 7.8 The Remove Objects Protocol

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

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

770 **7.8.1 Deletion Scope DeleteRepositoryItemOnly**

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

774 **7.8.2 Deletion Scope DeleteAll**

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

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



782

783 **Figure 11: Remove Objects Sequence Diagram**

784 In the event of success, the registry sends a RegistryResponse with a status of
 785 “success” back to the client. In the event of failure, the registry sends a
 786 RegistryResponse with a status of “failure” back to the client.

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

789 **8 Object Query Management Service**

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

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

- 794 1. Browse and Drill Down Query
 795 2. Filtered Query
 796 3. SQL Query

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

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

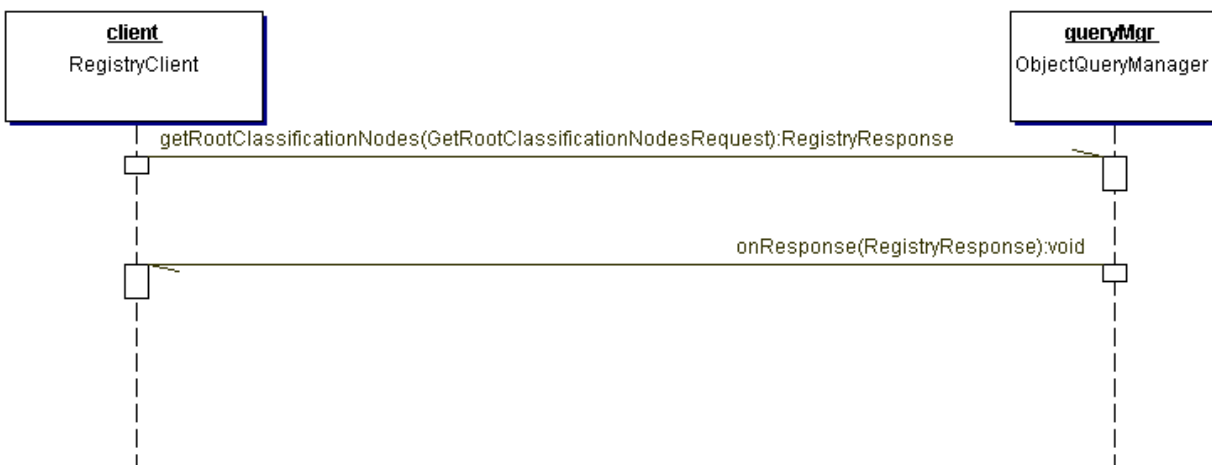
804 Any errors in the query request messages are indicated in the corresponding query
 805 response message.

806 8.1 Browse and Drill Down Query Support

807 The browse and drill down query style is supported by a set of interaction protocols
 808 between the ObjectQueryManagerClient and the ObjectQueryManager. Sections 8.1.1,
 809 8.1.2 and 8.1.3 describe these protocols.

810 8.1.1 Get Root Classification Nodes Request

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



817

818

Figure 12: Get Root Classification Nodes Sequence Diagram

819 In the event of success, the registry sends a `GetRootClassificationNodeResponse` with
 820 a status of “success” back to the client. In the event of failure, the registry sends a
 821 `GetRootClassificationNodeResponse` with a status of “failure” back to the client.

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

824 8.1.2 Get Classification Tree Request

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

831



832

833 **Figure 14: Get Classification Tree Sequence Diagram**

834 In the event of success, the registry sends a `GetClassificationTreeResponse` with a
 835 status of “success” back to the client. In the event of failure, the registry sends a
 836 `GetClassificationTreeResponse` with a status of “failure” back to the client.

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

839 8.1.3 Get Classified Objects Request

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

843 It is possible to get `RegistryEntries` based on matches with multiple classifications. Note
 844 that specifying a `ClassificationNode` is implicitly specifying a logical OR with all
 845 descendants of the specified `ClassificationNode`.

846 When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it should
847 return Objects that are:

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

850 8.1.3.1 Get Classified Objects Request Example



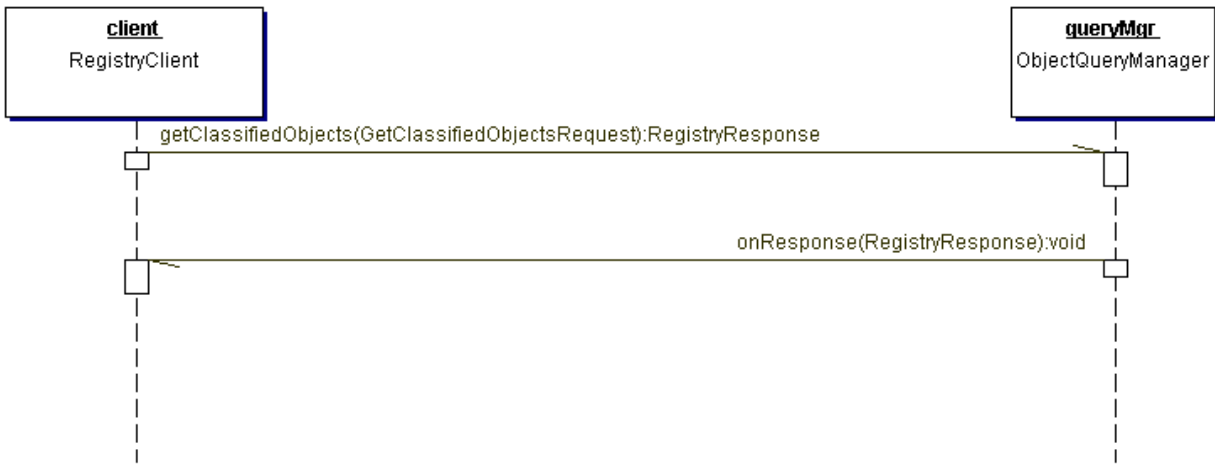
851

852

Figure 16: A Sample Geography Classification

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

- 854 • If the Geography node is specified in the GetClassifiedObjectsRequest then the
855 GetClassifiedObjectsResponse should include all RegistryEntries that are directly
856 classified by Geography *or* North America *or* US *or* Asia *or* Japan *or* Korea *or*
857 Europe *or* Germany.
- 858 • If the Asia node is specified in the GetClassifiedObjectsRequest then the
859 GetClassifiedObjectsResponse should include all RegistryEntries that are directly
860 classified by Asia *or* Japan *or* Korea.
- 861 • If the Japan *and* Korea nodes are specified in the GetClassifiedObjectsRequest
862 then the GetClassifiedObjectsResponse should include all RegistryEntries that
863 are directly classified by both Japan *and* Korea.
- 864 • If the North America *and* Asia node is specified in the
865 GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should
866 include all RegistryEntries that are directly classified by (North America *or* US)
867 *and* (Asia *or* Japan *or* Korea).



868

869

Figure 17: Get Classified Objects Sequence Diagram

870 In the event of success, the registry sends a `GetClassifiedObjectsResponse` with a
871 status of “success” back to the client. In the event of failure, the registry sends a
872 `GetClassifiedObjectsResponse` with a status of “failure” back to the client.

8.2 Filter Query Support

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

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

Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated with Z. The ebRIM Binding for C might be as in Figure 19.

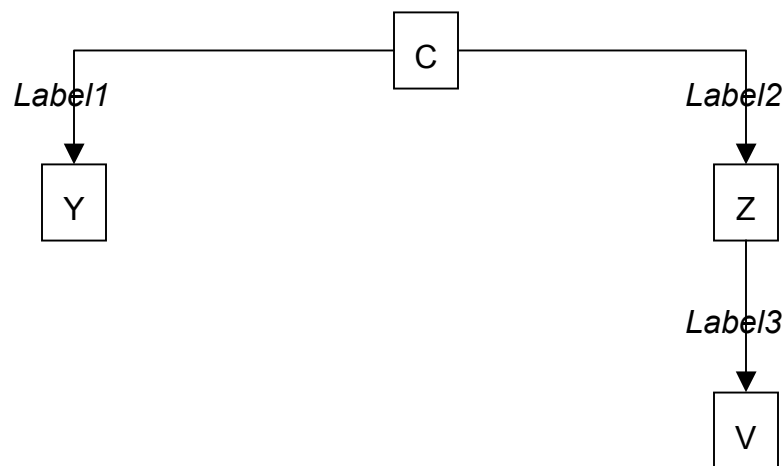


Figure 19: Example ebRIM Binding

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

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

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

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

928 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is
929 intended to be stable, the FilterQuery syntax is stable. However, if new structures are
930 added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the
931 same time.

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

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

938 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a
939 way to structure an XML document as an expansion of the result of a
940 RegistryEntryQuery. The ReturnRegistryEntry element specified in Section 8.2.7 allows
941 one to specify what metadata one wants returned with each registry entry identified in
942 the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section
943 8.2.8 allows one to specify what repository items one wants returned based on their
944 relationships to the registry entries identified by the result of a RegistryEntryQuery.

945

945 **8.2.1 FilterQuery**946 **Purpose**

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

951 **Definition**

```

952
953 <!ELEMENT FilterQuery
954   (
955     RegistryEntryQuery
956     | AuditableEventQuery
957     | ClassificationNodeQuery
958     | RegistryPackageQuery
959     | OrganizationQuery          )>
960
961 <!ELEMENT FilterQueryResult
962   (
963     RegistryEntryQueryResult
964     | AuditableEventQueryResult
965     | ClassificationNodeQueryResult
966     | RegistryPackageQueryResult
967     | OrganizationQueryResult  )>
968
969 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
970
971 <!ELEMENT RegistryEntryView EMPTY >
972 <!ATTLIST RegistryEntryView
973   objectURN      CDATA      #REQUIRED
974   contentURI     CDATA      #IMPLIED
975   objectID       CDATA      #IMPLIED >
976
977 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
978
979 <!ELEMENT AuditableEventView EMPTY >
980 <!ATTLIST AuditableEventView
981   objectID       CDATA      #REQUIRED
982   timestamp      CDATA      #REQUIRED >
983
984 <!ELEMENT ClassificationNodeQueryResult
985   (ClassificationNodeView*)>
986
987 <!ELEMENT ClassificationNodeView EMPTY >
988 <!ATTLIST ClassificationNodeView
989   objectURN      CDATA      #REQUIRED
990   contentURI     CDATA      #IMPLIED
991   objectID       CDATA      #IMPLIED >
992
993 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
994
995 <!ELEMENT RegistryPackageView EMPTY >
996 <!ATTLIST RegistryPackageView

```

```
995      objectURN      CDATA      #REQUIRED
996      contentURI      CDATA      #IMPLIED
997      objectID        CDATA      #IMPLIED >
998
999      <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
1000
1001      <!ELEMENT OrganizationView EMPTY >
1002      <!ATTLIST OrganizationView
1003          orgURN      CDATA      #REQUIRED
1004          objectID    CDATA      #IMPLIED >
1005
1006
```

1007 **Semantic Rules**

- 1008 1. The semantic rules for each FilterQuery alternative are specified in subsequent
1009 subsections.
- 1010 2. Each FilterQueryResult is a set of XML reference elements to identify each instance
1011 of the result set. Each XML attribute carries a value derived from the value of an
1012 attribute specified in the Registry Information Model as follows:
 - 1013 a) objectID is the value of the ID attribute of the RegistryObject class,
 - 1014 b) objectURN and orgURN are URN values derived from the object ID,
 - 1015 c) contentURI is a URL value derived from the contentURI attribute of the
1016 RegistryEntry class,
 - 1017 d) timestamp is a literal value to represent the value of the timestamp attribute of
1018 the AuditableEvent class.
- 1019 3. If an error condition is raised during any part of the execution of a FilterQuery, then
1020 the status attribute of the XML RegistryResult is set to “failure” and no query result
1021 element is returned; instead, a RegistryErrorList element must be returned with its
1022 highestSeverity element set to “error”. At least one of the RegistryError elements in
1023 the RegistryErrorList will have its severity attribute set to “error”.
- 1024 4. If no error conditions are raised during execution of a FilterQuery, then the status
1025 attribute of the XML RegistryResult is set to “success” and an appropriate query
1026 result element must be included. If a RegistryErrorList is also returned, then the
1027 highestSeverity attribute of the RegistryErrorList is set to “warning” and the severity
1028 attribute of each RegistryError is set to “warning”.
- 1029
- 1030
- 1031

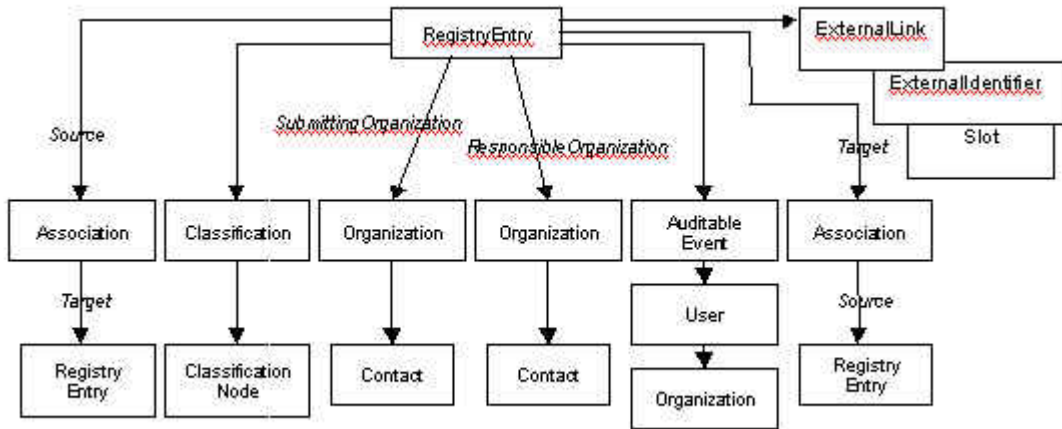
1031 **8.2.2 RegistryEntryQuery**

1032 **Purpose**

1033 To identify a set of registry entry instances as the result of a query over selected registry
 1034 metadata.

1035 **ebRIM Binding**

1036



1037

1038 **Definition**

1039

```

1040 <!ELEMENT RegistryEntryQuery
1041 ( RegistryEntryFilter?,
1042 SourceAssociationBranch*,
1043 TargetAssociationBranch*,
1044 HasClassificationBranch*,
1045 SubmittingOrganizationBranch?,
1046 ResponsibleOrganizationBranch?,
1047 ExternalIdentifierFilter*,
1048 ExternalLinkFilter*,
1049 SlotFilter*,
1050 HasAuditableEventBranch* )>
1051
1052 <!ELEMENT SourceAssociationBranch
1053 ( AssociationFilter?,
1054 RegistryEntryFilter? )>
1055
1056 <!ELEMENT TargetAssociationBranch
1057 ( AssociationFilter?,
1058 RegistryEntryFilter? )>
1059
1060 <!ELEMENT HasClassificationBranch
1061 ( ClassificationFilter?,
1062 ClassificationNodeFilter? )>
1063
    
```

```
1064 <!ELEMENT SubmittingOrganizationBranch
1065 ( OrganizationFilter?,
1066 ContactFilter? )>
1067
1068 <!ELEMENT ResponsibleOrganizationBranch
1069 ( OrganizationFilter?,
1070 ContactFilter? )>
1071
1072 <!ELEMENT HasAuditableEventBranch
1073 ( AuditableEventFilter?,
1074 UserFilter?,
1075 OrganizationFilter? )>
```

1076 Semantic Rules

- 1077 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The
1078 following steps will eliminate instances in RE that do not satisfy the conditions of the
1079 specified filters.
- 1080 a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below;
1081 otherwise, let x be a registry entry in RE. If x does not satisfy the
1082 RegistryEntryFilter as defined in Section 8.2.9, then remove x from RE.
- 1083 b) If a SourceAssociationBranch element is not specified, or if RE is empty, then
1084 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1085 source object of some Association instance, then remove x from RE; otherwise,
1086 treat each SourceAssociationBranch element separately as follows:
- 1087 If no AssociationFilter is specified within SourceAssociationBranch, then let AF
1088 be the set of all Association instances that have x as a source object; otherwise,
1089 let AF be the set of Association instances that satisfy the AssociationFilter and
1090 have x as the source object. If AF is empty, then remove x from RE. If no
1091 RegistryEntryFilter is specified within SourceAssociationBranch, then let RET be
1092 the set of all RegistryEntry instances that are the target object of some element
1093 of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the
1094 RegistryEntryFilter and are the target object of some element of AF. If RET is
1095 empty, then remove x from RE.
- 1096 c) If a TargetAssociationBranch element is not specified, or if RE is empty, then
1097 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1098 target object of some Association instance, then remove x from RE; otherwise,
1099 treat each TargetAssociationBranch element separately as follows:

- 1100 If no AssociationFilter is specified within TargetAssociationBranch, then let AF be
1101 the set of all Association instances that have x as a target object; otherwise, let
1102 AF be the set of Association instances that satisfy the AssociationFilter and have
1103 x as the target object. If AF is empty, then remove x from RE. If no
1104 RegistryEntryFilter is specified within TargetAssociationBranch, then let RES be
1105 the set of all RegistryEntry instances that are the source object of some element
1106 of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the
1107 RegistryEntryFilter and are the source object of some element of AF. If RES is
1108 empty, then remove x from RE.
- 1109 d) If a HasClassificationBranch element is not specified, or if RE is empty, then
1110 continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1111 source object of some Classification instance, then remove x from RE; otherwise,
1112 treat each HasClassificationBranch element separately as follows:
- 1113 If no ClassificationFilter is specified within the HasClassificationBranch, then let
1114 CL be the set of all Classification instances that have x as a source object;
1115 otherwise, let CL be the set of Classification instances that satisfy the
1116 ClassificationFilter and have x as the source object. If CL is empty, then remove
1117 x from RE. If no ClassificationNodeFilter is specified within
1118 HasClassificationBranch, then let CN be the set of all ClassificationNode
1119 instances that are the target object of some element of CL; otherwise, let CN be
1120 the set of RegistryEntry instances that satisfy the ClassificationNodeFilter and
1121 are the target object of some element of CL. If CN is empty, then remove x from
1122 RE.
- 1123 e) If a SubmittingOrganizationBranch element is not specified, or if RE is empty,
1124 then continue below; otherwise, let x be a remaining registry entry in RE. If x
1125 does not have a submitting organization, then remove x from RE. If no
1126 OrganizationFilter is specified within SubmittingOrganizationBranch, then let SO
1127 be the set of all Organization instances that are the submitting organization for x;
1128 otherwise, let SO be the set of Organization instances that satisfy the
1129 OrganizationFilter and are the submitting organization for x. If SO is empty, then
1130 remove x from RE. If no ContactFilter is specified within
1131 SubmittingOrganizationBranch, then let CT be the set of all Contact instances
1132 that are the contacts for some element of SO; otherwise, let CT be the set of
1133 Contact instances that satisfy the ContactFilter and are the contacts for some
1134 element of SO. If CT is empty, then remove x from RE.

- 1135 f) If a ResponsibleOrganizationBranch element is not specified, or if RE is empty,
1136 then continue below; otherwise, let x be a remaining registry entry in RE. If x
1137 does not have a responsible organization, then remove x from RE. If no
1138 OrganizationFilter is specified within ResponsibleOrganizationBranch, then let
1139 RO be the set of all Organization instances that are the responsible organization
1140 for x; otherwise, let RO be the set of Organization instances that satisfy the
1141 OrganizationFilter and are the responsible organization for x. If RO is empty, then
1142 remove x from RE. If no ContactFilter is specified within
1143 SubmittingOrganizationBranch, then let CT be the set of all Contact instances
1144 that are the contacts for some element of RO; otherwise, let CT be the set of
1145 Contact instances that satisfy the ContactFilter and are the contacts for some
1146 element of RO. If CT is empty, then remove x from RE.
- 1147 g) If an ExternalLinkFilter element is not specified, or if RE is empty, then continue
1148 below; otherwise, let x be a remaining registry entry in RE. If x is not linked to
1149 some ExternalLink instance, then remove x from RE; otherwise, treat each
1150 ExternalLinkFilter element separately as follows:
1151 Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter and
1152 are linked to x. If EL is empty, then remove x from RE.
- 1153 h) If an ExternalIdentifierFilter element is not specified, or if RE is empty, then
1154 continue below; otherwise, let x be a remaining registry entry in RE. If x is not
1155 linked to some ExternalIdentifier instance, then remove x from RE; otherwise,
1156 treat each ExternalIdentifierFilter element separately as follows:
1157 Let EI be the set of ExternalIdentifier instances that satisfy the
1158 ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RE.
- 1159 i) If a SlotFilter element is not specified, or if RE is empty, then continue below;
1160 otherwise, let x be a remaining registry entry in RE. If x is not linked to some Slot
1161 instance, then remove x from RE; otherwise, treat each SlotFilter element
1162 separately as follows:
1163 Let SL be the set of Slot instances that satisfy the SlotFilter and are linked to x. If
1164 SL is empty, then remove x from RE.
- 1165 j) If a HasAuditableEventBranch element is not specified, or if RE is empty, then
1166 continue below; otherwise, let x be a remaining registry entry in RE. If x is not
1167 linked to some AuditableEvent instance, then remove x from RE; otherwise, treat
1168 each HasAuditableEventBranch element separately as follows:
1169 If an AuditableEventFilter is not specified within HasAuditableEventBranch, then
1170 let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the
1171 set of AuditableEvent instances that satisfy the AuditableEventFilter and are
1172 auditable events for x. If AE is empty, then remove x from RE. If a UserFilter is
1173 not specified within HasAuditableEventBranch, then let AI be the set of all User
1174 instances linked to an element of AE; otherwise, let AI be the set of User
1175 instances that satisfy the UserFilter and are linked to an element of AE.

