



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

**OASIS/ebXML Registry Information Model v2.0<sup>24</sup>**  
**Bug Fixes To Approved OASIS Standard**  
**OASIS/ebXML Registry Technical Committee**  
**May 2002**

**1 Status of this Document**

Distribution of this document is unlimited.

***This version:***

<http://www.oasis-open.org/committees/regrep/documents/2.0<sup>24</sup>/specs/ebRIM.pdf>

***Latest version:***

<http://www.oasis-open.org/committees/regrep/documents/2.0<sup>24</sup>/specs/ebRIM.pdf>

## 20 **2 OASIS/ebXML Registry Technical Committee**

21 This document has no standing and currently represents works-in-progress of the  
22 OASIS ebXML Registry TC. A future version of this document will be finalized  
23 and approved by the Registry TC as version 2.1.

24

25 At the time of v2.0 committee approval, the following were members of the  
26 OASIS/ebXML Registry Technical Committee:

27

28 Kathryn Breining, Boeing ~~(TC Chair)~~

29 Lisa Carnahan, US NIST ~~(TC Chair)~~

30 Joseph M. Chiusano, LMI

31 Suresh Damodaran, Sterling Commerce

32 Mike DeNicola Fujitsu

33 Anne Fischer, Drummond Group

34 Sally Fuger, AIAG

35 Jong Kim InnoDigital

36 Kyu-Chul Lee, Chungnam National University

37 Joel Munter, Intel

38 Farrukh Najmi, Sun Microsystems

39 Joel Neu, Vitria Technologies

40 Sanjay Patil, IONA

41 Neal Smith, ChevronTexaco

42 Nikola Stojanovic, Encoda Systems, Inc.

43 Prasad Yendluri, webMethods

44 Yutaka Yoshida, Sun Microsystems

45

### 46 **2.1 Contributors**

47 The following persons contributed to the content of this document, but are not  
48 voting members of the OASIS/ebXML Registry Technical Committee.

49

50 Len Gallagher, NIST

51 Sekhar Vajjhala, Sun Microsystems

52

53

## 53 **Table of Contents**

54

55	<b>1</b>	<b>STATUS OF THIS DOCUMENT</b> .....	<b>1</b>
56	<b>2</b>	<b>OASIS/EBXML REGISTRY TECHNICAL COMMITTEE</b> .....	<b>2</b>
57	2.1	CONTRIBUTORS.....	2
58	<b>3</b>	<b>INTRODUCTION</b> .....	<b>9</b>
59	3.1	SUMMARY OF CONTENTS OF DOCUMENT.....	9
60	3.2	GENERAL CONVENTIONS .....	9
61	3.2.1	<i>Naming Conventions</i> .....	9
62	3.3	AUDIENCE.....	10
63	3.4	RELATED DOCUMENTS .....	10
64	<b>4</b>	<b>DESIGN OBJECTIVES</b> .....	<b>10</b>
65	4.1	GOALS .....	10
66	<b>5</b>	<b>SYSTEM OVERVIEW</b> .....	<b>11</b>
67	5.1	ROLE OF EBXML <i>REGISTRY</i> .....	11
68	5.2	REGISTRY SERVICES .....	11
69	5.3	WHAT THE REGISTRY INFORMATION MODEL DOES .....	11
70	5.4	HOW THE REGISTRY INFORMATION MODEL WORKS .....	11
71	5.5	WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED .....	11
72	5.6	CONFORMANCE TO AN EBXML <i>REGISTRY</i> .....	12
73	<b>6</b>	<b>REGISTRY INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW</b> .....	<b>12</b>
74	6.1	REGISTRYOBJECT .....	13
75	6.2	SLOT .....	13
76	6.3	ASSOCIATION.....	13
77	6.4	EXTERNALIDENTIFIER .....	13
78	6.5	EXTERNALLINK .....	13
79	6.6	CLASSIFICATIONSCHEME.....	13
80	6.7	CLASSIFICATIONNODE.....	14
81	6.8	CLASSIFICATION .....	14
82	6.9	REGISTRYPACKAGE.....	14
83	6.10	AUDITABLEEVENT.....	14
84	6.11	USER.....	14
85	6.12	POSTALADDRESS .....	14
86	6.13	EMAILADDRESS .....	14
87	6.14	ORGANIZATION.....	15
88	6.15	SERVICE.....	15
89	6.16	SERVICEBINDING.....	15
90	6.17	SPECIFICATIONLINK .....	15