1176 If AI is empty, then remove x from RE. If an OrganizationFilter is not specified
 1177 within HasAuditableEventBranch, then let OG be the set of all Organization
 1178 instances that are linked to an element of AI; otherwise, let OG be the set of
 1179 Organization instances that satisfy the OrganizationFilter and are linked to an
 1180 element of AI. If OG is empty, then remove x from RE.

1181 2. If RE is empty, then raise the warning: *registry entry query result is empty*.

1182 3. Return RE as the result of the RegistryEntryQuery.

1183

1184 Examples

1185 A client wants to establish a trading relationship with XYZ Corporation and wants to
 1186 know if they have registered any of their business documents in the Registry. The
 1187 following query returns a set of registry entry identifiers for currently registered items
 1188 submitted by any organization whose name includes the string "XYZ". It does not return
 1189 any registry entry identifiers for superseded, replaced, deprecated, or withdrawn items.

```

1190
1191 <RegistryEntryQuery>
1192   <RegistryEntryFilter>
1193     status EQUAL "Approved"           -- code by Clause, Section 8.2.10
1194   </RegistryEntryFilter>
1195   <SubmittingOrganizationBranch>
1196     <OrganizationFilter>
1197       name CONTAINS "XYZ"             -- code by Clause, Section 8.2.10
1198     </OrganizationFilter>
1199   </SubmittingOrganizationBranch>
1200 </RegistryEntryQuery>
1201
```

1202 A client is using the United Nations Standard Product and Services Classification
 1203 (UNSPSC) scheme and wants to identify all companies that deal with products
 1204 classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client
 1205 knows that companies have registered their party profile documents in the Registry, and
 1206 that each profile has been classified by the products the company deals with. The
 1207 following query returns a set of registry entry identifiers for profiles of companies that
 1208 deal with integrated circuit components.

```

1209
1210 <RegistryEntryQuery>
1211   <RegistryEntryFilter>
1212     objectType EQUAL "CPP" AND         -- code by Clause, Section 8.2.10
1213     status EQUAL "Approved"
1214   </RegistryEntryFilter>
1215   <HasClassificationBranch>
1216     <ClassificationNodeFilter>
1217       id STARTSWITH "urn:un:spsc:321118" -- code by Clause, Section 8.2.10
1218     </ClassificationNodeFilter>
1219   </HasClassificationBranch>
1220 </RegistryEntryQuery>

```

1221 A client application needs all items that are classified by two different classification
1222 schemes, one based on "Industry" and another based on "Geography". Both schemes
1223 have been defined by ebXML and are registered. The root nodes of each scheme are
1224 identified by "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The
1225 following query identifies registry entries for all registered items that are classified by
1226 "Industry/Automotive" and by "Geography/Asia/Japan".

```
1227
1228 <RegistryEntryQuery>
1229   <HasClassificationBranch>
1230     <ClassificationNodeFilter>
1231       id STARTSWITH "urn:ebxml:cs:industry" AND
1232       path EQUAL "Industry/Automotive"      -- code by Clause, Section 8.2.10
1233     </ClassificationNodeFilter>
1234     <ClassificationNodeFilter>
1235       id STARTSWITH "urn:ebxml:cs:geography" AND
1236       path EQUAL "Geography/Asia/Japan"    -- code by Clause, Section 8.2.10
1237     </ClassificationNodeFilter>
1238   </HasClassificationBranch>
1239 </RegistryEntryQuery>
```

1240 A client application wishes to identify all registry Package instances that have a given
1241 registry entry as a member of the package. The following query identifies all registry
1242 packages that contain the registry entry identified by URN "urn:path:myitem" as a
1243 member:

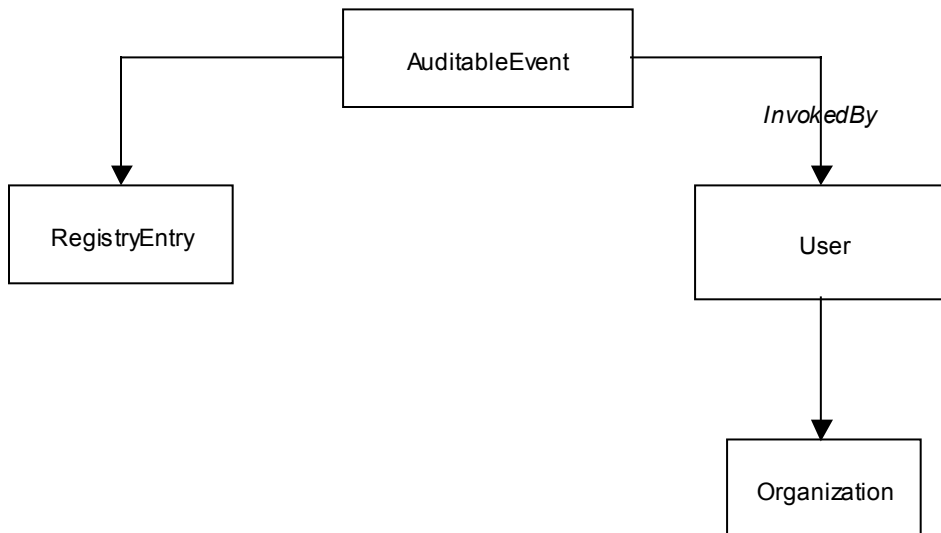
```
1244
1245 <RegistryEntryQuery>
1246   <RegistryEntryFilter>
1247     objectType EQUAL "RegistryPackage"      -- code by Clause, Section 8.2.10
1248   </RegistryEntryFilter>
1249   <SourceAssociationBranch>
1250     <AssociationFilter>                    -- code by Clause, Section 8.2.10
1251       associationType EQUAL "HasMember" AND
1252       targetObject EQUAL "urn:path:myitem"
1253     </AssociationFilter>
1254   </SourceAssociationBranch>
1255 </RegistryEntryQuery>
```

1256 A client application wishes to identify all ClassificationNode instances that have some
1257 given keyword as part of their name or description. The following query identifies all
1258 registry classification nodes that contain the keyword "transistor" as part of their name
1259 or as part of their description.

```
1260
1261 <RegistryEntryQuery>
1262   <RegistryEntryFilter>
1263     ObjectType="ClassificationNode" AND
1264     (name CONTAINS "transistor" OR        -- code by Clause, Section 8.2.10
1265      description CONTAINS "transistor")
1266   </RegistryEntryFilter>
1267 </RegistryEntryQuery>
1268
```

1268 **8.2.3 AuditableEventQuery**1269 **Purpose**

1270 To identify a set of auditable event instances as the result of a query over selected
 1271 registry metadata.

1272 **ebRIM Binding**1273 **Definition**

```

1274
1275 <!ELEMENT AuditableEventQuery
1276   ( AuditableEventFilter?,
1277     RegistryEntryQuery*,
1278     InvokedByBranch? )>
1279
1280 <!ELEMENT InvokedByBranch
1281   ( UserFilter?,
1282     OrganizationQuery? )>
  
```

1283

1284 **Semantic Rules**

1285 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
 1286 following steps will eliminate instances in AE that do not satisfy the conditions of the
 1287 specified filters.

1288

- 1289 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below;
1290 otherwise, let x be an auditable event in AE. If x does not satisfy the
1291 AuditableEventFilter as defined in Section 8.2.9, then remove x from AE.
- 1292 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue
1293 below; otherwise, let x be a remaining auditable event in AE. Treat each
1294 RegistryEntryQuery element separately as follows:
- 1295 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If
1296 x is not an auditable event for some registry entry in RE, then remove x from AE.
- 1297 c) If an InvokedByBranch element is not specified, or if AE is empty, then continue
1298 below; otherwise, let x be a remaining auditable event in AE.
- 1299 Let u be the user instance that invokes x. If a UserFilter element is specified within the
1300 InvokedByBranch, and if u does not satisfy that filter, then remove x from AE; otherwise,
1301 continue below.
- 1302 If an OrganizationQuery element is not specified within the InvokedByBranch,
1303 then continue below; otherwise, let OG be the set of Organization instances that
1304 are identified by the organization attribute of u and are in the result set of the
1305 OrganizationQuery. If OG is empty, then remove x from AE.
- 1306 2. If AE is empty, then raise the warning: *auditable event query result is empty*.
- 1307 3. Return AE as the result of the AuditableEventQuery.

1308

1309 Examples

1310 A Registry client has registered an item and it has been assigned a URN identifier
1311 "urn:path:myitem". The client is now interested in all events since the beginning of the
1312 year that have impacted that item. The following query will return a set of
1313 AuditableEvent identifiers for all such events.

```
1314  
1315 <AuditableEventquery>  
1316   <AuditableEventFilter>  
1317     timestamp GE "2001-01-01" AND -- code by Clause, Section 8.2.10  
1318     registryEntry EQUAL "urn:path:myitem"  
1319   </AuditableEventFilter>  
1320 </AuditableEventQuery>
```

1321

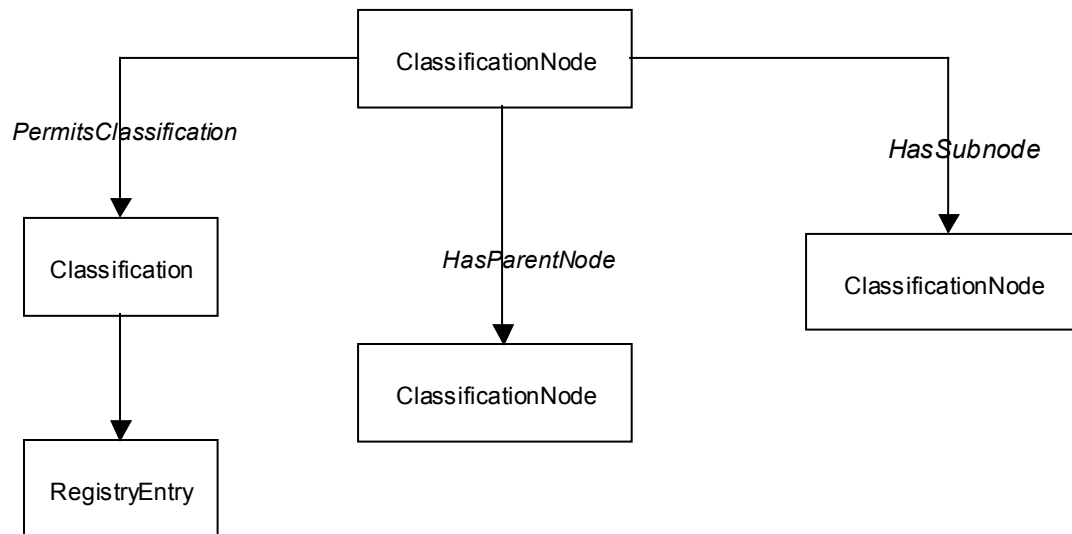
1322 A client company has many registered objects in the Registry. The Registry allows
1323 events submitted by other organizations to have an impact on your registered items,
1324 e.g. new classifications and new associations. The following query will return a set of
1325 identifiers for all auditable events, invoked by some other party, that had an impact on
1326 an item submitted by "myorg" and for which "myorg" is the responsible organization.

```
1327  
1328 <AuditableEventQuery>  
1329   <RegistryEntryQuery>
```

```
1330     <SubmittingOrganizationBranch>
1331         <OrganizationFilter>
1332             id EQUAL "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1333         </OrganizationFilter>
1334     </SubmittingOrganizationBranch>
1335     <ResponsibleOrganizationBranch>
1336         <OrganizationFilter>
1337             id EQUAL "urn:somepath:myorg"           -- code by Clause, Section 8.2.10
1338         </OrganizationFilter>
1339     </ResponsibleOrganizationBranch>
1340 </RegistryEntryQuery>
1341 <InvokedByBranch>
1342     <OrganizationQuery>
1343         <OrganizationFilter>
1344             id -EQUAL "urn:somepath:myorg"         -- code by Clause, Section 8.2.10
1345         </OrganizationFilter>
1346     </OrganizationQuery>
1347 </InvokedByBranch>
1348 </AuditableEventQuery>
1349
```

1349 **8.2.4 ClassificationNodeQuery**1350 **Purpose**

1351 To identify a set of classification node instances as the result of a query over selected
 1352 registry metadata.

1353 **ebRIM Binding**1354 **Definition**

```

1355 <!--ELEMENT ClassificationNodeQuery
1356   ( ClassificationNodeFilter?,
1357     PermitsClassificationBranch*,
1358     HasParentNode?,
1359     HasSubnode*           )>
1361
1362 <!--ELEMENT PermitsClassificationBranch
1363   ( ClassificationFilter?,
1364     RegistryEntryQuery?   )>
1365
1366 <!--ELEMENT HasParentNode
1367   ( ClassificationNodeFilter?,
1368     HasParentNode?       )>
1369
1370 <!--ELEMENT HasSubnode
1371   ( ClassificationNodeFilter?,
1372     HasSubnode*         )>
  
```

1373

1374

1375 **Semantic Rules**

- 1376 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry.
1377 The following steps will eliminate instances in CN that do not satisfy the conditions of
1378 the specified filters.
- 1379 a) If a ClassificationNodeFilter is not specified, or if CN is empty, then continue
1380 below; otherwise, let x be a classification node in CN. If x does not satisfy the
1381 ClassificationNodeFilter as defined in Section 8.2.9, then remove x from AE.
- 1382 b) If a PermitsClassificationBranch element is not specified, or if CN is empty, then
1383 continue below; otherwise, let x be a remaining classification node in CN. If x is
1384 not the target object of some Classification instance, then remove x from CN;
1385 otherwise, treat each PermitsClassificationBranch element separately as follows:
- 1386 If no ClassificationFilter is specified within the PermitsClassificationBranch
1387 element, then let CL be the set of all Classification instances that have x as the
1388 target object; otherwise, let CL be the set of Classification instances that satisfy
1389 the ClassificationFilter and have x as the target object. If CL is empty, then
1390 remove x from CN. If no RegistryEntryQuery is specified within the
1391 PermitsClassificationBranch element, then let RES be the set of all RegistryEntry
1392 instances that are the source object of some classification instance in CL;
1393 otherwise, let RE be the result set of the RegistryEntryQuery as defined in
1394 Section 8.2.2 and let RES be the set of all instances in RE that are the source
1395 object of some classification in CL. If RES is empty, then remove x from CN.
- 1396 c) If a HasParentNode element is not specified, or if CN is empty, then continue
1397 below; otherwise, let x be a remaining classification node in CN and execute the
1398 following paragraph with $n=x$.
- 1399 Let n be a classification node instance. If n does not have a parent node (i.e. if n
1400 is a root node), then remove x from CN. Let p be the parent node of n. If a
1401 ClassificationNodeFilter element is directly contained in HasParentNode and if p
1402 does not satisfy the ClassificationNodeFilter, then remove x from CN.
- 1403 If another HasParentNode element is directly contained within this
1404 HasParentNode element, then repeat the previous paragraph with $n=p$.
- 1405 d) If a HasSubnode element is not specified, or if CN is empty, then continue below;
1406 otherwise, let x be a remaining classification node in CN. If x is not the parent
1407 node of some ClassificationNode instance, then remove x from CN; otherwise,
1408 treat each HasSubnode element separately and execute the following paragraph
1409 with $n = x$.
- 1410 Let n be a classification node instance. If a ClassificationNodeFilter is not
1411 specified within the HasSubnode element then let CNC be the set of all
1412 classification nodes that have n as their parent node; otherwise, let CNC be the
1413 set of all classification nodes that satisfy the ClassificationNodeFilter and have n
1414 as their parent node. If CNC is empty then remove x from CN; otherwise, let y be
1415 an element of CNC and continue with the next paragraph.

1416 If the HasSubnode element is terminal, i.e. if it does not directly contain another
1417 HasSubnode element, then continue below; otherwise, repeat the previous
1418 paragraph with the new HasSubnode element and with $n = y$.

1419 2. If CN is empty, then raise the warning: *classification node query result is empty*.

1420 3. Return CN as the result of the ClassificationNodeQuery.

1421

1422 Examples

1423 A client application wishes to identify all classification nodes defined in the Registry that
1424 are root nodes and have a name that contains the phrase “product code” or the phrase
1425 “product type”. Note: By convention, if a classification node has no parent (i.e. is a root
1426 node), then the parent attribute of that instance is set to null and is represented as a
1427 literal by a zero length string.

1428

```
1429 <ClassificationNodeQuery>
1430   <ClassificationNodeFilter>
1431     (name CONTAINS "product code" OR      -- code by Clause, Section 8.2.10
1432      name CONTAINS "product type") AND
1433     parent EQUAL ""
1434   </ClassificationNodeFilter>
1435 </ClassificationNodeQuery>
```

1436

1437 A client application wishes to identify all of the classification nodes at the third level of a
1438 classification scheme hierarchy. The client knows that the URN identifier for the root
1439 node is “urn:ebxml:cs:myroot”. The following query identifies all nodes at the second
1440 level under “myroot” (i.e. third level overall).

1441

```
1442 <ClassificationNodeQuery>
1443   <HasParentNode>
1444     <HasParentNode>
1445       <ClassificationNodeFilter>
1446         id EQ "urn:ebxml:cs:myroot" -- code by Clause, Section 8.2.10
1447       </ClassificationNodeFilter>
1448     </HasParentNode>
1449   </HasParentNode>
1450 </ClassificationNodeQuery>
```

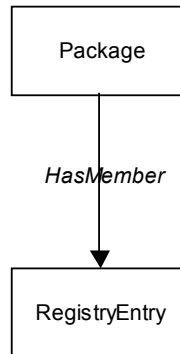
1451

1451 8.2.5 RegistryPackageQuery

1452 Purpose

1453 To identify a set of registry package instances as the result of a query over selected
1454 registry metadata.

1455 ebRIM Binding



1456 Definition

```

1457
1458 <!ELEMENT RegistryPackageQuery
1459   ( PackageFilter?,
1460     HasMemberBranch*   )>
1461
1462 <!ELEMENT HasMemberBranch
1463   ( RegistryEntryQuery?   )>
  
```

1464

1465 Semantic Rules

1466 1. Let RP denote the set of all persistent Package instances in the Registry. The
1467 following steps will eliminate instances in RP that do not satisfy the conditions of the
1468 specified filters.

- 1469 a) If a PackageFilter is not specified, or if RP is empty, then continue below;
1470 otherwise, let x be a package instance in RP. If x does not satisfy the
1471 PackageFilter as defined in Section 8.2.9, then remove x from RP.
- 1472 b) If a HasMemberBranch element is not directly contained in the
1473 RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x
1474 be a remaining package instance in RP. If x is an empty package, then remove x
1475 from RP; otherwise, treat each HasMemberBranch element separately as
1476 follows:

1477

1478 If a RegistryEntryQuery element is not directly contained in the
1479 HasMemberBranch element, then let PM be the set of all RegistryEntry instances
1480 that are members of the package x; otherwise, let RE be the set of RegistryEntry
1481 instances returned by the RegistryEntryQuery as defined in Section 8.2.2 and let
1482 PM be the subset of RE that are members of the package x. If PM is empty, then
1483 remove x from RP.