91	<b>7</b>	<b>REGISTRY INFORMATION MODEL: DETAIL VIEW.....</b>	<b>15</b>
92	7.1	ATTRIBUTE AND METHODS OF INFORMATION MODEL CLASSES .....	16
93	7.2	DATA TYPES .....	17
94	7.3	INTERNATIONALIZATION (I18N) SUPPORT.....	17
95	7.3.1	Class <i>InternationalString</i> .....	17
96	7.3.2	Class <i>LocalizedString</i> .....	18
97	7.4	CLASS REGISTRYOBJECT .....	18
98	7.4.1	Attribute <i>Summary</i> .....	19
99	7.4.2	Attribute <i>accessControlPolicy</i> .....	19
100	7.4.3	Attribute <i>description</i> .....	19
101	7.4.4	Attribute <i>id</i> .....	20
102	7.4.5	Attribute <i>name</i> .....	20
103	7.4.6	Attribute <i>objectType</i> .....	20
104	7.4.7	Method <i>Summary</i> .....	21
105	7.5	CLASS REGISTRYENTRY .....	22
106	7.5.1	Attribute <i>Summary</i> .....	22
107	7.5.2	Attribute <i>expiration</i> .....	23
108	7.5.3	Attribute <i>majorVersion</i> .....	23
109	7.5.4	Attribute <i>minorVersion</i> .....	23
110	7.5.5	Attribute <i>stability</i> .....	23
111	7.5.6	Attribute <i>status</i> .....	24
112	7.5.7	Attribute <i>userVersion</i> .....	24
113	7.5.8	Method <i>Summary</i> .....	24
114	7.6	CLASS SLOT.....	<del>2525</del>
115	7.6.1	Attribute <i>Summary</i> .....	<del>2525</del>
116	7.6.2	Attribute <i>name</i> .....	25
117	7.6.3	Attribute <i>slotType</i> .....	25
118	7.6.4	Attribute <i>values</i> .....	25
119	7.7	CLASS EXTRINSICOBJECT .....	<del>2526</del>
120	7.7.1	Attribute <i>Summary</i> .....	<del>2626</del>
121	7.7.2	Attribute <i>isOpaque</i> .....	26
122	7.7.3	Attribute <i>mimeType</i> .....	26
123	7.8	CLASS REGISTRYPACKAGE.....	26
124	7.8.1	Attribute <i>Summary</i> .....	<del>2627</del>
125	7.8.2	Method <i>Summary</i> .....	<del>2727</del>
126	7.9	CLASS EXTERNALIDENTIFIER .....	<del>2727</del>
127	7.9.1	Attribute <i>Summary</i> .....	27
128	7.9.2	Attribute <i>identificationScheme</i> .....	<del>2728</del>
129	7.9.3	Attribute <i>registryObject</i> .....	<del>2828</del>
130	7.9.4	Attribute <i>value</i> .....	<del>2828</del>
131	7.10	CLASS EXTERNALLINK .....	<del>2828</del>
132	7.10.1	Attribute <i>Summary</i> .....	<del>2828</del>
133	7.10.2	Attribute <i>externalURI</i> .....	28
134	7.10.3	Method <i>Summary</i> .....	28
135	<b>8</b>	<b>REGISTRY AUDIT TRAIL.....</b>	<del><b>2929</b></del>