1484 2. If RP is empty, then raise the warning: *registry package query result is empty*.

1485 3. Return RP as the result of the RegistryPackageQuery.

1486

1487 Examples

1488 A client application wishes to identify all package instances in the Registry that contain
1489 an Invoice extrinsic object as a member of the package.

1490

```
1491 <RegistryPackageQuery>  
1492   <HasMemberBranch>  
1493     <RegistryEntryQuery>  
1494       <RegistryEntryFilter>  
1495         objectType EQ "Invoice"      -- code by Clause, Section 8.2.10  
1496       </RegistryEntryFilter>  
1497     </RegistryEntryQuery>  
1498   </HasMemberBranch>  
1499 </RegistryPackageQuery>
```

1500

1501 A client application wishes to identify all package instances in the Registry that are not
1502 empty.

1503

```
1504 <RegistryEntryQuery>  
1505   <HasMemberBranch/>  
1506 </RegistryEntryQuery>
```

1507

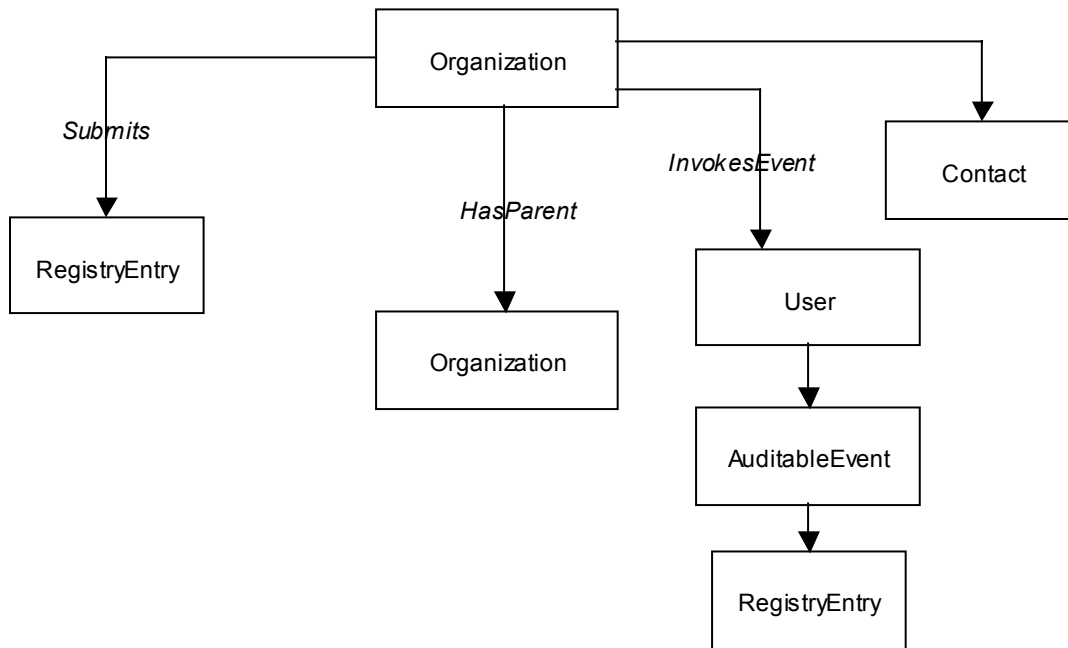
1508 A client application wishes to identify all package instances in the Registry that are
1509 empty. Since the RegistryPackageQuery is not set up to do negations, clients will have
1510 to do two separate RegistryPackageQuery requests, one to find all packages and
1511 another to find all non-empty packages, and then do the set difference themselves.
1512 Alternatively, they could do a more complex RegistryEntryQuery and check that the
1513 packaging association between the package and its members is non-existent.

1514 Note: A registry package is an intrinsic RegistryEntry instance that is completely
1515 determined by its associations with its members. Thus a RegistryPackageQuery can
1516 always be re-specified as an equivalent RegistryEntryQuery using appropriate "Source"
1517 and "Target" associations. However, the equivalent RegistryEntryQuery is often more
1518 complicated to write.

1519

1519 **8.2.6 OrganizationQuery**1520 **Purpose**

1521 To identify a set of organization instances as the result of a query over selected registry
 1522 metadata.

1523 **ebRIM Binding**

1524

1525 **Definition**

```

1526
1527 <!ELEMENT OrganizationQuery
1528   (   OrganizationFilter?,
1529       SubmitsRegistryEntry*,
1530       HasParentOrganization?,
1531       InvokesEventBranch*,
1532       ContactFilter
1533   )>
1534
1535 <!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>
1536
1537 <!ELEMENT HasParentOrganization
1538   (   OrganizationFilter?,
1539       HasParentOrganization?
1540   )>
1541
1542 <!ELEMENT InvokesEventBranch
1543   (   UserFilter?,
1544       AuditableEventFilter?,
1545       RegistryEntryQuery?
1546   )>
  
```

1544 **Semantic Rules**

- 1545 1. Let ORG denote the set of all persistent Organization instances in the Registry. The
1546 following steps will eliminate instances in ORG that do not satisfy the conditions of
1547 the specified filters.
- 1548 a) If an OrganizationFilter element is not directly contained in the
1549 OrganizationQuery element, or if ORG is empty, then continue below; otherwise,
1550 let x be an organization instance in ORG. If x does not satisfy the
1551 OrganizationFilter as defined in Section 8.2.9, then remove x from RP.
- 1552 b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery,
1553 or if ORG is empty, then continue below; otherwise, consider each
1554 SubmitsRegistryEntry element separately as follows:
- 1555 If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element,
1556 then let RES be the set of all RegistryEntry instances that have been submitted
1557 to the Registry by organization x; otherwise, let RE be the result of the
1558 RegistryEntryQuery as defined in Section 8.2.2 and let RES be the set of all
1559 instances in RE that have been submitted to the Registry by organization x. If
1560 RES is empty, then remove x from ORG.
- 1561 c) If a HasParentOrganization element is not specified within the
1562 OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute
1563 the following paragraph with o = x:
- 1564 Let o be an organization instance. If an OrganizationFilter is not specified within
1565 the HasParentOrganization and if o has no parent (i.e. if o is a root organization
1566 in the Organization hierarchy), then remove x from ORG; otherwise, let p be the
1567 parent organization of o. If p does not satisfy the OrganizationFilter, then remove
1568 x from ORG.
- 1569 If another HasParentOrganization element is directly contained within this
1570 HasParentOrganization element, then repeat the previous paragraph with o = p.
- 1571 d) If an InvokesEventBranch element is not specified within the OrganizationQuery,
1572 or if ORG is empty, then continue below; otherwise, consider each
1573 InvokesEventBranch element separately as follows:
- 1574 If an UserFilter is not specified, and if x is not the submitting organization of some
1575 AuditableEvent instance, then remove x from ORG. If an AuditableEventFilter is
1576 not specified, then let AE be the set of all AuditableEvent instances that have x
1577 as the submitting organization; otherwise, let AE be the set of AuditableEvent
1578 instances that satisfy the AuditableEventFilter and have x as the submitting
1579 organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is
1580 not specified in the InvokesEventBranch element, then let RES be the set of all
1581 RegistryEntry instances associated with an event in AE; otherwise, let RE be the
1582 result set of the RegistryEntryQuery, as specified in Section 8.2.2, and let RES
1583 be the subset of RE of entries submitted by x. If RES is empty, then remove x
1584 from ORG.

1585 e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG is
 1586 empty, then continue below; otherwise, consider each ContactFilter separately as
 1587 follows:

1588 Let CT be the set of Contact instances that satisfy the ContactFilter and are the
 1589 contacts for organization x. If CT is empty, then remove x from ORG.

1590 2. If ORG is empty, then raise the warning: *organization query result is empty*.

1591 3. Return ORG as the result of the OrganizationQuery.

1592

1593 Examples

1594 A client application wishes to identify a set of organizations, based in France, that have
 1595 submitted a PartyProfile extrinsic object this year.

```

1596 <OrganizationQuery>
1597   <OrganizationFilter>
1598     country EQUAL "France"           -- code by Clause, Section 8.2.10
1599   </OrganizationFilter>
1600   <SubmitsRegistryEntry>
1601     <RegistryEntryQuery>
1602       <RegistryEntryFilter>
1603         objectType EQUAL "CPP"      -- code by Clause, Section 8.2.10
1604       </RegistryEntryFilter>
1605       <HasAuditableEventBranch>
1606         <AuditableEventFilter>
1607           timestamp GE "2001-01-01" -- code by Clause, Section 8.2.10
1608         </AuditableEventFilter>
1609       </HasAuditableEventBranch>
1610     </RegistryEntryQuery>
1611   </SubmitsRegistryEntry>
1612 </OrganizationQuery>
  
```

1614

1615 A client application wishes to identify all organizations that have XYZ, Corporation as a
 1616 parent. The client knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is
 1617 no guarantee that subsidiaries of XYZ have a URN that uses the same format, so a full
 1618 query is required.

```

1619 <OrganizationQuery>
1620   <HasParentOrganization>
1621     <OrganizationFilter>
1622       id EQUAL "urn:ebxml:org:xyz"  -- code by Clause, Section 8.2.10
1623     </OrganizationFilter>
1624   </HasParentOrganization>
1625 </OrganizationQuery>
  
```

1627

1627 8.2.7 ReturnRegistryEntry

1628 Purpose

1629 To construct an XML document that contains selected registry metadata associated with
 1630 the registry entries identified by a RegistryEntryQuery. NOTE: Initially, the
 1631 RegistryEntryQuery could be the URN identifier for a single registry entry.

1632 Definition

```

1633
1634 <!ELEMENT ReturnRegistryEntry
1635   ( RegistryEntryQuery,
1636     WithClassifications?,
1637     WithSourceAssociations?,
1638     WithTargetAssociations?,
1639     WithAuditableEvents?,
1640     WithExternalLinks? )>
1641
1642 <!ELEMENT WithClassifications ( ClassificationFilter? )>
1643 <!ELEMENT WithSourceAssociations ( AssociationFilter? )>
1644 <!ELEMENT WithTargetAssociations ( AssociationFilter? )>
1645 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>
1646 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
1647
1648 <!ELEMENT ReturnRegistryEntryResult
1649   ( RegistryEntryMetadata*)>
1650
1651 <!ELEMENT RegistryEntryMetadata
1652   ( RegistryEntry,
1653     Classification*,
1654     SourceAssociations?,
1655     TargetAssociations?,
1656     AuditableEvent*,
1657     ExternalLink* )>
1658
1659 <!ELEMENT SourceAssociations ( Association* )>
1660 <!ELEMENT TargetAssociations ( Association* )>

```

1661 Semantic Rules

- 1662 1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink
 1663 elements contained in the ReturnRegistryEntryResult are defined by the ebXML
 1664 Registry DTD specified in Appendix A.
- 1665 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in
 1666 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let
 1667 S be the set of warnings and errors returned. If any element in S is an error
 1668 condition, then stop execution and return the same set of warnings and errors along
 1669 with the ReturnRegistryEntryResult.

- 1670 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the
1671 ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*.
1672 Add this warning to the error list returned by the RegistryEntryQuery and return this
1673 enhanced error list with the ReturnRegistryEntryResult.
- 1674 4. For each registry entry E referenced by an element of R, use the attributes of E to
1675 create a new RegistryEntry element as defined in Appendix A. Then create a new
1676 RegistryEntryMetadata element as defined above to be the parent element of that
1677 RegistryEntry element.
- 1678 5. If no With option is specified, then the resulting RegistryEntryMetadata element has
1679 no Classification, SourceAssociations, TargetAssociations, AuditableEvent, or
1680 ExternalData subelements. The set of RegistryEntryMetadata elements, with the
1681 Error list from the RegistryEntryQuery, is returned as the ReturnRegistryEntryResult.
- 1682 6. If WithClassifications is specified, then for each E in R do the following: If a
1683 ClassificationFilter is not present, then let C be any classification instance linked to
1684 E; otherwise, let C be a classification instance linked to E that satisfies the
1685 ClassificationFilter (Section 8.2.9). For each such C, create a new Classification
1686 element as defined in Appendix A. Add these Classification elements to their parent
1687 RegistryEntryMetadata element.
- 1688 7. If WithSourceAssociations is specified, then for each E in R do the following: If an
1689 AssociationFilter is not present, then let A be any association instance whose source
1690 object is E; otherwise, let A be an association instance that satisfies the
1691 AssociationFilter (Section 8.2.9) and whose source object is E. For each such A,
1692 create a new Association element as defined in Appendix A. Add these Association
1693 elements as subelements of the WithSourceAssociations and add that element to its
1694 parent RegistryEntryMetadata element.
- 1695 8. If WithTargetAssociations is specified, then for each E in R do the following: If an
1696 AssociationFilter is not present, then let A be any association instance whose target
1697 object is E; otherwise, let A be an association instance that satisfies the
1698 AssociationFilter (Section 8.2.9) and whose target object is E. For each such A,
1699 create a new Association element as defined in Appendix A. Add these Association
1700 elements as subelements of the WithTargetAssociations and add that element to its
1701 parent RegistryEntryMetadata element.
- 1702 9. If WithAuditableEvents is specified, then for each E in R do the following: If an
1703 AuditableEventFilter is not present, then let A be any auditable event instance linked
1704 to E; otherwise, let A be any auditable event instance linked to E that satisfies the
1705 AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent
1706 element as defined in Appendix A. Add these AuditableEvent elements to their
1707 parent RegistryEntryMetadata element.

- 1708 10. If WithExternalLinks is specified, then for each E in R do the following: If an
1709 ExternalLinkFilter is not present, then let L be any external link instance linked to E;
1710 otherwise, let L be any external link instance linked to E that satisfies the
1711 ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink
1712 element as defined in Appendix A. Add these ExternalLink elements to their parent
1713 RegistryEntryMetadata element.
- 1714 11. If any warning or error condition results, then add the code and the message to the
1715 RegistryResponse element that includes the RegistryEntryQueryResult.
- 1716 12. Return the set of RegistryEntryMetadata elements as the content of the
1717 ReturnRegistryEntryResult.

1718

1719 Examples

1720 A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by
1721 XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer wishes to
1722 check on the current status of that DTD, especially if it has been superceded or
1723 replaced, and get all of its current classifications. The following query request will return
1724 an XML document with the registry entry for the existing DTD as the root, with all of its
1725 classifications, and with associations to registry entries for any items that have
1726 superceded or replaced it.

```
1727  
1728 <ReturnRegistryEntry>  
1729   <RegistryEntryQuery>  
1730     <RegistryEntryFilter>  
1731       id EQUAL "urn:com:xyz:po:325"           -- code by Clause, Section 8.2.10  
1732     </RegistryEntryFilter>  
1733   </RegistryEntryQuery>  
1734   <WithClassifications/>  
1735   <WithSourceAssociations>  
1736     <AssociationFilter>                       -- code by Clause, Section 8.2.10  
1737       associationType EQUAL "SupercededBy" OR  
1738       associationType EQUAL "ReplacedBy"  
1739     </AssociationFilter>  
1740   </WithSourceAssociations>  
1741 </ReturnRegistryEntry>
```

1742

1743 A client of the Registry registered an XML DTD several years ago and is now thinking of
1744 replacing it with a revised version. The identifier for the existing DTD is
1745 "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the
1746 existing DTD. The client desires a list of all registered items that use the existing DTD
1747 so they can assess the impact of an incompatible change. The following query returns
1748 an XML document that is a list of all RegistryEntry elements that represent registered
1749 items that use, contain, or extend the given DTD. The document also links each
1750 RegistryEntry element in the list to an element for the identified association.

1751

```

1752
1753 <ReturnRegistryEntry>
1754   <RegistryEntryQuery>
1755     <SourceAssociationBranch>
1756       <AssociationFilter>           -- code by Clause, Section 8.2.10
1757         associationType EQUAL "Contains" OR
1758         associationType EQUAL "Uses" OR
1759         associationType EQUAL "Extends"
1760       </AssociationFilter>
1761       <RegistryEntryFilter>       -- code by Clause, Section 8.2.10
1762         id EQUAL "urn:xyz:dtd:po97"
1763       </RegistryEntryFilter>
1764     </SourceAssociationBranch>
1765   </RegistryEntryQuery>
1766   <WithSourceAssociations>
1767     <AssociationFilter>           -- code by Clause, Section 8.2.10
1768       associationType EQUAL "Contains" OR
1769       associationType EQUAL "Uses" OR
1770       associationType EQUAL "Extends"
1771     </AssociationFilter>
1772   </WithSourceAssociations>
1773 </ReturnRegistryEntry>

```

1774

1775 A user has been browsing the registry and has found a registry entry that describes a
1776 package of core-components that should solve the user's problem. The package URN
1777 identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to know what's in the package.
1778 The following query returns an XML document with a registry entry for each member of
1779 the package along with that member's Uses and HasMemberBranch associations.

```

1780
1781 <ReturnRegistryEntry>
1782   <RegistryEntryQuery>
1783     <TargetAssociationBranch>
1784       <AssociationFilter>           -- code by Clause, Section 8.2.10
1785         associationType EQUAL "HasMember"
1786       </AssociationFilter>
1787       <RegistryEntryFilter>       -- code by Clause, Section 8.2.10
1788         id EQUAL " urn:com:cc:pkg:ccstuff "
1789       </RegistryEntryFilter>
1790     </TargetAssociationBranch>
1791   </RegistryEntryQuery>
1792   <WithSourceAssociations>
1793     <AssociationFilter>           -- code by Clause, Section 8.2.10
1794       associationType EQUAL "HasMember" OR
1795       associationType EQUAL "Uses"
1796     </AssociationFilter>
1797   </WithSourceAssociations>
1798 </ReturnRegistryEntry>
1799

```

1799 **8.2.8 ReturnRepositoryItem**1800 **Purpose**

1801 To construct an XML document that contains one or more repository items, and some
 1802 associated metadata, by submitting a RegistryEntryQuery to the registry/repository that
 1803 holds the desired objects. NOTE: Initially, the RegistryEntryQuery could be the URN
 1804 identifier for a single registry entry.

1805 **Definition**

```

1806
1807 <!ELEMENT ReturnRepositoryItem
1808 ( RegistryEntryQuery,
1809 RecursiveAssociationOption?,
1810 WithDescription? )>
1811
1812 <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1813 <!ATTLIST RecursiveAssociationOption
1814 depthLimit CDATA #IMPLIED >
1815
1816 <!ELEMENT AssociationType EMPTY >
1817 <!ATTLIST AssociationType
1818 role CDATA #REQUIRED >
1819
1820 <!ELEMENT WithDescription EMPTY >
1821
1822 <!ELEMENT ReturnRepositoryItemResult
1823 ( RepositoryItem*)>
1824
1825 <!ELEMENT RepositoryItem
1826 ( ClassificationScheme
1827 | RegistryPackage
1828 | ExtrinsicObject
1829 | WithdrawnObject
1830 | ExternalLinkItem )>
1831 <!ATTLIST RepositoryItem
1832 identifier CDATA #REQUIRED
1833 name CDATA #REQUIRED
1834 contentURI CDATA #REQUIRED
1835 objectType CDATA #REQUIRED
1836 status CDATA #REQUIRED
1837 stability CDATA #REQUIRED
1838 description CDATA #IMPLIED >
1839
1840 <!ELEMENT ExtrinsicObject (#PCDATA) >
1841 <!ATTLIST ExtrinsicObject
1842 byteEncoding CDATA "Base64" >
1843
1844 <!ELEMENT WithdrawnObject EMPTY >
1845
1846 <!ELEMENT ExternalLinkItem EMPTY >
1847
1848
1849

```

1850 **Semantic Rules**

- 1851 1. If the RecursiveOption element is not present , then set Limit=0. If the
1852 RecursiveOption element is present, interpret its depthLimit attribute as an integer
1853 literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0
1854 means that no recursion occurs. A Limit of -1 means that recursion occurs
1855 indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive
1856 integer, then stop execution and raise the exception: *invalid depth limit*; otherwise,
1857 set Limit=N, where N is that positive integer. A Limit of N means that exactly N
1858 recursive steps will be executed unless the process terminates prior to that limit.
- 1859 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned
1860 as part of the ReturnRepositoryItemResult. Initially Result is empty. Semantic rules
1861 4 through 10 determine the content of Result.
- 1862 3. If the WithDescription element is present, then set WSD="yes"; otherwise, set
1863 WSD="no".
- 1864 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in
1865 Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let
1866 S be the set of warnings and errors returned. If any element in S is an error
1867 condition, then stop execution and return the same set of warnings and errors along
1868 with the ReturnRepositoryItemResult.
- 1869 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from
1870 R. After execution of these rules, if Depth is now equal to Limit, then return the
1871 content of Result as the set of RepositoryItem elements in the
1872 ReturnRepositoryItemResult element; otherwise, continue with Semantic Rule 8.
- 1873 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the
1874 following:
- 1875 a) If E.contentURI references a repository item in this registry/repository, then
1876 create a new RepositoryItem element, with values for its attributes derived as
1877 specified in Semantic Rule 7.
- 1878 1) If E.objectType="ClassificationScheme", then put the referenced
1879 ClassificationScheme DTD as the subelement of this RepositoryItem.
1880 [NOTE: Requires DTD specification!]
- 1881 2) If E.objectType="RegistryPackage", then put the referenced
1882 RegistryPackage DTD as the subelement of this RepositoryItem. [NOTE:
1883 Requires DTD specification!]
- 1884 3) Otherwise, i.e., if the object referenced by E has an unknown internal
1885 structure, then put the content of the repository item as the #PCDATA of a
1886 new ExtrinsicObject subelement of this RepositoryItem.
- 1887 b) If E.objectURL references a registered object in some other registry/repository,
1888 then create a new RepositoryItem element, with values for its attributes derived
1889 as specified in Semantic Rule 7, and create a new ExternalLink element as the
1890 subelement of this RepositoryItem.