136	8.1	CLASS AUDITABLEEVENT.....	<del>2929</del>
137	8.1.1	Attribute Summary.....	29
138	8.1.2	Attribute eventType.....	<del>3030</del>
139	8.1.3	Attribute registryObject.....	<del>3030</del>
140	8.1.4	Attribute timestamp.....	<del>3030</del>
141	8.1.5	Attribute user.....	<del>3030</del>
142	8.2	CLASS USER.....	30
143	8.2.1	Attribute Summary.....	30
144	8.2.2	Attribute address.....	<del>3131</del>
145	8.2.3	Attribute emailAddresses.....	<del>3131</del>
146	8.2.4	Attribute organization.....	<del>3131</del>
147	8.2.5	Attribute personName.....	<del>3131</del>
148	8.2.6	Attribute telephoneNumbers.....	31
149	8.2.7	Attribute url.....	31
150	8.3	CLASS ORGANIZATION.....	31
151	8.3.1	Attribute Summary.....	<del>3232</del>
152	8.3.2	Attribute address.....	<del>3232</del>
153	8.3.3	Attribute parent.....	<del>3232</del>
154	8.3.4	Attribute primaryContact.....	<del>3232</del>
155	8.3.5	Attribute telephoneNumbers.....	<del>3232</del>
156	8.4	CLASS POSTALADDRESS.....	32
157	8.4.1	Attribute Summary.....	32
158	8.4.2	Attribute city.....	<del>3333</del>
159	8.4.3	Attribute country.....	<del>3333</del>
160	8.4.4	Attribute postalCode.....	<del>3333</del>
161	8.4.5	Attribute state.....	<del>3333</del>
162	8.4.6	Attribute street.....	<del>3333</del>
163	8.4.7	Attribute streetNumber.....	<del>3333</del>
164	8.4.8	Method Summary.....	33
165	8.5	CLASS TELEPHONENUMBER.....	33
166	8.5.1	Attribute Summary.....	<del>3334</del>
167	8.5.2	Attribute areaCode.....	<del>3434</del>
168	8.5.3	Attribute countryCode.....	<del>3434</del>
169	8.5.4	Attribute extension.....	<del>3434</del>
170	8.5.5	Attribute number.....	34
171	8.5.6	Attribute phoneType.....	34
172	8.6	CLASS EMAILADDRESS.....	34
173	8.6.1	Attribute Summary.....	34
174	8.6.2	Attribute address.....	<del>3535</del>
175	8.6.3	Attribute type.....	<del>3535</del>
176	8.7	CLASS PERSONNAME.....	<del>3535</del>
177	8.7.1	Attribute Summary.....	<del>3535</del>
178	8.7.2	Attribute firstName.....	<del>3535</del>
179	8.7.3	Attribute lastName.....	35
180	8.7.4	Attribute middleName.....	35
181	8.8	CLASS SERVICE.....	35

182	8.8.1	<i>Attribute Summary</i> .....	35
183	8.8.2	<i>Method Summary</i> .....	<u>3636</u>
184	8.9	CLASS SERVICEBINDING.....	<u>3636</u>
185	8.9.1	<i>Attribute Summary</i> .....	36
186	8.9.2	<i>Attribute accessURI</i> .....	36
187	8.9.3	<i>Attribute targetBinding</i> .....	36
188	8.9.4	<i>Method Summary</i> .....	<u>3737</u>
189	8.10	CLASS SPECIFICATIONLINK.....	<u>3737</u>
190	8.10.1	<i>Attribute Summary</i> .....	37
191	8.10.2	<i>Attribute specificationObject</i> .....	37
192	8.10.3	<i>Attribute usageDescription</i> .....	37
193	8.10.4	<i>Attribute usageParameters</i> .....	<u>3838</u>
194	<b>9</b>	<b>ASSOCIATION OF REGISTRY OBJECTS</b> .....	<u><b>3939</b></u>
195	9.1	EXAMPLE OF AN ASSOCIATION.....	<u>3939</u>
196	9.2	SOURCE AND TARGET OBJECTS.....	<u>3939</u>
197	9.3	ASSOCIATION TYPES.....	<u>3939</u>
198	9.4	INTRAMURAL ASSOCIATION.....	<u>4040</u>
199	9.5	EXTRAMURAL ASSOCIATION.....	<u>4040</u>
200	9.6	CONFIRMATION OF AN ASSOCIATION.....	<u>4141</u>
201	9.6.1	<i>Confirmation of Intramural Associations</i> .....	<u>4141</u>
202	9.6.2	<i>Confirmation of Extramural Associations</i> .....	<u>4242</u>
203	9.6.3	<i>Deleting an Extramural Associations</i> .....	<u>4242</u>
204	9.7	VISIBILITY OF UNCONFIRMED ASSOCIATIONS.....	<u>4242</u>
205	9.8	POSSIBLE CONFIRMATION STATES.....	<u>4242</u>
206	9.9	CLASS ASSOCIATION.....	<u>4343</u>
207	9.9.1	<i>Attribute Summary</i> .....	<u>4343</u>
208	9.9.2	<i>Attribute associationType</i> .....	<u>4343</u>
209	9.9.3	<i>Attribute sourceObject</i> .....	<u>4444</u>
210	9.9.4	<i>Attribute targetObject</i> .....	<u>4444</u>
211	9.9.5	<i>Attribute isConfirmedBySourceOwner</i> .....	<u>4444</u>
212	9.9.6	<i>Attribute isConfirmedByTargetOwner</i> .....	<u>4545</u>
213	<b>10</b>	<b>CLASSIFICATION OF REGISTRYOBJECT</b> .....	<u><b>4545</b></u>
214	10.1	CLASS CLASSIFICATIONSCHEME.....	<u>4848</u>
215	10.1.1	<i>Attribute Summary</i> .....	<u>4848</u>
216	10.1.2	<i>Attribute isInternal</i> .....	<u>4848</u>
217	10.1.3	<i>Attribute nodeType</i> .....	<u>4848</u>
218	10.2	CLASS CLASSIFICATIONNODE.....	<u>4949</u>
219	10.2.1	<i>Attribute Summary</i> .....	<u>4949</u>
220	10.2.2	<i>Attribute parent</i> .....	<u>4949</u>
221	10.2.3	<i>Attribute code</i> .....	<u>4949</u>
222	10.2.4	<i>Attribute path</i> .....	<u>4949</u>
223	10.2.5	<i>Method Summary</i> .....	<u>5050</u>
224	10.2.6	<i>Canonical Path Syntax</i> .....	<u>5050</u>
225	10.3	CLASS CLASSIFICATION.....	<u>5151</u>

226	10.3.1	Attribute Summary.....	<del>5151</del>
227	10.3.2	Attribute classificationScheme.....	<del>5252</del>
228	10.3.3	Attribute classificationNode .....	<del>5252</del>
229	10.3.4	Attribute classifiedObject .....	<del>5252</del>
230	10.3.5	Attribute nodeRepresentation .....	<del>5252</del>
231	10.3.6	Context Sensitive Classification .....	<del>5252</del>
232	10.3.7	Method Summary.....	<del>5454</del>
233	10.4	EXAMPLE OF CLASSIFICATION SCHEMES.....	<del>5455</del>
234	<b>11</b>	<b>INFORMATION MODEL: SECURITY VIEW .....</b>	<b><del>5555</del></b>
235	11.1	CLASS ACCESSCONTROLPOLICY.....	<del>5656</del>
236	11.2	CLASS PERMISSION .....	<del>5757</del>
237	11.3	CLASS PRIVILEGE .....	<del>5757</del>
238	11.4	CLASS PRIVILEGEATTRIBUTE.....	<del>5858</del>
239	11.5	CLASS ROLE .....	<del>5858</del>
240	11.5.1	A security Role PrivilegeAttribute.....	<del>5858</del>
241	11.6	CLASS GROUP .....	<del>5858</del>
242	11.6.1	A security Group PrivilegeAttribute.....	<del>5858</del>
243	11.7	CLASS IDENTITY .....	<del>5959</del>
244	11.7.1	A security Identity PrivilegeAttribute.....	<del>5959</del>
245	11.8	CLASS PRINCIPAL .....	<del>5959</del>
246	<b>12</b>	<b>REFERENCES .....</b>	<b><del>6060</del></b>
247	<b>13</b>	<b>DISCLAIMER .....</b>	<b><del>6060</del></b>
248	<b>14</b>	<b>CONTACT INFORMATION.....</b>	<b><del>6161</del></b>
249		<b>COPYRIGHT STATEMENT.....</b>	<b><del>6262</del></b>