- 1891 c) If E.objectURL is void, i.e. the object it would have referenced has been
1892 withdrawn, then create a new RepositoryItem element, with values for its
1893 attributes derived as specified in Semantic Rule 7, and create a new
1894 WithdrawnObject element as the subelement of this RepositoryItem.
- 1895 7. Let E be a registry entry and let RO be the RepositoryItem element created in
1896 Semantic Rule 6. Set the attributes of RO to the values derived from the
1897 corresponding attributes of E. If WSD="yes", include the value of the description
1898 attribute; otherwise, do not include it. Insert this new RepositoryItem element into the
1899 Result set.
- 1900 8. Let R be defined as in Semantic Rule 3. Execute Semantic Rule 9 with Y as the set
1901 of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.
- 1902 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty
1903 set of RegistryEntry instances. For each registry entry E in Y, and for each
1904 AssociationType A of the RecursiveAssociationOption, do the following:
- 1905 a) Let Z be the set of target items E' linked to E under association instances having
1906 E as the source object, E' as the target object, and A as the AssociationType.
- 1907 b) Add the elements of Z to NextLevel.
- 1908 10. Let X be the set of new registry entries that are in NextLevel but are not yet
1909 represented in the Result set.
- 1910 Case:
- 1911 a) If X is empty, then return the content of Result as the set of RepositoryItem
1912 elements in the ReturnRepositoryItemResult element.
- 1913 b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set.
1914 When finished, add the elements of X to Y and set Depth=Depth+1. If Depth is
1915 now equal to Limit, then return the content of Result as the set of RepositoryItem
1916 elements in the ReturnRepositoryItemResult element; otherwise, repeat
1917 Semantic Rules 9 and 10 with the new set Y of registry entries.
- 1918 11. If any exception, warning, or other status condition results during the execution of
1919 the above, then return appropriate RegistryError elements in the RegistryResult
1920 associated with the ReturnRepositoryItemResult element created in Semantic Rule 5
1921 or Semantic Rule 10.

1922 Examples

1923 A registry client has found a registry entry for a core-component item. The item's URN
1924 identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many
1925 other registered items. The client desires the collection of all items needed for a
1926 complete implementation of "goodthing". The following query returns an XML document
1927 that is a collection of all needed items.

```
1928 <ReturnRepositoryItem>  
1929 <RegistryEntryQuery>
```

```

1931     <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1932         id EQUAL "urn:ebxml:cc:goodthing"
1933     </RegistryEntryFilter>
1934 </RegistryEntryQuery>
1935 <RecursiveAssociationOption>
1936     <AssociationType role="Uses" />
1937     <AssociationType role="ValidatesTo" />
1938 </RecursiveAssociationOption>
1939 </ReturnRepositoryItem>
1940

```

1941 A registry client has found a reference to a core-component routine
 1942 ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client knows
 1943 that all routines have a required association to its defining UML specification. The
 1944 following query returns both the routine and its UML specification as a collection of two
 1945 items in a single XML document.

```

1946
1947 <ReturnRepositoryItem>
1948     <RegistryEntryQuery>
1949         <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1950             id EQUAL "urn:ebxml:cc:rtn:nice87"
1951         </RegistryEntryFilter>
1952     </RegistryEntryQuery>
1953     <RecursiveAssociationOption depthLimit="1" >
1954         <AssociationType role="ValidatesTo" />
1955     </RecursiveAssociationOption>
1956 </ReturnRepositoryItem>
1957

```

1958 A user has been told that the 1997 version of the North American Industry Classification
 1959 System (NAICS) is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The
 1960 following query would retrieve the complete classification scheme, with all 1810 nodes,
 1961 as an XML document that validates to a classification scheme DTD.

```

1962
1963 <ReturnRepositoryItem>
1964     <RegistryEntryQuery>
1965         <RegistryEntryFilter>                                -- code by Clause, Section 8.2.10
1966             id EQUAL "urn:nist:cs:naics-1997"
1967         </RegistryEntryFilter>
1968     </RegistryEntryQuery>
1969 </ReturnRepositoryItem>

```

1970
 1971 Note: The ReturnRepositoryItemResult would include a single RepositoryItem that
 1972 consists of a ClassificationScheme document whose content is determined by the URL
 1973 <ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt>.
 1974

1974 8.2.9 Registry Filters

1975 Purpose

1976 To identify a subset of the set of all persistent instances of a given registry class.

1977 Definition

1978
1979 <!ELEMENT ObjectFilter (Clause)>
1980
1981 <!ELEMENT RegistryEntryFilter (Clause)>
1982
1983 <!ELEMENT IntrinsicObjectFilter (Clause)>
1984
1985 <!ELEMENT ExtrinsicObjectFilter (Clause)>
1986
1987 <!ELEMENT PackageFilter (Clause)>
1988
1989 <!ELEMENT OrganizationFilter (Clause)>
1990
1991 <!ELEMENT ContactFilter (Clause)>
1992
1993 <!ELEMENT ClassificationNodeFilter (Clause)>
1994
1995 <!ELEMENT AssociationFilter (Clause)>
1996
1997 <!ELEMENT ClassificationFilter (Clause)>
1998
1999 <!ELEMENT ExternalLinkFilter (Clause)>
2000
2001 <!ELEMENT ExternalIdentifierFilter (Clause)>
2002
2003 <!ELEMENT SlotFilter (Clause)>
2004
2005 <!ELEMENT AuditableEventFilter (Clause)>
2006
2007 <!ELEMENT UserFilter (Clause)>

2008

2009 Semantic Rules

- 2010 1. The Clause element is defined in Section 8.2.10, Clause.
- 2011 2. For every ObjectFilter XML element, the leftArgument attribute of any containing
2012 SimpleClause shall identify a public attribute of the RegistryObject UML class
2013 defined in [ebRIM]. If not, raise exception: *object attribute error*. The ObjectFilter
2014 returns a set of identifiers for RegistryObject instances whose attribute values
2015 evaluate to *True* for the Clause predicate.
- 2016 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any
2017 containing SimpleClause shall identify a public attribute of the RegistryEntry UML
2018 class defined in [ebRIM].

- 2019 If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a
2020 set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
2021 for the Clause predicate.
- 2022 4. For every IntrinsicObjectFilter XML element, the leftArgument attribute of any
2023 containing SimpleClause shall identify a public attribute of the IntrinsicObject UML
2024 class defined in [ebRIM]. If not, raise exception: *intrinsic object attribute error*. The
2025 IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject instances whose
2026 attribute values evaluate to *True* for the Clause predicate.
- 2027 5. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any
2028 containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML
2029 class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The
2030 ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose
2031 attribute values evaluate to *True* for the Clause predicate.
- 2032 6. For every PackageFilter XML element, the leftArgument attribute of any containing
2033 SimpleClause shall identify a public attribute of the Package UML class defined in
2034 [ebRIM]. If not, raise exception: *package attribute error*. The PackageFilter returns a
2035 set of identifiers for Package instances whose attribute values evaluate to *True* for
2036 the Clause predicate.
- 2037 7. For every OrganizationFilter XML element, the leftArgument attribute of any
2038 containing SimpleClause shall identify a public attribute of the Organization or
2039 PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization*
2040 *attribute error*. The OrganizationFilter returns a set of identifiers for Organization
2041 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2042 8. For every ContactFilter XML element, the leftArgument attribute of any containing
2043 SimpleClause shall identify a public attribute of the Contact or PostalAddress UML
2044 class defined in [ebRIM]. If not, raise exception: *contact attribute error*. The
2045 ContactFilter returns a set of identifiers for Contact instances whose attribute values
2046 evaluate to *True* for the Clause predicate.
- 2047 9. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
2048 containing SimpleClause shall identify a public attribute of the ClassificationNode
2049 UML class defined in [ebRIM]. If not, raise exception: *classification node attribute*
2050 *error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode
2051 instances whose attribute values evaluate to *True* for the Clause predicate.
- 2052 10. For every AssociationFilter XML element, the leftArgument attribute of any
2053 containing SimpleClause shall identify a public attribute of the Association UML
2054 class defined in [ebRIM]. If not, raise exception: *association attribute error*. The
2055 AssociationFilter returns a set of identifiers for Association instances whose attribute
2056 values evaluate to *True* for the Clause predicate.

- 2057 11. For every ClassificationFilter XML element, the leftArgument attribute of any
 2058 containing SimpleClause shall identify a public attribute of the Classification UML
 2059 class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The
 2060 ClassificationFilter returns a set of identifiers for Classification instances whose
 2061 attribute values evaluate to *True* for the Clause predicate.
- 2062 12. For every ExternalLinkFilter XML element, the leftArgument attribute of any
 2063 containing SimpleClause shall identify a public attribute of the ExternalLink UML
 2064 class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The
 2065 ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose
 2066 attribute values evaluate to *True* for the Clause predicate.
- 2067 13. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any
 2068 containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML
 2069 class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The
 2070 ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances
 2071 whose attribute values evaluate to *True* for the Clause predicate.
- 2072 14. For every SlotFilter XML element, the leftArgument attribute of any containing
 2073 SimpleClause shall identify a public attribute of the Slot UML class defined in
 2074 [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of
 2075 identifiers for Slot instances whose attribute values evaluate to *True* for the Clause
 2076 predicate.
- 2077 15. For every AuditableEventFilter XML element, the leftArgument attribute of any
 2078 containing SimpleClause shall identify a public attribute of the AuditableEvent UML
 2079 class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The
 2080 AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose
 2081 attribute values evaluate to *True* for the Clause predicate.
- 2082 16. For every UserFilter XML element, the leftArgument attribute of any containing
 2083 SimpleClause shall identify a public attribute of the User UML class defined in
 2084 [ebRIM]. If not, raise exception: *auditable identity attribute error*. The UserFilter
 2085 returns a set of identifiers for User instances whose attribute values evaluate to *True*
 2086 for the Clause predicate.

2087

2088 **Example**

2089 The following is a complete example of RegistryEntryQuery combined with Clause
 2090 expansion of RegistryEntryFilter to return a set of RegistryEntry instances whose
 2091 objectType attribute is "CPP" and whose status attribute is "Approved".

```

2092 <RegistryEntryQuery>
2093   <RegistryEntryFilter>
2094     <Clause>
2095       <CompoundClause   connectivePredicate="And" >
2096         <Clause>
2097           <SimpleClause leftArgument="objectType" >
2098             <StringClause stringPredicate="equal" >CPP</StringClause>
2099

```

```
2100         </SimpleClause>
2101     </Clause>
2102 <Clause>
2103     <SimpleClause leftArgument="status" >
2104 <StringClause stringPredicate="equal" >Approved</StringClause>
2105     </SimpleClause>
2106 </Clause>
2107 </CompoundClause>
2108 </Clause>
2109 </RegistryEntryFilter>
2110 </RegistryEntryQuery>
2111
2112
```

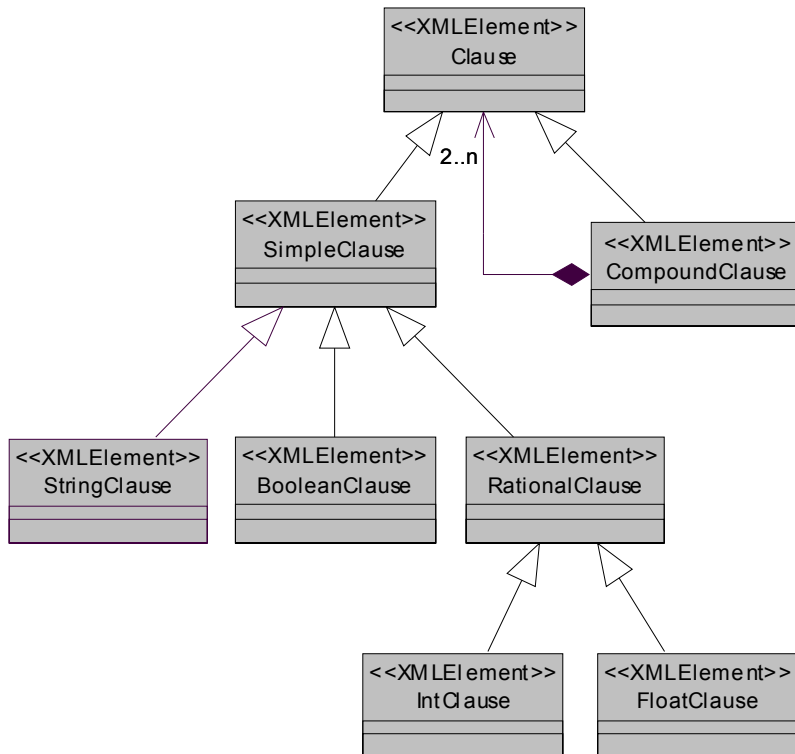
2112 8.2.10 XML Clause Constraint Representation

2113 Purpose

2114 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate*
 2115 *Clauses*. Predicate Clauses are utilized to formally define the constraint mechanism,
 2116 and are referred to simply as **Clauses** in this specification.

2117 Conceptual UML Diagram

2118 The following is a conceptual diagram outlining the Clause base structure. It is
 2119 expressed in UML for visual depiction.



2120

2121

Figure 20: The Clause base structure

2122 Semantic Rules

2123 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate -
 2124 RightArgument" format to form a *Clause*. There are two types of Clauses:
 2125 *SimpleClauses* and *CompoundClauses*.

2126 SimpleClauses

2127 A SimpleClause always defines the leftArgument as a text string, sometimes referred to
 2128 as the *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be
 2129 extended. SimpleClause is extended to support BooleanClause, StringClause, and
 2130 RationalClause (abstract).

2131 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
 2132 boolean. StringClause defines the predicate as an enumerated attribute of appropriate
 2133 string-compare operations and a right argument as the element's text data. Rational
 2134 number support is provided through a common RationalClause providing an
 2135 enumeration of appropriate rational number compare operations, which is further
 2136 extended to IntClause and FloatClause, each with appropriate signatures for the right
 2137 argument.

2138 CompoundClauses

2139 A CompoundClause contains two or more Clauses (Simple or Compound) and a
 2140 connective predicate. This provides for arbitrarily complex Clauses to be formed.

2141

2142 **Definition**

```

2143
2144 <!ELEMENT Clause ( SimpleClause | CompoundClause )>
2145
2146 <!ELEMENT SimpleClause
2147   ( BooleanClause | RationalClause | StringClause )>
2148 <!ATTLIST SimpleClause
2149   leftArgument CDATA #REQUIRED >
2150
2151 <!ELEMENT CompoundClause ( Clause, Clause+ )>
2152 <!ATTLIST CompoundClause
2153   connectivePredicate ( And | Or ) #REQUIRED>
2154
2155 <!ELEMENT BooleanClause EMPTY >
2156 <!ATTLIST BooleanClause
2157   booleanPredicate ( True | False ) #REQUIRED>
2158
2159 <!ELEMENT RationalClause ( IntClause | FloatClause )>
2160 <!ATTLIST RationalClause
2161   logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
2162
2163 <!ELEMENT IntClause ( #PCDATA )
2164 <!ATTLIST IntClause
2165   e-dtype NMTOKEN #FIXED 'int' >
2166
2167 <!ELEMENT FloatClause ( #PCDATA )>
2168 <!ATTLIST FloatClause
2169   e-dtype NMTOKEN #FIXED 'float' >
2170
2171 <!ELEMENT StringClause ( #PCDATA )>
2172 <!ATTLIST StringClause
2173   stringPredicate
2174     ( contains | -contains |
2175     startswith | -startswith |
  
```

2176 equal | -equal
2177 endswith | -endswith) #REQUIRED >

2178

2179 **Examples**

2180 **Simple BooleanClause: "Smoker" = True**

2181
2182 <?xml version="1.0" encoding="UTF-8"?>
2183 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2184 <Clause>
2185 <SimpleClause leftArgument="Smoker">
2186 <BooleanClause booleanPredicate="True"/>
2187 </SimpleClause>
2188 </Clause>
2189

2190 **Simple StringClause: "Smoker" contains "mo"**

2191
2192 <?xml version="1.0" encoding="UTF-8"?>
2193 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2194 <Clause>
2195 <SimpleClause leftArgument="Smoker">
2196 <StringClause stringcomparepredicate="contains">
2197 mo
2198 </StringClause>
2199 </SimpleClause>
2200 </Clause>

2201

2202 **Simple IntClause: "Age" >= 7**

2203
2204 <?xml version="1.0" encoding="UTF-8"?>
2205 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2206 <Clause>
2207 <SimpleClause leftArgument="Age">
2208 <RationalClause logicalPredicate="GE">
2209 <IntClause e-dtype="int">7</IntClause>
2210 </RationalClause>
2211 </SimpleClause>
2212 </Clause>
2213

2214 **Simple FloatClause: "Size" = 4.3**

2215
2216 <?xml version="1.0" encoding="UTF-8"?>
2217 <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2218 <Clause>
2219 <SimpleClause leftArgument="Size">
2220 <RationalClause logicalPredicate="E">
2221 <FloatClause e-dtype="float">4.3</FloatClause>
2222 </RationalClause>

2223 </SimpleClause>

2224 </Clause>

2225

2226 Compound with two Simples (("Smoker" = False)AND("Age" =< 45))

2227

2228 <?xml version="1.0" encoding="UTF-8"?>

2229 <!DOCTYPE Clause SYSTEM "Clause.dtd" >

2230 <Clause>

2231 <CompoundClause connectivePredicate="And">

2232 <Clause>

2233 <SimpleClause leftArgument="Smoker">

2234 <BooleanClause booleanPredicate="False"/>

2235 </SimpleClause>

2236 </Clause>

2237 <Clause>

2238 <SimpleClause leftArgument="Age">

2239 <RationalClause logicalPredicate="EL">

2240 <IntClause e-dtype="int">45</IntClause>

2241 </RationalClause>

2242 </SimpleClause>

2243 </Clause>

2244 </CompoundClause>

2245 </Clause>

2246

2247 Coumpound with one Simple and one Compound

2248 (("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

2249

2250 <?xml version="1.0" encoding="UTF-8"?>

2251 <!DOCTYPE Clause SYSTEM "Clause.dtd" >

2252 <Clause>

2253 <CompoundClause connectivePredicate="And">

2254 <Clause>

2255 <SimpleClause leftArgument="Smoker">

2256 <BooleanClause booleanPredicate="False"/>

2257 </SimpleClause>

2258 </Clause>

2259 <Clause>

2260 <CompoundClause connectivePredicate="Or">

2261 <Clause>

2262 <SimpleClause leftArgument="Age">

2263 <RationalClause logicalPredicate="EL">

2264 <IntClause e-dtype="int">45</IntClause>

2265 </RationalClause>

2266 </SimpleClause>

2267 </Clause>

2268 <Clause>

2269 <SimpleClause leftArgument="American">

2270 <BooleanClause booleanPredicate="True"/>

2271 </SimpleClause>

2272 </Clause>

2273 </CompoundClause>

2274 </Clause>
2275 </CompoundClause>
2276 </Clause>

2277 **8.3 SQL Query Support**

2278 The Registry may optionally support an SQL based query capability that is designed for
2279 Registry clients that demand more complex query capability. The optional SQLQuery
2280 element in the AdhocQueryRequest allows a client to submit complex SQL queries
2281 using a declarative query language.

2282 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper
2283 subset of the “SELECT” statement of Entry level SQL defined by ISO/IEC 9075:1992,
2284 Database Language SQL [SQL], extended to include `<sql invoked routines>`
2285 (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-
2286 defined routines defined in template form in Appendix C.3. The exact syntax of the
2287 Registry query language is defined by the BNF grammar in C.1.

2288 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement
2289 to use relational databases in a Registry implementation.

2290 **8.3.1 SQL Query Syntax Binding To [ebRIM]**

2291 SQL Queries are defined based upon the query syntax in in Appendix C.1 and a fixed
2292 relational schema defined in Appendix C.3. The relational schema is an algorithmic
2293 binding to [ebRIM] as described in the following sections.

2294 **8.3.1.1 Interface and Class Binding**

2295 A subset of the Interface and class names defined in [ebRIM] map to table names that
2296 may be queried by an SQL query. Appendix C.3 defines the names of the ebRIM
2297 interfaces and classes that may be queried by an SQL query.

2298 The algorithm used to define the binding of [ebRIM] classes to table definitions in
2299 Appendix C.3 is as follows:

- 2300 • Only those classes and interfaces that have concrete instances are mapped to
2301 relational tables. This results in intermediate interfaces in the inheritance
2302 hierarchy, such as RegistryObject and IntrinsicObject, to not map to SQL tables.
2303 An exception to this rule is RegistryEntry, which is defined next.
- 2304 • A special view called RegistryEntry is defined to allow SQL queries to be made
2305 against RegistryEntry instances. This is the only interface defined in [ebRIM] that
2306 does not have concrete instances but is queryable by SQL queries.
- 2307 • The names of relational tables are the same as the corresponding [ebRIM] class
2308 or interface name. However, the name binding is case insensitive.

- 2309 • Each [ebRIM] class or interface that maps to a table in Appendix C.3 includes
2310 column definitions in Appendix C.3 where the column definitions are based on a
2311 subset of attributes defined for that class or interface in [ebRIM]. The attributes
2312 that map to columns include the inherited attributes for the [ebRIM] class or
2313 interface. Comments in Appendix C.3 indicate which ancestor class or interface
2314 contributed which column definitions.

2315 An SQLQuery against a table not defined in Appendix C.3 may raise an error condition:
2316 InvalidQueryException.

2317 The following sections describe the algorithm for mapping attributes of [ebRIM] to
2318 SQLcolumn definitions.

2319 **8.3.1.2 Accessor Method To Attribute Binding**

2320 Most of the [ebRIM] interfaces methods are simple get methods that map directly to
2321 attributes. For example the getName method on RegistryObject maps to a name
2322 attribute of type String. Each get method in [ebRIM] defines the exact attribute name
2323 that it maps to in the interface definitions in [ebRIM].

2324 **8.3.1.3 Primitive Attributes Binding**

2325 Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the
2326 same way as column names in SQL. Again the exact attribute names are defined in the
2327 interface definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is
2328 case insensitive. It is therefore valid for a query to contain attribute names that do not
2329 exactly match the case defined in [ebRIM].

2330 **8.3.1.4 Reference Attribute Binding**

2331 A few of the [ebRIM] interface methods return references to instances of interfaces or
2332 classes defined by [ebRIM]. For example, the getAccessControlPolicy method of the
2333 RegistryObject class returns a reference to an instance of an AccessControlPolicy
2334 object.

2335 In such cases the reference maps to the `id` attribute for the referenced object. The
2336 name of the resulting column is the same as the attribute name in [ebRIM] as defined by
2337 8.3.1.3. The data type for the column is UUID as defined in Appendix C.3.

2338 When a reference attribute value holds a null reference, it maps to a null value in the
2339 SQL binding and may be tested with the `<null specification>` as defined by [SQL].

2340 Reference attribute binding is a special case of a primitive attribute mapping.

2341 **8.3.1.5 Complex Attribute Binding**

2342 A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead
2343 they are of a complex type as defined by an entity class in [ebRIM]. Examples include
2344 attributes of type TelephoneNumber, Contact, PersonName etc. in interface
2345 Organization and class Contact.

2346 The SQL query schema algorithmically maps such complex attributes as multiple
2347 primitive attributes within the parent table. The mapping simply flattens out the entity
2348 class attributes within the parent table. The attribute name for the flattened attributes
2349 are composed of a concatenation of attribute names in the reference chain. For example
2350 Organization has a contact attribute of type Contact. Contact has an address attribute of
2351 type PostalAddress. PostalAddress has a String attribute named city. This city attribute
2352 will be named contact_address_city.

2353 **8.3.1.6 Collection Attribute Binding**

2354 A few of the [ebRIM] interface methods return a collection of references to instances of
2355 interfaces or classes defined by [ebRIM]. For example, the getPackages method of the
2356 ManagedObject class returns a Collection of references to instances of Packages that
2357 the object is a member of.

2358 Such collection attributes in [ebRIM] classes have been mapped to stored procedures in
2359 Appendix C.3 such that these stored procedures return a collection of `id` attribute
2360 values. The returned value of these stored procedures can be treated as the result of a
2361 table sub-query in SQL.

2362 These stored procedures may be used as the right-hand-side of an SQL IN clause to
2363 test for membership of an object in such collections of references.

2364 **8.3.2 Semantic Constraints On Query Syntax**

2365 This section defines simplifying constraints on the query syntax that cannot be
2366 expressed in the BNF for the query syntax. These constraints must be applied in the
2367 semantic analysis of the query.

- 2368 1. Class names and attribute names must be processed in a case insensitive manner.
- 2369 2. The syntax used for stored procedure invocation must be consistent with the syntax
2370 of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 2371 3. For this version of the specification, the SQL select column list consists of exactly
2372 one column, and must always be `t.id`, where `t` is a table reference in the FROM
2373 clause.

2374 **8.3.3 SQL Query Results**

2375 The results of an SQL query is always an ObjectRefList as defined by the
2376 AdHocQueryResponse in 8.4. This means the result of an SQL query is always a
2377 collection of references to instances of a sub-class of the RegistryObject interface in
2378 [ebRIM]. This is reflected in a semantic constraint that requires that the SQL select
2379 column specified must always be an `id` column in a table in Appendix C.3 for this
2380 version of the specification.

2381 **8.3.4 Simple Metadata Based Queries**

2382 The simplest form of an SQL query is based upon metadata attributes specified for a
2383 single class within [ebRIM]. This section gives some examples of simple metadata
2384 based queries.

2385 For example, to get the collection of ExtrinsicObjects whose name contains the word
2386 'Acme' and that have a version greater than 1.3, the following query predicates must be
2387 supported:

```
2388  
2389 SELECT id FROM ExtrinsicObject WHERE name LIKE '%Acme%' AND  
2390 majorVersion >= 1 AND  
2391 (majorVersion >= 2 OR minorVersion > 3);
```

2392 Note that the query syntax allows for conjugation of simpler predicates into more
2393 complex queries as shown in the simple example above.

2394 **8.3.5 RegistryEntry Queries**

2395 Given the central role played by the RegistryEntry interface in ebRIM, the schema for
2396 the SQL query defines a special view called RegistryEntry that allows doing a
2397 polymorphic query against all RegistryEntry instances regardless of their actual
2398 concrete type or table name.

2399 The following example is the same as Section 8.3.4 except that it is applied against all
2400 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will
2401 include id for all qualifying RegistryEntry instances whose name contains the word
2402 'Acme' and that have a version greater than 1.3.

```
2403 SELECT id FROM RegistryEntry WHERE name LIKE '%Acme%' AND  
2404 objectType = 'ExtrinsicObject' AND  
2405 majorVersion >= 1 AND  
2406 (majorVersion >= 2 OR minorVersion > 3);
```

2407 **8.3.6 Classification Queries**

2408 This section describes the various classification related queries that must be supported.

2409 **8.3.6.1 Identifying ClassificationNodes**

2410 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they
2411 may also be identified as a path attribute that specifies an XPATH expression [XPT]
2412 from a root classification node to the specified classification node in the XML document
2413 that would represent the ClassificationNode tree including the said ClassificationNode.

2414 **8.3.6.2 Getting Root Classification Nodes**

2415 To get the collection of root ClassificationNodes the following query predicate must be
2416 supported:

```
2417 SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

2418 The above query returns all ClassificationNodes that have their parent attribute set to
 2419 null. Note that the above query may also specify a predicate on the name if a specific
 2420 root ClassificationNode is desired.

2421 **8.3.6.3 Getting Children of Specified ClassificationNode**

2422 To get the children of a ClassificationNode given the ID of that node the following style
 2423 of query must be supported:

```
2424 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

2425 The above query returns all ClassificationNodes that have the node specified by <id> as
 2426 their parent attribute.

2427 **8.3.6.4 Getting Objects Classified By a ClassificationNode**

2428 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the
 2429 following style of query must be supported:

```
2430 SELECT id FROM ExtrinsicObject
2431 WHERE
2432   id IN (SELECT classifiedObject FROM Classification
2433         WHERE
2434           classificationNode IN (SELECT id FROM ClassificationNode
2435                                WHERE path = '/Geography/Asia/Japan'))
2436 AND
2437   id IN (SELECT classifiedObject FROM Classification
2438         WHERE
2439           classificationNode IN (SELECT id FROM ClassificationNode
2440                                WHERE path = '/Industry/Automotive'))
2441
```

2442 The above query gets the collection of ExtrinsicObjects that are classified by the
 2443 Automotive Industry and the Japan Geography. Note that according to the semantics
 2444 defined for GetClassifiedObjectsRequest, the query will also contain any objects that
 2445 are classified by descendents of the specified ClassificationNodes.

2446 **8.3.6.5 Getting ClassificationNodes That Classify an Object**

2447 To get the collection of ClassificationNodes that classify a specified Object the following
 2448 style of query must be supported:

```
2449 SELECT id FROM ClassificationNode
2450 WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

2451 **8.3.7 Association Queries**

2452 This section describes the various Association related queries that must be supported.

2453 **8.3.7.1 Getting All Association With Specified Object As Its Source**

2454 To get the collection of Associations that have the specified Object as its source, the
 2455 following query must be supported:

```
2456 SELECT id FROM Association WHERE sourceObject = <id>
```

2457 8.3.7.2 Getting All Association With Specified Object As Its Target

2458 To get the collection of Associations that have the specified Object as its target, the
2459 following query must be supported:

```
2460 SELECT id FROM Association WHERE targetObject = <id>
```

2461 8.3.7.3 Getting Associated Objects Based On Association Attributes

2462 To get the collection of Associations that have specified Association attributes, the
2463 following queries must be supported:

2464 Select Associations that have the specified name.

```
2465 SELECT id FROM Association WHERE name = <name>
```

2466 Select Associations that have the specified source role name.

```
2467 SELECT id FROM Association WHERE sourceRole = <roleName>
```

2468 Select Associations that have the specified target role name.

```
2469 SELECT id FROM Association WHERE targetRole = <roleName>
```

2470 Select Associations that have the specified association type, where association type is a
2471 string containing the corresponding field name described in [ebRIM].

```
2472 SELECT id FROM Association WHERE  
2473 associationType = <associationType>
```

2474 8.3.7.4 Complex Association Queries

2475 The various forms of Association queries may be combined into complex predicates.

2476 The following query selects Associations from an object with a specified id, that have
2477 the sourceRole "buysFrom" and targetRole "sellsTo":

```
2478 SELECT id FROM Association WHERE  
2479 sourceObject = <id> AND  
2480 sourceRole = 'buysFrom' AND  
2481 targetRole = 'sellsTo'
```

2482 8.3.8 Package Queries

2483 To find all Packages that a specified ExtrinsicObject belongs to, the following query is
2484 specified:

```
2485 SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

2486 8.3.8.1 Complex Package Queries

2487 The following query gets all Packages that a specified object belongs to, that are not
2488 deprecated and where name contains "RosettaNet."

```
2489 SELECT id FROM Package WHERE  
2490 id IN (RegistryEntry_packages(<id>)) AND  
2491 name LIKE '%RosettaNet%' AND  
2492 status <> 'Deprecated'
```

2493 8.3.9 ExternalLink Queries

2494 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query
2495 is specified:

```
2496 SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

2497 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following
 2498 query is specified:

2499

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

2500 8.3.9.1 Complex ExternalLink Queries

2501 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to,
 2502 that contain the word 'legal' in their description and have a URL for their externalURI.

2503

```
SELECT id FROM ExternalLink WHERE  

  2504 id IN (RegistryEntry_externalLinks(<id>)) AND  

  2505 description LIKE '%legal%' AND  

  2506 externalURI LIKE '%http://%'
```

2507 8.3.10 Audit Trail Queries

2508 To get the complete collection of AuditableEvent objects for a specified ManagedObject,
 2509 the following query is specified:

2510

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

2511 8.4 Ad Hoc Query Request/Response

2512 A client submits an ad hoc query to the ObjectQueryManager by sending an
 2513 AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a
 2514 query in one of the supported Registry query mechanisms.

2515 The ObjectQueryManager sends an AdhocQueryResponse either synchronously or
 2516 asynchronously back to the client. The AdhocQueryResponse returns a collection of
 2517 objects whose element type is in the set of element types represented by the leaf nodes
 2518 of the RegistryEntry hierarchy in [ebRIM].

2519



2520

2521 **Figure 21: Submit Ad Hoc Query Sequence Diagram**

2522 For details on the schema for the business documents shown in this process refer to
2523 Appendix A.

2524 8.5 Content Retrieval

2525 A client retrieves content via the Registry by sending the GetContentRequest to the
2526 ObjectQueryManager. The GetContentRequest specifies a list of Object references for
2527 Objects that need to be retrieved. The ObjectQueryManager returns the specified
2528 content by sending a GetContentResponse message to the ObjectQueryManagerClient
2529 interface of the client. If there are no errors encountered, the GetContentResponse
2530 message includes the specified content as additional payloads within the message. In
2531 addition to the GetContentResponse payload, there is one additional payload for each
2532 content that was requested. If there are errors encountered, the RegistryResponse
2533 payload includes an error and there are no additional content specific payloads.

2534 8.5.1 Identification Of Content Payloads

2535 Since the GetContentResponse message may include several repository items as
2536 additional payloads, it is necessary to have a way to identify each payload in the
2537 message. To facilitate this identification, the Registry must do the following:

- 2538 • Use the ID for each RegistryEntry instance that describes the repository item as
2539 the DocumentLabel element in the DocumentReference for that object in the
2540 Manifest element of the ebXMLHeader.

2541 8.5.2 GetContentResponse Message Structure

2542 The following message fragment illustrates the structure of the GetContentResponse
2543 Message that is returning a Collection of CPPs as a result of a GetContentRequest that
2544 specified the IDs for the requested objects. Note that the ID for each object retrieved in
2545 the message as additional payloads is used as its DocumentLabel in the Manifest of the
2546 ebXMLHeader.

```
2547 ...
2548 ...
2549 --PartBoundary
2550 ...
2551 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2552 ...
2553   <eb:Service eb:type="ebXMLRegistry">ObjectManager</eb:Service>
2554   <eb:Action>submitObjects</eb:Action>
2555 </eb:MessageHeader>
2556 ...
2557 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2558   <eb:Reference xlink:href="cid:registryentries@example.com" ...>
2559     <eb:Description xml:lang="en-us">XML instances that are parameters for the particular
2560     Registry Interface / Method. These are RIM structures that don't include repository items, just a
2561     reference - contentURI to them.</eb:Description>
2562   </eb:Reference>
2563   <eb:Reference xlink:href="cid:cpp1@example.com" ...>
```

```

2567   <eb:Description xml:lang="en-us">XML instance of CPP 1. This is a repository
2568 item.</eb:Description>
2569   </eb:Reference>
2570   <eb:Reference xlink:href="cid:cpp2@example.com" ...>
2571     <eb:Description xml:lang="en-us">XML instance of CPP 2. This is a repository
2572 item.</eb:Description>
2573     </eb:Reference>
2574   </eb:Manifest>
2575
2576   --PartBoundary
2577   Content-ID: registryentries@example.com
2578   Content-Type: text/xml
2579   ...
2580   <?xml version="1.0" encoding="UTF-8"?>
2581   <RootElement>
2582     <SubmitObjectsRequest>
2583       <RegistryEntryList>
2584         <ExtrinsicObject ... contentURI="cid:cpp1@example.com" .../>
2585         <ExtrinsicObject ... contentURI="cid:cpp2@example.com" .../>
2586       </RegistryEntryList>
2587     </SubmitObjectsRequest>
2588   </RootElement>
2589   --PartBoundary
2590   Content-ID: cpp1@example.com
2591   Content-Type: text/xml
2592   ...
2593   <CPP>
2594   ...
2595   </CPP>
2596
2597   --PartBoundary
2598   Content-ID: cpp2@example.com
2599   Content-Type: text/xml
2600   ...
2601   <CPP>
2602   ...
2603   </CPP>
2604
2605   --PartBoundary--
2606

```

2607

2608 8.6 Query And Retrieval: Typical Sequence

2609 The following diagram illustrates the use of both browse/drilldown and ad hoc queries
 2610 followed by a retrieval of content that was selected by the queries.



2611

2612

Figure 23: Typical Query and Retrieval Sequence

2613 9 Registry Security

2614 This chapter describes the security features of the ebXML Registry. It is assumed that
 2615 the reader is familiar with the security related classes in the Registry information model
 2616 as described in [ebRIM].

2617 In the current version of this specification, a minimalist approach has been specified for
 2618 Registry security. The philosophy is that “Any *known* entity can publish content and
 2619 *anyone* can view published content.” The Registry information model has been
 2620 designed to allow more sophisticated security policies in future versions of this
 2621 specification.

2622 **9.1 Integrity of Registry Content**

2623 It is assumed that most business registries do not have the resources to validate the
2624 veracity of the content submitted to them. The minimal integrity that the Registry must
2625 provide is to ensure that content submitted by a Submitting Organization (SO) is
2626 maintained in the Registry without any tampering either *en-route* or *within* the Registry.
2627 Furthermore, the Registry must make it possible to identify the SO for any Registry
2628 content unambiguously.

2629 **9.1.1 Message Payload Signature**

2630 Integrity of Registry content requires that all submitted content must be signed by the
2631 Registry client as defined by [SEC]. The signature on the submitted content ensures
2632 that:

- 2633 • The content has not been tampered with en-route or within the Registry.
- 2634 • The content's veracity can be ascertained by its association with a specific
2635 submitting organization

2636 **9.2 Authentication**

2637 The Registry must be able to authenticate the identity of the Principal associated with
2638 client requests. *Authentication* is required to identify the ownership of content as well as
2639 to identify what "privileges" a Principal can be assigned with respect to the specific
2640 objects in the Registry.

2641 The Registry must perform Authentication on a per request basis. From a security point
2642 of view, all messages are independent and there is no concept of a session
2643 encompassing multiple messages or conversations. Session support may be added as
2644 an optimization feature in future versions of this specification.

2645 The Registry must implement a credential-based authentication mechanism based on
2646 digital certificates and signatures. The Registry uses the certificate DN from the
2647 signature to authenticate the user.

2648 **9.2.1 Message Header Signature**

2649 Message headers may be signed by the sending ebXML Messaging Service as defined
2650 by [SEC]. Since this specification is not yet finalized, this version does not require that
2651 the message header be signed. In the absence of a message header signature, the
2652 payload signature is used to authenticate the identity of the requesting client.

2653 9.3 Confidentiality

2654 9.3.1 On-the-wire Message Confidentiality

2655 It is suggested but not required that message payloads exchanged between clients and
2656 the Registry be encrypted during transmission. Payload encryption must abide by any
2657 restrictions set forth in [SEC].

2658 9.3.2 Confidentiality of Registry Content

2659 In the current version of this specification, there are no provisions for confidentiality of
2660 Registry content. All content submitted to the Registry may be discovered and read by
2661 *any* client. Therefore, the Registry must be able to decrypt any submitted content after it
2662 has been received and prior to storing it in its repository. This implies that the Registry
2663 and the client have an a priori agreement regarding encryption algorithm, key exchange
2664 agreements, etc. This service is not addressed in this specification.

2665 9.4 Authorization

2666 The Registry must provide an authorization mechanism based on the information model
2667 defined in [ebRIM]. In this version of the specification the authorization mechanism is
2668 based on a default Access Control Policy defined for a pre-defined set of roles for
2669 Registry users. Future versions of this specification will allow for custom Access Control
2670 Policies to be defined by the Submitting Organization.

2671 9.4.1 Pre-defined Roles For Registry Users

2672 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.

2673 9.4.2 Default Access Control Policies

2674 The Registry must create a default AccessControlPolicy object that grants the default
2675 permissions to Registry users based upon their assigned role.

2676 The following table defines the Permissions granted by the Registry to the various pre-
2677 defined roles for Registry users based upon the default AccessControlPolicy.

2678

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).

2679

2680 The following list summarizes the default role-based AccessControlPolicy:

- 2681 • The Registry must implement the default AccessControlPolicy and associate it
- 2682 with all Objects in the Registry
- 2683 • Anyone can publish content, but needs to be authenticated
- 2684 • Anyone can access the content without requiring authentication
- 2685 • The ContentOwner has access to all methods for Registry Objects owned by
- 2686 them
- 2687 • The RegistryAdministrator has access to all methods on all Registry Objects
- 2688 • Unauthenticated clients can access all read-only (getXXX) methods
- 2689 • At the time of content submission, the Registry must assign the default
- 2690 ContentOwner role to the Submitting Organization (SO) as authenticated by the
- 2691 credentials in the submission message. In the current version of this
- 2692 specification, it will be the DN as identified by the certificate
- 2693 • Clients that browse the Registry need not use certificates. The Registry must
- 2694 assign the default RegistryGuest role to such clients.

2695 **Appendix A ebXML Registry DTD Definition**

2696 The following is the definition for the various ebXML Message payloads described in
2697 this document.

2698

2699 `<?xml version="1.0" encoding="UTF-8"?>`2700 `<!-- Begin information model mapping. -->`

2701

2702 `<!--`2703 `ObjectAttributes are attributes from the RegistryObject interface in ebRIM.`

2704

2705 `id may be empty. If specified it may be in urn:uuid format or be in some`2706 `arbitrary format. If id is empty registry must generate globally unique id.`

2707
2708 If id is provided and in proper UUID syntax (starts with urn:uuid:)
2709 registry will honour it.
2710
2711 If id is provided and is not in proper UUID syntax then it is used for
2712 linkage within document and is ignored by the registry. In this case the
2713 registry generates a UUID for id attribute.
2714
2715 id must not be null when object is being retrieved from the registry.
2716 -->
2717 <!ENTITY % ObjectAttributes "
2718 id ID #IMPLIED
2719 name CDATA #IMPLIED
2720 description CDATA #IMPLIED
2721 ">
2722
2723 <!--
2724 Use as a proxy for an Object that is in the registry already.
2725 Specifies the id attribute of the object in the registry as its id attribute.
2726 id attribute in ObjectAttributes is exactly the same syntax and semantics as
2727 id attribute in RegistryObject.
2728 -->
2729 <!ELEMENT ObjectRef EMPTY>
2730 <!ATTLIST ObjectRef
2731 id ID #IMPLIED
2732 >
2733
2734 <!ELEMENT ObjectRefList (ObjectRef)*>
2735
2736 <!--
2737 RegistryEntryAttributes are attributes from the RegistryEntry interface
2738 in ebRIM.
2739 It inherits ObjectAttributes
2740 -->
2741 <!ENTITY % RegistryEntryAttributes "%ObjectAttributes;
2742 majorVersion CDATA '1'
2743 minorVersion CDATA '0'
2744 status CDATA #IMPLIED
2745 userVersion CDATA #IMPLIED
2746 stability CDATA 'Dynamic'
2747 expirationDate CDATA #IMPLIED">
2748
2749 <!ELEMENT RegistryEntry (SlotList?)>
2750 <!ATTLIST RegistryEntry
2751 %RegistryEntryAttributes; >
2752 <!ELEMENT Value (#PCDATA)>
2753 <!ELEMENT ValueList (Value*)>
2754 <!ELEMENT Slot (ValueList?)>
2755 <!ATTLIST Slot
2756 name CDATA #REQUIRED
2757 slotType CDATA #IMPLIED
2758 >
2759 <!ELEMENT SlotList (Slot*)>
2760
2761 <!--
2762 ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.

```
2763 It inherits RegistryEntryAttributes
2764 -->
2765
2766
2767 <!ELEMENT ExtrinsicObject EMPTY >
2768 <!ATTLIST ExtrinsicObject
2769     %RegistryEntryAttributes;
2770     contentURI CDATA #REQUIRED
2771     mimeType CDATA #IMPLIED
2772     objectType CDATA #REQUIRED
2773     opaque (true | false) "false"
2774 >
2775
2776
2777 <!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
2778
2779 <!-- Leaf classes that reflect the concrete classes in ebRIM -->
2780 <!ELEMENT RegistryEntryList
2781 (Association | Classification | ClassificationNode | Package |
2782 ExternalLink | ExternalIdentifier | Organization |
2783 ExtrinsicObject | ObjectRef)*>
2784
2785 <!--
2786 An ExternalLink specifies a link from a RegistryEntry and an external URI
2787 -->
2788 <!ELEMENT ExternalLink EMPTY>
2789 <!ATTLIST ExternalLink
2790     %IntrinsicObjectAttributes;
2791     externalURI CDATA #IMPLIED
2792 >
2793
2794 <!--
2795 An ExternalIdentifier provides an identifier for a RegistryEntry
2796
2797 The value is the value of the identifier (e.g. the social security number)
2798 -->
2799 <!ELEMENT ExternalIdentifier EMPTY>
2800 <!ATTLIST ExternalIdentifier
2801     %IntrinsicObjectAttributes;
2802     value CDATA #REQUIRED
2803 >
2804
2805 <!--
2806 An Association specifies references to two previously submitted
2807 registry entrys.
2808
2809 The sourceObject is id of the sourceObject in association
2810 The targetObject is id of the targetObject in association
2811 -->
2812 <!ELEMENT Association EMPTY>
2813 <!ATTLIST Association
2814     %IntrinsicObjectAttributes;
2815     sourceRole CDATA #IMPLIED
2816     targetRole CDATA #IMPLIED
2817     associationType CDATA #REQUIRED
2818     bidirection (true | false) "false"
```

```
2819         sourceObject IDREF #REQUIRED
2820         targetObject IDREF #REQUIRED
2821     >
2822
2823 <!--
2824 A Classification specifies references to two registry entries.
2825
2826 The classifiedObject is id of the Object being classified.
2827 The classificationNode is id of the ClassificationNode classifying the object
2828 -->
2829 <!ELEMENT Classification EMPTY>
2830 <!ATTLIST Classification
2831         %IntrinsicObjectAttributes;
2832         classifiedObject IDREF #REQUIRED
2833         classificationNode IDREF #REQUIRED
2834 >
2835
2836 <!--
2837 A Package is a named collection of objects.
2838 -->
2839 <!ELEMENT Package EMPTY>
2840 <!ATTLIST Package
2841         %IntrinsicObjectAttributes;
2842 >
2843
2844 <!-- Attributes inherited by various types of telephone number elements -->
2845 <!ENTITY % TelephoneNumberAttributes " areaCode    CDATA    #REQUIRED
2846         contryCode CDATA    #REQUIRED
2847         extension  CDATA    #IMPLIED
2848         number     CDATA    #REQUIRED
2849         url        CDATA    #IMPLIED">
2850 <!ELEMENT TelephoneNumber EMPTY>
2851 <!ATTLIST TelephoneNumber
2852         %TelephoneNumberAttributes;
2853 >
2854 <!ELEMENT FaxNumber EMPTY>
2855 <!ATTLIST FaxNumber
2856         %TelephoneNumberAttributes;
2857 >
2858
2859 <!ELEMENT PagerNumber EMPTY>
2860 <!ATTLIST PagerNumber
2861         %TelephoneNumberAttributes;
2862 >
2863
2864 <!ELEMENT MobileTelephoneNumber EMPTY>
2865 <!ATTLIST MobileTelephoneNumber
2866         %TelephoneNumberAttributes;
2867 >
2868 <!-- PostalAddress -->
2869 <!ELEMENT PostalAddress EMPTY>
2870 <!ATTLIST PostalAddress
2871         city CDATA #REQUIRED
2872         country CDATA #REQUIRED
2873         postalCode CDATA #REQUIRED
2874         state CDATA #IMPLIED
```

```
2875         street CDATA #REQUIRED
2876     >
2877     <!-- PersonName -->
2878     <!ELEMENT PersonName EMPTY>
2879     <!ATTLIST PersonName
2880         firstName CDATA #REQUIRED
2881         middleName CDATA #IMPLIED
2882         lastName CDATA #REQUIRED
2883     >
2884
2885     <!-- Organization -->
2886     <!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
2887     <!ATTLIST Organization
2888         %IntrinsicObjectAttributes;
2889         parent IDREF #IMPLIED
2890         primaryContact IDREF #REQUIRED
2891     >
2892
2893     <!ELEMENT User (PersonName, PostalAddress, TelephoneNumber,
2894         MobileTelephoneNumber?,
2895         FaxNumber?, PagerNumber?)>
2896     <!ATTLIST User
2897         %ObjectAttributes;
2898         organization IDREF #IMPLIED
2899         email CDATA #IMPLIED
2900         url CDATA #IMPLIED
2901     >
2902
2903     <!ELEMENT AuditableEvent EMPTY>
2904     <!ATTLIST AuditableEvent
2905         %ObjectAttributes;
2906         eventType CDATA #REQUIRED
2907         registryEntry IDREF #REQUIRED
2908         timestamp CDATA #REQUIRED
2909         user IDREF #REQUIRED
2910     >
2911
2912     <!--
2913     ClassificationNode is used to submit a Classification tree to the Registry.
2914
2915     parent is the id to the parent node. code is an optional code value for a
2916         ClassificationNode
2917     often defined by an external taxonomy (e.g. NAICS)
2918     -->
2919     <!ELEMENT ClassificationNode EMPTY>
2920     <!ATTLIST ClassificationNode
2921         %IntrinsicObjectAttributes;
2922         parent IDREF #IMPLIED
2923         code CDATA #IMPLIED
2924     >
2925
2926     <!--
2927     End information model mapping.
2928
2929     Begin Registry Services Interface
2930
```

```
2931 <!ELEMENT RequestAcceptedResponse EMPTY>
2932 <!ATTLIST RequestAcceptedResponse
2933     xml:lang NMTOKEN #REQUIRED
2934 >
2935 <!--
2936
2937 The SubmitObjectsRequest allows one to submit a list of RegistryEntry
2938 elements. Each RegistryEntry element provides metadata for a single submitted
2939 object. Note that the repository item being submitted is in a separate
2940 document that is not in this DTD. The ebXML Messaging Services Specification
2941 defines packaging, for submission, of the metadata of a repository item with
2942 the repository item itself. The value of the contentURI attribute of the
2943 ExtrinsicObject element must be the same as the xlink:href attribute within
2944 the Reference element within the Manifest element of the MessageHeader.
2945 -->
2946 <!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
2947 <!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
2948 <!-- Only need name in Slot within SlotList -->
2949 <!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
2950 <!--
2951 The ObjectRefList is the list of
2952 refs to the registry entrys being approved.
2953 -->
2954 <!ELEMENT ApproveObjectsRequest (ObjectRefList)>
2955 <!--
2956 The ObjectRefList is the list of
2957 refs to the registry entrys being deprecated.
2958 -->
2959 <!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
2960 <!--
2961 The ObjectRefList is the list of
2962 refs to the registry entrys being removed
2963 -->
2964 <!ELEMENT RemoveObjectsRequest (ObjectRefList)>
2965 <!ATTLIST RemoveObjectsRequest
2966     deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
2967 >
2968 <!ELEMENT GetRootClassificationNodesRequest EMPTY>
2969 <!--
2970 The namePattern follows SQL-92 syntax for the pattern specified in
2971 LIKE clause. It allows for selecting only those root nodes that match
2972 the namePattern. The default value of '*' matches all root nodes.
2973 -->
2974 <!ATTLIST GetRootClassificationNodesRequest
2975     namePattern CDATA "*"
2976 >
2977 <!--
2978 The response includes one or more ClassificationNodes
2979 -->
2980 <!ELEMENT GetRootClassificationNodesResponse ( ClassificationNode+ )>
2981 <!--
2982 Get the classification tree under the ClassificationNode specified parentRef.
2983
2984 If depth is 1 just fetch immediate child
2985 nodes, otherwise fetch the descendant tree upto the specified depth level.
2986 If depth is 0 that implies fetch entire sub-tree
```

```
2987 -->
2988 <!ELEMENT GetClassificationTreeRequest EMPTY>
2989 <!ATTLIST GetClassificationTreeRequest
2990     parent CDATA #REQUIRED
2991     depth CDATA "1"
2992 >
2993 <!--
2994 The response includes one or more ClassificationNodes which includes only
2995 immediate ClassificationNode children nodes if depth attribute in
2996 GetClassificationTreeRequest was 1, otherwise the decendent nodes
2997 upto specified depth level are returned.
2998 -->
2999 <!ELEMENT GetClassificationTreeResponse ( ClassificationNode+ )>
3000 <!--
3001 Get refs to all registry entrys that are classified by all the
3002 ClassificationNodes specified by ObjectRefList.
3003 Note this is an implicit logical AND operation
3004 -->
3005 <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
3006 <!--
3007 objectType attribute can specify the type of objects that the registry
3008 client is interested in, that is classified by this ClassificationNode.
3009 It is a String that matches a choice in the type attribute of
3010     ExtrinsicObject.
3011 The default value of '*' implies that client is interested in all types
3012 of registry entrys that are classified by the specified ClassificationNode.
3013 -->
3014 <!--
3015 The response includes a RegistryEntryList which has zero or more
3016 RegistryEntrys that are classified by the ClassificationNodes
3017 specified in the ObjectRefList in GetClassifiedObjectsRequest.
3018 -->
3019 <!ELEMENT GetClassifiedObjectsResponse ( RegistryEntryList )>
3020 <!--
3021 An Ad hoc query request specifies a query string as defined by [RS] in the
3022     queryString attribute
3023 -->
3024 <!ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |
3025     ReturnRepositoryItem | SQLQuery)>
3026 <!ELEMENT SQLQuery (#PCDATA)>
3027 <!--
3028 The response includes a RegistryEntryList which has zero or more
3029 RegistryEntrys that match the query specified in AdhocQueryRequest.
3030 -->
3031 <!ELEMENT AdhocQueryResponse
3032     ( RegistryEntryList |
3033     FilterQueryResult |
3034     ReturnRegistryEntryResult |
3035     ReturnRepositoryItemResult )>
3036 <!--
3037 Gets the actual content (not metadata) specified by the ObjectRefList
3038 -->
3039 <!ELEMENT GetContentRequest (ObjectRefList)>
3040 <!--
3041 The GetObjectsResponse will have no sub-elements if there were no errors.
3042 The actual contents will be in the other payloads of the message.
```

```
3043 -->
3044 <!ELEMENT GetContentResponse EMPTY >
3045 <!--
3046 Describes the capability profile for the registry and what optional features
3047 are supported
3048 -->
3049 <!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
3050 <!ATTLIST RegistryProfile
3051     version CDATA #REQUIRED
3052 >
3053
3054 <!ELEMENT OptionalFeaturesSupported EMPTY>
3055 <!ATTLIST OptionalFeaturesSupported
3056     sqlQuery (true | false) "false"
3057     xQuery (true | false) "false"
3058 >
3059 <!-- Begin FilterQuery DTD -->
3060 <!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
3061     ClassificationNodeQuery |
3062     RegistryPackageQuery |
3063     OrganizationQuery)>
3064 <!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
3065     AuditableEventQueryResult |
3066     ClassificationNodeQueryResult |
3067     RegistryPackageQueryResult |
3068     OrganizationQueryResult)>
3069 <!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
3070 <!ELEMENT RegistryEntryView EMPTY>
3071 <!ATTLIST RegistryEntryView
3072     objectURN CDATA #REQUIRED
3073     contentURI CDATA #IMPLIED
3074     objectID CDATA #IMPLIED
3075 >
3076 <!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
3077 <!ELEMENT AuditableEventView EMPTY>
3078 <!ATTLIST AuditableEventView
3079     objectID CDATA #REQUIRED
3080     timestamp CDATA #REQUIRED
3081 >
3082 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
3083 <!ELEMENT ClassificationNodeView EMPTY>
3084 <!ATTLIST ClassificationNodeView
3085     objectURN CDATA #REQUIRED
3086     contentURI CDATA #IMPLIED
3087     objectID CDATA #IMPLIED
3088 >
3089 <!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
3090 <!ELEMENT RegistryPackageView EMPTY>
3091 <!ATTLIST RegistryPackageView
3092     objectURN CDATA #REQUIRED
3093     contentURI CDATA #IMPLIED
3094     objectID CDATA #IMPLIED
3095 >
3096 <!ELEMENT OrganizationQueryResult (OrganizationView*)>
3097 <!ELEMENT OrganizationView EMPTY>
3098 <!ATTLIST OrganizationView
```

```
3099         orgURN CDATA #REQUIRED
3100         objectID CDATA #IMPLIED
3101     >
3102
3103 <!ELEMENT RegistryEntryQuery
3104     ( RegistryEntryFilter?,
3105       SourceAssociationBranch*,
3106       TargetAssociationBranch*,
3107       HasClassificationBranch*,
3108       SubmittingOrganizationBranch?,
3109       ResponsibleOrganizationBranch?,
3110       ExternalIdentifierFilter*,
3111       ExternalLinkFilter*,
3112       SlotFilter*,
3113       HasAuditableEventBranch*           )>
3114
3115 <!ELEMENT SourceAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3116 <!ELEMENT TargetAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3117 <!ELEMENT HasClassificationBranch (ClassificationFilter?,
3118                                   ClassificationNodeFilter?)>
3119 <!ELEMENT SubmittingOrganizationBranch (OrganizationFilter?, ContactFilter?)>
3120 <!ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
3121                                         ContactFilter?)>
3122 <!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
3123                                   OrganizationFilter?)>
3124 <!ELEMENT AuditableEventQuery
3125     (AuditableEventFilter?, RegistryEntryQuery*, InvokedByBranch? )>
3126
3127 <!ELEMENT InvokedByBranch
3128     ( UserFilter?, OrganizationQuery? )>
3129
3130 <!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
3131                                   PermitsClassificationBranch*,
3132                                   HasParentNode?, HasSubnode*)>
3133 <!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
3134                                       RegistryEntryQuery?)>
3135 <!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
3136 <!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
3137 <!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
3138 <!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
3139 <!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
3140                             HasParentOrganization?,
3141                             InvokesEventBranch*,
3142                             ContactFilter*)>
3143 <!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
3144 <!ELEMENT HasParentOrganization (OrganizationFilter?,
3145                                 HasParentOrganization?)>
3146 <!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
3147                               RegistryEntryQuery?)>
3148 <!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
3149                               WithSourceAssociations?,
3150                               WithTargetAssociations?,
3151                               WithAuditableEvents?,
3152                               WithExternalLinks?)>
3153 <!ELEMENT WithClassifications (ClassificationFilter?)>
3154 <!ELEMENT WithSourceAssociations (AssociationFilter?)>
```

```
3155 <!ELEMENT WithTargetAssociations (AssociationFilter?)>
3156 <!ELEMENT WithAuditableEvents (AuditableEventFilter?)>
3157 <!ELEMENT WithExternalLinks (ExternalLinkFilter?)>
3158 <!ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*)>
3159 <!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
3160                                     SourceAssociations?,
3161                                     TargetAssociations?,
3162                                     AuditableEvent*, ExternalLink*)>
3163 <!ELEMENT SourceAssociations (Association*)>
3164 <!ELEMENT TargetAssociations (Association*)>
3165 <!ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
3166                                     RecursiveAssociationOption?,
3167                                     WithDescription?)>
3168 <!ELEMENT RecursiveAssociationOption (AssociationType+)>
3169 <!ATTLIST RecursiveAssociationOption
3170     depthLimit CDATA #IMPLIED
3171 >
3172 <!ELEMENT AssociationType EMPTY>
3173 <!ATTLIST AssociationType
3174     role CDATA #REQUIRED
3175 >
3176 <!ELEMENT WithDescription EMPTY>
3177 <!ELEMENT ReturnRepositoryItemResult (RepositoryItem*)>
3178 <!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject
3179                             | ExternalLink)>
3180 <!ATTLIST RepositoryItem
3181     identifier CDATA #REQUIRED
3182     name CDATA #REQUIRED
3183     contentURI CDATA #REQUIRED
3184     objectType CDATA #REQUIRED
3185     status CDATA #REQUIRED
3186     stability CDATA #REQUIRED
3187     description CDATA #IMPLIED
3188 >
3189 <!ELEMENT RegistryPackage EMPTY>
3190 <!ELEMENT WithdrawnObject EMPTY>
3191 <!ELEMENT ExternalLinkItem EMPTY>
3192 <!ELEMENT ObjectFilter (Clause)>
3193 <!ELEMENT RegistryEntryFilter (Clause)>
3194 <!ELEMENT IntrinsicObjectFilter (Clause)>
3195 <!ELEMENT ExtrinsicObjectFilter (Clause)>
3196 <!ELEMENT PackageFilter (Clause)>
3197 <!ELEMENT OrganizationFilter (Clause)>
3198 <!ELEMENT ContactFilter (Clause)>
3199 <!ELEMENT ClassificationNodeFilter (Clause)>
3200 <!ELEMENT AssociationFilter (Clause)>
3201 <!ELEMENT ClassificationFilter (Clause)>
3202 <!ELEMENT ExternalLinkFilter (Clause)>
3203 <!ELEMENT SlotFilter (Clause)>
3204 <!ELEMENT ExternalIdentifierFilter (Clause)>
3205 <!ELEMENT AuditableEventFilter (Clause)>
3206 <!ELEMENT UserFilter (Clause)>
3207
3208 <!--
3209 The following lines define the XML syntax for Clause.
3210 -->
```

```
3211 <!ELEMENT Clause (SimpleClause | CompoundClause)>
3212 <!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
3213 <!ATTLIST SimpleClause
3214     leftArgument CDATA #REQUIRED
3215 >
3216 <!ELEMENT CompoundClause (Clause, Clause+)>
3217 <!ATTLIST CompoundClause
3218     connectivePredicate (And | Or) #REQUIRED
3219 >
3220 <!ELEMENT BooleanClause EMPTY>
3221 <!ATTLIST BooleanClause
3222     booleanPredicate (true | false) #REQUIRED
3223 >
3224 <!ELEMENT RationalClause (IntClause | FloatClause)>
3225 <!ATTLIST RationalClause
3226     logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
3227 >
3228 <!ELEMENT IntClause (#PCDATA)>
3229 <!ATTLIST IntClause
3230     e-dtype NMTOKEN #FIXED "int"
3231 >
3232 <!ELEMENT FloatClause (#PCDATA)>
3233 <!ATTLIST FloatClause
3234     e-dtype NMTOKEN #FIXED "float"
3235 >
3236 <!ELEMENT StringClause (#PCDATA)>
3237 <!ATTLIST StringClause
3238     stringPredicate
3239     (contains | -contains |
3240      startswith | -startswith |
3241      equal | -equal |
3242      endswith | -endswith) #REQUIRED
3243 >
3244 <!-- End FilterQuery DTD -->
3245
3246 <!-- Begin RegistryError definition -->
3247 <!-- The RegistryErrorList is derived from the ErrorList element from the
3248     ebXML Message Service Specification -->
3249 <!ELEMENT RegistryErrorList ( RegistryError+ )>
3250 <!ATTLIST RegistryErrorList
3251     highestSeverity ( Warning | Error ) 'Warning' >
3252
3253 <!ELEMENT RegistryError (#PCDATA) >
3254 <!ATTLIST RegistryError
3255     codeContext CDATA #REQUIRED
3256     errorCode CDATA #REQUIRED
3257     severity ( Warning | Error ) 'Warning'
3258     location CDATA #IMPLIED
3259     xml:lang NMTOKEN #IMPLIED>
3260
3261 <!ELEMENT RegistryResponse
3262     (( AdhocQueryResponse |
3263      GetContentResponse |
3264      GetClassificationTreeResponse |
3265      GetClassifiedObjectsResponse |
3266      GetRootClassificationNodesResponse )?),
```

```
3267     RegistryErrorList? )>
3268 <!ATTLIST RegistryResponse
3269     status (success | failure) #REQUIRED >
3270
3271 <!-- The contrived root node -->
3272
3273 <!ELEMENT RootElement
3274     ( SubmitObjectsRequest |
3275       ApproveObjectsRequest |
3276       DeprecateObjectsRequest |
3277       RemoveObjectsRequest |
3278       GetRootClassificationNodesRequest |
3279       GetClassificationTreeRequest |
3280       GetClassifiedObjectsRequest |
3281       AdhocQueryRequest |
3282       GetContentRequest |
3283       AddSlotsRequest |
3284       RemoveSlotsRequest |
3285       RegistryResponse |
3286       RegistryProfile) >
3287
3288 <!ELEMENT Href (#PCDATA) >
3289
3290 <!ELEMENT XMLDocumentErrorLocn (DocumentId , Xpath) >
3291
3292 <!ELEMENT DocumentId (#PCDATA) >
3293
3294 <!ELEMENT Xpath (#PCDATA) >
```

3295 **Appendix B**

3296 **Interpretation of UML Diagrams**

3297 This section describes in *abstract terms* the conventions used to define ebXML
3298 business process description in UML.

3299 **B.1 UML Class Diagram**

3300 A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP])
3301 required to implement an ebXML Registry Services and clients. See Figure 2 on page
3302 14 for an example. The UML class diagram contains:

- 3303
- 3304 1. A collection of UML interfaces where each interface represents a Service
3305 Interface for a Registry service.
 - 3306 2. Tabular description of methods on each interface where each method represents
3307 an Action (as defined by [ebCPP]) within the Service Interface representing the
3308 UML interface.

- 3309 3. Each method within a UML interface specifies one or more parameters, where
3310 the type of each method argument represents the ebXML message type that is
3311 exchanged as part of the Action corresponding to the method. Multiple
3312 arguments imply multiple payload documents within the body of the
3313 corresponding ebXML message.

3314 **B.2 UML Sequence Diagram**

3315 A UML sequence diagram is used to specify the business protocol representing the
3316 interactions between the UML interfaces for a Registry specific ebXML business
3317 process. A UML sequence diagram provides the necessary information to determine the
3318 sequencing of messages, request to response association as well as request to error
3319 response association as described by [ebCPP].

3320 Each sequence diagram shows the sequence for a specific conversation protocol as
3321 method calls from the requestor to the responder. Method invocation may be
3322 synchronous or asynchronous based on the UML notation used on the arrow-head for
3323 the link. A half arrow-head represents asynchronous communication. A full arrow-head
3324 represents synchronous communication.

3325 Each method invocation may be followed by a response method invocation from the
3326 responder to the requestor to indicate the ResponseName for the previous Request.
3327 Possible error response is indicated by a conditional response method invocation from
3328 the responder to the requestor. See on page 20 for an example.

3329 **Appendix C SQL Query**

3330 **C.1 SQL Query Syntax Specification**

3331 This section specifies the rules that define the SQL Query syntax as a subset of
3332 SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in
3333 [SQL/PSM]. The SQL query syntax conforms to the <query specification>, modulo
3334 the restrictions identified below:

- 3335 1. A <select list> may contain at most one <select sublist>.
- 3336 2. In a <select list> must be is a single column whose data type is UUID, from the
3337 table in the <from clause>.
- 3338 3. A <derived column> may not have an <as clause>.
- 3339 4. <table expression> does not contain the optional <group by clause> and <having
3340 clause> clauses.
- 3341 5. A <table reference> can only consist of <table name> and <correlation name>.
- 3342 6. A <table reference> does not have the optional AS between <table name> and
3343 <correlation name>.

- 3344 7. There can only be one <table reference> in the <from clause>.
- 3345 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in
- 3346 predicate> allows for the right hand side of the <in predicate> to be limited to a
- 3347 restricted <query specification> as defined above.
- 3348 9. A <search condition> within the <where clause> may not include a <query
- 3349 expression>.
- 3350 10. The SQL query syntax allows for the use of <sql invoked routines>
- 3351 invocation from [SQL/PSM] as the RHS of the <in predicate>.

3352 C.2 Non-Normative BNF for Query Syntax Grammar

3353 The following BNF exemplifies the grammar for the registry query syntax. It is provided

3354 here as an aid to implementors. Since this BNF is not based on [SQL] it is provided as

3355 non-normative syntax. For the normative syntax rules see Appendix C.1.

```

3356 /*****
3357 * The Registry Query (Subset of SQL-92) grammar starts here
3358 *****/
3359 RegistryQuery = SQLSelect [ ";" ]
3360
3361 SQLSelect = "SELECT" SQLSelectCols "FROM" SQLTableList [ SQLWhere ]
3362
3363 SQLSelectCols = ID
3364
3365 SQLTableList = SQLTableRef
3366
3367 SQLTableRef = ID
3368
3369 SQLWhere = "WHERE" SQLOrExpr
3370
3371 SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr ) *
3372
3373 SQLAndExpr = SQLNotExpr ( "AND" SQLNotExpr ) *
3374
3375 SQLNotExpr = [ "NOT" ] SQLCompareExpr
3376
3377 SQLCompareExpr =
3378   (SQLColRef "IS") SQLIsClause
3379   | SQLSumExpr [ SQLCompareExprRight ]
3380
3381 SQLCompareExprRight =
3382   SQLLikeClause
3383   | SQLInClause
3384   | SQLCompareOp SQLSumExpr
3385
3386 SQLCompareOp =
3387   "="
3388   | "<>"
3389   | ">"
3390   | ">="
3391   | "<"
3392   | "<="
3393
3394 SQLInClause = [ "NOT" ] "IN" "(" SQLValueList ")"
3395
3396

```

```

3400 SQLValueList = SQLValueElement ( "," SQLValueElement ) *
3401
3402 SQLValueElement = "NULL" | SQLSelect
3403
3404 SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
3405
3406 SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
3407
3408 SQLPattern = STRING_LITERAL
3409
3410 SQLLiteral =
3411     STRING_LITERAL
3412     | INTEGER_LITERAL
3413     | FLOATING_POINT_LITERAL
3414
3415 SQLColRef = SQLValue
3416
3417 SQLValue = SQLValueTerm
3418
3419 SQLValueTerm = ID ( "." ID ) *
3420
3421 SQLSumExpr = SQLProductExpr ( ( "+" | "-" ) SQLProductExpr ) *
3422
3423 SQLProductExpr = SQLUnaryExpr ( ( "*" | "/" ) SQLUnaryExpr ) *
3424
3425 SQLUnaryExpr = [ ( "+" | "-" ) ] SQLTerm
3426
3427 SQLTerm = "(" SQLOrExpr ")"
3428     | SQLColRef
3429     | SQLLiteral
3430
3431 INTEGER_LITERAL = ([ "0"-"9" ] ) +
3432
3433 FLOATING_POINT_LITERAL =
3434     ([ "0"-"9" ] ) + "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3435     | "." ([ "0"-"9" ] ) + ( EXPONENT ) ?
3436     | ([ "0"-"9" ] ) + EXPONENT
3437     | ([ "0"-"9" ] ) + ( EXPONENT ) ?
3438
3439 EXPONENT = [ "e", "E" ] ( [ "+" , "-" ] ) ? ([ "0"-"9" ] ) +
3440
3441 STRING_LITERAL: "'" (~[ "'" ] ) * ( '"' (~[ '"' ] ) * ) * "'"
3442
3443 ID = ( <LETTER> ) + ( " $" | "#" | <DIGIT> | <LETTER> ) *
3444 LETTER = [ "A"-"Z", "a"-"z" ]
3445 DIGIT = [ "0"-"9" ]

```

3446 C.3 Relational Schema For SQL Queries

```

3447
3448 --SQL Load file for creating the ebXML Registry tables
3449
3450
3451 --Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92
3452
3453
3454 CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
3455 CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
3456 CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
3457
3458 CREATE TYPE UUID UNDER ShortName FINAL;
3459 CREATE TYPE URI UNDER LongName FINAL;
3460
3461 CREATE TABLE ExtrinsicObject (
3462
3463 --RegistryObject Attributes
3464     id                                UUID PRIMARY KEY NOT NULL,
3465     name                               LongName,

```

```

3466     description                               FreeFormText,
3467     accessControlPolicy                       UUID NOT NULL,
3468
3469 --Versionable attributes
3470     majorVersion                             INT DEFAULT 0 NOT NULL,
3471     minorVersion                             INT DEFAULT 1 NOT NULL,
3472
3473 --RegistryEntry attributes
3474     status                                    INT DEFAULT 0 NOT NULL,
3475     userVersion                              ShortName,
3476     stability                                INT     DEFAULT 0 NOT NULL,
3477     expirationDate                          TIMESTAMP,
3478
3479 --ExtrinsicObject attributes
3480     contentURI                               URI,
3481     mimeType                                 ShortName,
3482     objectType                              INT DEFAULT 0 NOT NULL,
3483     opaque                                   BOOLEAN DEFAULT false NOT NULL
3484
3485 );
3486
3487 CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
3488 --Must return a collection of UUIDs for related RegistryEntry instances
3489 }
3490
3491 CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
3492 --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
3493 --Collection must be in ascending order by timestamp
3494 }
3495
3496 CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
3497 --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
3498 }
3499
3500 CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
3501 --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
3502 }
3503
3504 CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
3505 --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
3506 }
3507
3508 CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
3509 --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
3510 }
3511
3512 CREATE TABLE Package (
3513 --RegistryObject Attributes
3514     id                                    UUID PRIMARY KEY NOT NULL,
3515     name                                  LongName,
3516     description                           FreeFormText,
3517     accessControlPolicy                   UUID NOT NULL,
3518
3519 --Versionable attributes
3520     majorVersion                         INT DEFAULT 0 NOT NULL,
3521     minorVersion                         INT DEFAULT 1 NOT NULL,
3522
3523 --RegistryEntry attributes
3524     status                                INT DEFAULT 0 NOT NULL,
3525     userVersion                          ShortName,
3526     stability                             INT     DEFAULT 0 NOT NULL,
3527     expirationDate                       TIMESTAMP,
3528
3529 --Package attributes
3530 );
3531
3532
3533 CREATE PROCEDURE Package_memberbjects(packageId) {
3534 --Must return a collection of UUIDs for RegistryEntrys that are memebbers of this Package.
3535 }

```

```

3536 CREATE TABLE ExternalLink (
3537
3538
3539 --RegistryObject Attributes
3540     id                                UUID PRIMARY KEY NOT NULL,
3541     name                              LongName,
3542     description                        FreeFormText,
3543     accessControlPolicy                UUID NOT NULL,
3544
3545 --Versionable attributes
3546     majorVersion                       INT DEFAULT 0 NOT NULL,
3547     minorVersion                       INT DEFAULT 1 NOT NULL,
3548
3549 --RegistryEntry attributes
3550     status                              INT DEFAULT 0 NOT NULL,
3551     userVersion                         ShortName,
3552     stability                           INT     DEFAULT 0 NOT NULL,
3553     expirationDate                      TIMESTAMP,
3554
3555 --ExternalLink attributes
3556     externalURI                         URI NOT NULL
3557 );
3558
3559 CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
3560 --Must return a collection of UUIDs for objects in this relationship
3561 }
3562
3563 CREATE TABLE ExternalIdentifier (
3564
3565 --RegistryObject Attributes
3566     id                                UUID PRIMARY KEY NOT NULL,
3567     name                              LongName,
3568     description                        FreeFormText,
3569     accessControlPolicy                UUID NOT NULL,
3570
3571 --Versionable attributes
3572     majorVersion                       INT DEFAULT 0 NOT NULL,
3573     minorVersion                       INT DEFAULT 1 NOT NULL,
3574
3575 --RegistryEntry attributes
3576     status                              INT DEFAULT 0 NOT NULL,
3577     userVersion                         ShortName,
3578     stability                           INT     DEFAULT 0 NOT NULL,
3579     expirationDate                      TIMESTAMP,
3580
3581 --ExternalIdentifier attributes
3582     value                               ShortName NOT NULL
3583 );
3584
3585
3586 --A SlotValue row represents one value of one slot in some
3587 --RegistryEntry
3588 CREATE TABLE SlotValue (
3589
3590 --RegistryObject Attributes
3591     registryEntry                       UUID     PRIMARY KEY NOT NULL,
3592
3593 --Slot attributes
3594     name                                LongName NOT NULL PRIMARY KEY NOT NULL,
3595     value                                ShortName NOT NULL
3596 );
3597
3598 CREATE TABLE Association (
3599 --RegistryObject Attributes
3600     id                                UUID PRIMARY KEY NOT NULL,
3601     name                              LongName,
3602     description                        FreeFormText,
3603     accessControlPolicy                UUID NOT NULL,
3604
3605 --Versionable attributes

```

```

3606     majorVersion                INT DEFAULT 0 NOT NULL,
3607     minorVersion                INT DEFAULT 1 NOT NULL,
3608
3609 --RegistryEntry attributes
3610     status                        INT DEFAULT 0 NOT NULL,
3611     userVersion                  ShortName,
3612     stability                    INT     DEFAULT 0 NOT NULL,
3613     expirationDate              TIMESTAMP,
3614
3615 --Association attributes
3616     associationType              INT NOT NULL,
3617     bidirectional                BOOLEAN DEFAULT false NOT NULL,
3618     sourceObject                 UUID NOT NULL,
3619     sourceRole                   ShortName,
3620     label                        ShortName,
3621     targetObject                 UUID NOT NULL,
3622     targetRole                   ShortName
3623 );
3624
3625 --Classification is currently identical to Association
3626 CREATE TABLE Classification (
3627 --RegistryObject Attributes
3628     id                            UUID PRIMARY KEY NOT NULL,
3629     name                          LongName,
3630     description                    FreeFormText,
3631     accessControlPolicy            UUID NOT NULL,
3632
3633 --Versionable attributes
3634     majorVersion                  INT DEFAULT 0 NOT NULL,
3635     minorVersion                  INT DEFAULT 1 NOT NULL,
3636
3637 --RegistryEntry attributes
3638     status                        INT DEFAULT 0 NOT NULL,
3639     userVersion                  ShortName,
3640     stability                    INT     DEFAULT 0 NOT NULL,
3641     expirationDate              TIMESTAMP,
3642
3643 --Classification attributes. Assumes not derived from Association
3644     sourceObject                 UUID NOT NULL,
3645     targetObject                 UUID NOT NULL,
3646 );
3647
3648
3649 CREATE TABLE ClassificationNode (
3650 --RegistryObject Attributes
3651     id                            UUID PRIMARY KEY NOT NULL,
3652     name                          LongName,
3653     description                    FreeFormText,
3654     accessControlPolicy            UUID NOT NULL,
3655
3656 --Versionable attributes
3657     majorVersion                  INT DEFAULT 0 NOT NULL,
3658     minorVersion                  INT DEFAULT 1 NOT NULL,
3659
3660 --RegistryEntry attributes
3661     status                        INT DEFAULT 0 NOT NULL,
3662     userVersion                  ShortName,
3663     stability                    INT     DEFAULT 0 NOT NULL,
3664     expirationDate              TIMESTAMP,
3665
3666 --ClassificationNode attributes
3667     parent                        UUID,
3668     path                          VARCHAR(512) NOT NULL,
3669     code                          ShortName
3670 );
3671
3672 CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
3673 --Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
3674 }
3675

```

```

3676 --Begin Registry Audit Trail tables
3677
3678 CREATE TABLE AuditableEvent (
3679 --RegistryObject Attributes
3680     id                                UUID PRIMARY KEY NOT NULL,
3681     name                              LongName,
3682     description                        FreeFormText,
3683     accessControlPolicy                UUID NOT NULL,
3684
3685 --AuditableEvent attributes
3686     user                              UUID,
3687     eventType                          INT DEFAULT 0 NOT NULL,
3688     registryEntry                     UUID NOT NULL,
3689     timestamp                          TIMESTAMP NOT NULL,
3690 );
3691
3692
3693
3694 CREATE TABLE User (
3695 --RegistryObject Attributes
3696     id                                UUID PRIMARY KEY NOT NULL,
3697     name                              LongName,
3698     description                        FreeFormText,
3699     accessControlPolicy                UUID NOT NULL,
3700
3701 --User attributes
3702     organization                       UUID NOT NULL
3703
3704 --address attributes flattened
3705     address_city                       ShortName,
3706     address_country                     ShortName,
3707     address_postalCode                  ShortName,
3708     address_state                       ShortName,
3709     address_street                      ShortName,
3710
3711     email                               ShortName,
3712
3713 --fax attribute flattened
3714     fax_areaCode                       VARCHAR(4) NOT NULL,
3715     fax_countryCode                    VARCHAR(4),
3716     fax_extension                       VARCHAR(8),
3717     fax_umber                           VARCHAR(8) NOT NULL,
3718     fax_url                             URI
3719
3720 --mobilePhone attribute flattened
3721     mobilePhone_areaCode                VARCHAR(4) NOT NULL,
3722     mobilePhone_countryCode             VARCHAR(4),
3723     mobilePhone_extension                VARCHAR(8),
3724     mobilePhone_umber                   VARCHAR(8) NOT NULL,
3725     mobilePhone_url                     URI
3726
3727 --name attribute flattened
3728     name_firstName                     ShortName,
3729     name_middleName                     ShortName,
3730     name_lastName                       ShortName,
3731
3732 --pager attribute flattened
3733     pager_areaCode                      VARCHAR(4) NOT NULL,
3734     pager_countryCode                   VARCHAR(4),
3735     pager_extension                      VARCHAR(8),
3736     pager_umber                          VARCHAR(8) NOT NULL,
3737     pager_url                            URI
3738
3739 --telephone attribute flattened
3740     telephone_areaCode                  VARCHAR(4) NOT NULL,
3741     telephone_countryCode                VARCHAR(4),
3742     telephone_extension                  VARCHAR(8),
3743     telephone_umber                      VARCHAR(8) NOT NULL,
3744     telephone_url                        URI,
3745

```

```

3746 url URI,
3747
3748 );
3749
3750 CREATE TABLE Organization (
3751 --RegistryObject Attributes
3752 id UUID PRIMARY KEY NOT NULL,
3753 name LongName,
3754 description FreeFormText,
3755 accessControlPolicy UUID NOT NULL,
3756
3757 --Versionable attributes
3758 majorVersion INT DEFAULT 0 NOT NULL,
3759 minorVersion INT DEFAULT 1 NOT NULL,
3760
3761 --RegistryEntry attributes
3762 status INT DEFAULT 0 NOT NULL,
3763 userVersion ShortName,
3764 stability INT DEFAULT 0 NOT NULL,
3765 expirationDate TIMESTAMP,
3766
3767 --Organization attributes
3768
3769 --Organization.address attribute flattened
3770 address_city ShortName,
3771 address_country ShortName,
3772 address_postalCode ShortName,
3773 address_state ShortName,
3774 address_street ShortName,
3775
3776 --primary contact for Organization, points to a User.
3777 --Note many Users may belong to the same Organization
3778 contact UUID NOT NULL,
3779
3780 --Organization.fax attribute flattened
3781 fax_areaCode VARCHAR(4) NOT NULL,
3782 fax_countryCode VARCHAR(4),
3783 fax_extension VARCHAR(8),
3784 fax_umber VARCHAR(8) NOT NULL,
3785 fax_url URI,
3786
3787 --Organization.parent attribute
3788 parent UUID,
3789
3790 --Organization.telephone attribute flattened
3791 telephone_areaCode VARCHAR(4) NOT NULL,
3792 telephone_countryCode VARCHAR(4),
3793 telephone_extension VARCHAR(8),
3794 telephone_umber VARCHAR(8) NOT NULL,
3795 telephone_url URI
3796 );
3797
3798
3799 --Note that the ebRIM security view is not visible through the public query mechanism
3800 --in the current release
3801
3802
3803 --The RegistryEntry View allows polymorphic queries over all ebRIM classes derived
3804 --from RegistryEntry
3805
3806 CREATE VIEW RegistryEntry (
3807 --RegistryObject Attributes
3808 id,
3809 name,
3810 description,
3811 accessControlPolicy,
3812
3813 --Versionable attributes
3814 majorVersion,
3815 minorVersion,

```

```
3816
3817 --RegistryEntry attributes
3818     status,
3819     userVersion,
3820     stability,
3821     expirationDate
3822
3823 ) AS
3824 SELECT
3825 --RegistryObject Attributes
3826     id,
3827     name,
3828     description,
3829     accessControlPolicy,
3830
3831 --Versionable attributes
3832     majorVersion,
3833     minorVersion,
3834
3835 --RegistryEntry attributes
3836     status,
3837     userVersion,
3838     stability,
3839     expirationDate
3840
3841 FROM ExtrinsicObject
3842 UNION
3843
3844 SELECT
3845 --RegistryObject Attributes
3846     id,
3847     name,
3848     description,
3849     accessControlPolicy,
3850
3851 --Versionable attributes
3852     majorVersion,
3853     minorVersion,
3854
3855 --RegistryEntry attributes
3856     status,
3857     userVersion,
3858     stability,
3859     expirationDate
3860 FROM (Registry)Package
3861 UNION
3862
3863 SELECT
3864 --RegistryObject Attributes
3865     id,
3866     name,
3867     description,
3868     accessControlPolicy,
3869
3870 --Versionable attributes
3871     majorVersion,
3872     minorVersion,
3873
3874 --RegistryEntry attributes
3875     status,
3876     userVersion,
3877     stability,
3878     expirationDate
3879 FROM ClassificationNode;
```

3880

3881 **Appendix D Non-normative Content Based Ad Hoc Queries**

3882 The Registry SQL query capability supports the ability to search for content based not
3883 only on metadata that catalogs the content but also the data contained within the
3884 content itself. For example it is possible for a client to submit a query that searches for
3885 all Collaboration Party Profiles that define a role named "seller" within a RoleName
3886 element in the CPP document itself. Currently content-based query capability is
3887 restricted to XML content.

3888 **D.1.1 Automatic Classification of XML Content**

3889 Content-based queries are indirectly supported through the existing classification
3890 mechanism supported by the Registry.

3891 A submitting organization may define logical indexes on any XML schema or DTD when
3892 it is submitted. An instance of such a logical index defines a link between a specific
3893 attribute or element node in an XML document tree and a ClassificationNode in a
3894 classification scheme within the registry.

3895 The registry utilizes this index to automatically classify documents that are instances of
3896 the schema at the time the document instance is submitted. Such documents are
3897 classified according to the data contained within the document itself.

3898 Such automatically classified content may subsequently be discovered by clients using
3899 the existing classification-based discovery mechanism of the Registry and the query
3900 facilities of the ObjectQueryManager.

3901 [Note] This approach is conceptually similar to the way databases support
3902 indexed retrieval. DBAs define indexes on tables in the schema. When
3903 data is added to the table, the data gets automatically indexed.

3904 **D.1.2 Index Definition**

3905 This section describes how the logical indexes are defined in the SubmittedObject
3906 element defined in the Registry DTD. The complete Registry DTD is specified in
3907 Appendix A.

3908 A SubmittedObject element for a schema or DTD may define a collection of
3909 ClassificationIndexes in a ClassificationIndexList optional element. The
3910 ClassificationIndexList is ignored if the content being submitted is not of the SCHEMA
3911 objectType.

3912 The ClassificationIndex element inherits the attributes of the base class RegistryObject
3913 in [ebRIM]. It then defines specialized attributes as follows:

- 3914 1. classificationNode: This attribute references a specific ClassificationNode by its
3915 ID.

3916 2. contentIdentifier: This attribute identifies a specific data element within the
 3917 document instances of the schema using an XPATH expression as defined by
 3918 [XPT].

3919 **D.1.3 Example Of Index Definition**

3920 To define an index that automatically classifies a CPP based upon the roles defined
 3921 within its RoleName elements, the following index must be defined on the CPP schema
 3922 or DTD:

```
3923 <ClassificationIndex
3924     classificationNode='id-for-role-classification-scheme'
3925     contentIdentifier='/Role//RoleName'
3926 />
```

3927 **D.1.4 Proposed XML Definition**

```
3928 <!--
3929 A ClassificationIndexList is specified on ExtrinsicObjects of objectType
3930 'Schema' to define an automatic Classification of instance objects of the
3931 schema using the specified classificationNode as parent and a
3932 ClassificationNode created or selected by the object content as selected by
3933 the contentIdentifier
3934 -->
3935 <!ELEMENT ClassificationIndex EMPTY>
3936 <!ATTLIST ClassificationIndex
3937     %ObjectAttributes;
3938     classificationNode IDREF #REQUIRED
3939     contentIdentifier CDATA #REQUIRED
3940 >
3941
3942 <!-- ClassificationIndexList contains new ClassificationIndexes -->
3943 <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

3944 **D.1.5 Example of Automatic Classification**

3945 Assume that a CPP is submitted that defines two roles as “seller” and “buyer.” When the
 3946 CPP is submitted it will automatically be classified by two ClassificationNodes named
 3947 “buyer” and “seller” that are both children of the ClassificationNode (e.g. a node named
 3948 Role) specified in the classificationNode attribute of the ClassificationIndex. Note that if
 3949 either of the two ClassificationNodes named “buyer” and “seller” did not previously exist,
 3950 the ObjectManager would automatically create these ClassificationNodes.

3951 **Appendix E Security Implementation Guideline**

3952 This section provides a suggested blueprint for how security processing may be
 3953 implemented in the Registry. It is meant to be illustrative not prescriptive. Registries
 3954 may choose to have different implementations as long as they support the default
 3955 security roles and authorization rules described in this document.

3956 **E.1 Authentication**

- 3957 1. As soon as a message is received, the first work is the authentication. A principal
3958 object is created.
- 3959 2. If the message is signed, it is verified (including the validity of the certificate) and the
3960 DN of the certificate becomes the identity of the principal. Then the Registry is
3961 searched for the principal and if found, the roles and groups are filled in.
- 3962 3. If the message is not signed, an empty principal is created with the role
3963 RegistryGuest. This step is for symmetry and to decouple the rest of the processing.
- 3964 4. Then the message is processed for the command and the objects it will act on.

3965 **E.2 Authorization**

3966 For every object, the access controller will iterate through all the AccessControlPolicy
3967 objects with the object and see if there is a chain through the permission objects to
3968 verify that the requested method is permitted for the Principal. If any of the permission
3969 objects which the object is associated with has a common role, or identity, or group with
3970 the principal, the action is permitted.

3971 **E.3 Registry Bootstrap**

3972 When a Registry is newly created, a default Principal object should be created with the
3973 identity of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any
3974 message signed by the Registry Admin will get all the privileges.

3975 When a Registry is newly created, a singleton instance of AccessControlPolicy is
3976 created as the default AccessControlPolicy. This includes the creation of the necessary
3977 Permission instances as well as the Privileges and Privilege attributes.

3978 **E.4 Content Submission – Client Responsibility**

3979 The Registry client has to sign the contents before submission – otherwise the content
3980 will be rejected.

3981 **E.5 Content Submission – Registry Responsibility**

- 3982 1. Like any other request, the client will be first authenticated. In this case, the Principal
3983 object will get the DN from the certificate.
- 3984 2. As per the request in the message, the RegistryEntry will be created.
- 3985 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.

- 3986 4. If a principal with the identity of the SO is not available, an identity object with the
3987 SO's DN is created
- 3988 5. A principal with this identity is created

3989 **E.6 Content Delete/Deprecate – Client Responsibility**

3990 The Registry client has to sign the payload (not entire message) before submission, for
3991 authentication purposes; otherwise, the request will be rejected

3992 **E.7 Content Delete/Deprecate – Registry Responsibility**

- 3993 1. Like any other request, the client will be first authenticated. In this case, the Principal
3994 object will get the DN from the certificate. As there will be a principal with this identity
3995 in the Registry, the Principal object will get all the roles from that object
- 3996 2. As per the request in the message (delete or deprecate), the appropriate method in
3997 the RegistryObject class will be accessed.
- 3998 3. The access controller performs the authorization by iterating through the Permission
3999 objects associated with this object via the singleton default AccessControlPolicy.
- 4000 4. If authorization succeeds then the action will be permitted. Otherwise an error
4001 response is sent back with a suitable AuthorizationException error message.

4002 **Appendix F Native Language Support (NLS)**

4003 **F.1 Definitions**

4004 Although this section discusses only character set and language, the following terms
4005 have to be defined clearly.
4006

4007 **F.1.1 Coded Character Set (CCS):**

4008 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130].
4009 Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.
4010

4011 **F.1.2 Character Encoding Scheme (CES):**

4012 CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of
4013 CES are ISO-2022, UTF-8.

4014 **F.1.3 Character Set (charset):**

4015 charset is a set of rules for mapping from a sequence of octets to a sequence of
4016 characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-
4017 KR.

4018
4019 A list of registered character sets can be found at [IANA].

4020 **F.2 NLS And Request / Response Messages**

4021 For the accurate processing of data in both registry client and registry services, it is
4022 essential to know which character set is used. Although the body part of the transaction
4023 may contain the charset in xml encoding declaration, registry client and registry services
4024 shall specify charset parameter in MIME header when they use text/xml. Because as
4025 defined in [RFC 3023], if a text/xml entity is received with the charset parameter
4026 omitted, MIME processors and XML processors MUST use the default charset value of
4027 "us-ascii".

4028
4029 Ex. Content-Type: text/xml; charset=ISO-2022-JP

4030
4031 Also, when an application/xml entity is used, the charset parameter is optional, and
4032 registry client and registry services must follow the requirements in Section 4.3.3 of
4033 [REC-XML] which directly address this contingency.

4034
4035 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

4036 **F.3 NLS And Storing of RegistryEntry**

4037 This section provides NLS guidelines on how a registry should store **RegistryEntry**
4038 instances.

4039 **F.3.1 Character Set of RegistryEntry**

4040 This is basically an implementation issue because the actual character set that the
4041 **RegistryEntry** is stored with, does not affect the interface. However, it is highly
4042 recommended to use UTF-16 or UTF-8 for covering various languages.

4043 **F.3.2 Language Information of RegistryEntry**

4044 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]). If the
4045 xml:lang attribute is specified, then the registry may use that language code as the
4046 value of a special Slot with name **language** and sloType of **nls** in the **RegistryEntry**.
4047 The value must be compliant to [RFC 1766]. Slots are defined in [ebRIM].

4048 F.4 NLS And Storing of Repository Items

4049 This section provides NLS guidelines on how a registry should store repository items.

4050 F.4.1 Character Set of Repository Items

4051 Unlike the character set of **RegistryEntry**, the charset of a repository item must be
 4052 preserved as it is originally specified in the transaction. The registry may use a special
 4053 Slot with name **repositoryItemCharset**, and sloType of **nls** for the **RegistryEntry** for
 4054 storing the charset of the corresponding repository item. Value must be the one defined
 4055 in [RFC 2277], [RFC 2278]. The **repositoryItemCharset** is optional because not all
 4056 repository items require it.

4057 F.4.2 Language information of repository item

4058 Specifying only character set is not enough to tell which language is used in the
 4059 repository item. A registry may use a special Slot with name **repositoryItemLang**, and
 4060 sloType of **nls** to store that information. This attribute is optional because not all
 4061 repository items require it. Value must be compliant to [RFC 1766]

4062
 4063 This document currently specifies only the method of sending the information of
 4064 character set and language, and how it is stored in a registry. However, the language
 4065 information may be used as one of the query criteria, such as retrieving only DTD
 4066 written in French. Furthermore, a language negotiation procedure, like registry client is
 4067 asking a favorite language for messages from registry services, could be another
 4068 functionality for the future revision of this document.

4069 Appendix G Terminology Mapping

4070 While every attempt has been made to use the same terminology used in other works
 4071 there are some terminology differences.

4072 The following table shows the terminology mapping between this specification and that
 4073 used in other specifications and working groups.

4074

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	

4075

Table 1: Terminology Mapping Table

4076

4076 **References**

- 4077 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
- 4078 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 4079 [TA] ebXML Technical Architecture
- 4080 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf
- 4081 [OAS] OASIS Information Model
- 4082 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 4083 [ISO] ISO 11179 Information Model
- 4084 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 4085
- 4086 [ebRIM] ebXML Registry Information Model
- 4087 http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf
- 4088 [ebBPM] ebXML Business Process Specification Schema
- 4089 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 4090 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
- 4091 http://www.ebxml.org/project_teams/trade_partner/private/
- 4092 [ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
- 4093 <http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html>
- 4094 [CTB] Context table informal document from Core Components
- 4095 [ebMS] ebXML Messaging Service Specification, Version 0.21
- 4096 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf
- 4097 [ERR] ebXML TRP Error Handling Specification
- 4098 http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf
- 4099 [SEC] ebXML Risk Assessment Technical Report, Version 3.6
- 4100 <http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html>
- 4101 [XPT] XML Path Language (XPath) Version 1.0
- 4102 <http://www.w3.org/TR/xpath>
- 4103 [SQL] Structured Query Language (FIPS PUB 127-2)
- 4104 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>
- 4105
- 4106 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
- 4107 (SQL/PSM) [ISO/IEC 9075-4:1996]

- 4108
4109 [IANA] IANA (Internet Assigned Numbers Authority).
4110 Official Names for Character Sets, ed. Keld Simonsen et al.
4111 <ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets>
4112
- 4113 [RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
4114 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
4115 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
4116
- 4117 [RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
4118 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
4119 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
4120
- 4121 [RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
4122 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
4123 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
4124
- 4125 [RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
4126 XML Media Types, ed. M. Murata. 2001.
4127 <ftp://ftp.isi.edu/in-notes/rfc3023.txt>
4128
- 4129 [REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second
4130 Edition)
4131 <http://www.w3.org/TR/REC-xml>
4132
- 4133 [UUID] DCE 128 bit Universal Unique Identifier
4134 http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh_20
4135 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>

4136 Disclaimer

4137 The views and specification expressed in this document are those of the authors and
4138 are not necessarily those of their employers. The authors and their employers
4139 specifically disclaim responsibility for any problems arising from correct or incorrect
4140 implementation or use of this design.

4141

4141 Contact Information

4142 Team Leader

4143 Name: Scott Nieman
4144 Company: Norstan Consulting
4145 Street: 5101 Shady Oak Road
4146 City, State, Postal Code: Minnetonka, MN 55343
4147 Country: USA
4148 Phone: 952.352.5889
4149 Email: Scott.Nieman@Norstan

4150

4151 Vice Team Lead

4152 Name: Yutaka Yoshida
4153 Company: Sun Microsystems
4154 Street: 901 San Antonio Road, MS UMPK17-102
4155 City, State, Postal Code: Palo Alto, CA 94303
4156 Country: USA
4157 Phone: 650.786.5488
4158 Email: Yutaka.Yoshida@eng.sun.com

4159

4160 Editor

4161 Name: Farrukh S. Najmi
4162 Company: Sun Microsystems
4163 Street: 1 Network Dr., MS BUR02-302
4164 City, State, Postal Code: Burlington, MA, 01803-0902
4165 Country: USA
4166 Phone: 781.442.0703
4167 Email: najmi@east.sun.com

4168

4169

4169 **Copyright Statement**

4170 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

4171

4172 This document and translations of it may be copied and furnished to others, and
4173 derivative works that comment on or otherwise explain it or assist in its implementation
4174 MAY be prepared, copied, published and distributed, in whole or in part, without
4175 restriction of any kind, provided that the above copyright notice and this paragraph are
4176 included on all such copies and derivative works. However, this document itself MAY
4177 not be modified in any way, such as by removing the copyright notice or references to
4178 ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other
4179 than English.

4180

4181 The limited permissions granted above are perpetual and will not be revoked by ebXML
4182 or its successors or assigns.

4183

4184 This document and the information contained herein is provided on an
4185 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
4186 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
4187 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
4188 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
4189 PURPOSE.