## 250 **Table of Figures**

251	Figure 1: Information Model High Level Public View.....	12
252	Figure 2: Information Model <i>Inheritance</i> View.....	16
253	Figure 3: Example of RegistryObject Association.....	<del>3939</del>
254	Figure 4: Example of Intramural Association.....	<del>4040</del>
255	Figure 5: Example of Extramural Association .....	<del>4141</del>
256	Figure 6: Example showing a <i>Classification</i> Tree.....	<del>4646</del>
257	Figure 7: Information Model <i>Classification</i> View.....	<del>4747</del>
258	Figure 8: Classification <i>Instance</i> Diagram .....	<del>4747</del>
259	Figure 9: Context Sensitive <i>Classification</i> .....	<del>5353</del>
260	Figure 10: Information Model: Security View.....	<del>5656</del>

## 261 **Table of Tables**

262	Table 1: Sample <i>Classification</i> Schemes.....	<del>5555</del>
-----	--	-----------------

263

264



## 264 **3 Introduction**

### 265 **3.1 Summary of Contents of Document**

266 This document specifies the information model for the ebXML *Registry*.

267

268 A separate document, ebXML Registry Services Specification [ebRS], describes  
269 how to build *Registry Services* that provide access to the information content in  
270 the ebXML *Registry*.

### 271 **3.2 General Conventions**

272 The following conventions are used throughout this document:

273

274 UML diagrams are used as a way to concisely describe concepts. They are not  
275 intended to convey any specific *Implementation* or methodology requirements.

276

277 The term "*repository item*" is used to refer to an object that resides in a  
278 repository for storage and safekeeping (e.g., an XML document or a DTD). Every  
279 repository item is described in the Registry by a RegistryObject instance.

280

281 The term "*RegistryEntry*" is used to refer to an object that provides metadata  
282 about a *repository item*.

283

284 The information model does not deal with the actual content of the repository. All  
285 *Elements* of the information model represent metadata about the content and not  
286 the content itself.

287

288 *Capitalized Italic* words are defined in the ebXML Glossary.

289

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

293

294 Software practitioners MAY use this document in combination with other ebXML  
295 specification documents when creating ebXML compliant software.

#### 296 **3.2.1 Naming Conventions**

297

298 In order to enforce a consistent capitalization and naming convention in this  
299 document, "Upper Camel Case" (*UCC*) and "Lower Camel Case" (*LCC*)

300 Capitalization styles are used in the following conventions:

- 301 ○ Element name is in *UCC* convention
- 302 (example: <UpperCamelCaseElement/>)
- 303 ○ Attribute name is in *LCC* convention

- 304 (example: <UpperCamelCaseElement  
305 lowerCamelCaseAttribute="whatEver"/>)  
306 ○ Class, Interface names use UCC convention  
307 (examples: ClassificationNode, Versionable)  
308 ○ Method name uses LCC convention  
309 (example: getName(), setName()).  
310

311 Also, *Capitalized Italics* words are defined in the ebXML Glossary [ebGLOSS].

### 312 **3.3 Audience**

313 The target audience for this specification is the community of software  
314 developers who are:

- 315 ○ Implementers of ebXML *Registry Services*  
316 ○ Implementers of ebXML *Registry Clients*

### 317 **3.4 Related Documents**

318 The following specifications provide some background and related information to  
319 the reader:

- 320  
321 a) ebXML Registry Services Specification [ebRS] - defines the actual  
322 *Registry Services* based on this information model  
323 b) ebXML Collaboration-Protocol Profile and Agreement Specification  
324 [ebCPP] - defines how profiles can be defined for a *Party* and how two  
325 *Parties'* profiles may be used to define a *Party* agreement  
326

## 327 **4 Design Objectives**

### 328 **4.1 Goals**

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

- 330 ○ Communicate what information is in the *Registry* and how that information  
331 is organized  
332 ○ Leverage as much as possible the work done in the OASIS [OAS] and the  
333 ISO 11179 [ISO] Registry models  
334 ○ Align with relevant works within other ebXML working groups  
335 ○ Be able to evolve to support future ebXML *Registry* requirements  
336 ○ Be compatible with other ebXML specifications  
337

## 338 **5 System Overview**

### 339 **5.1 Role of ebXML Registry**

340

341 The *Registry* provides a stable store where information submitted by a  
342 *Submitting Organization* is made persistent. Such information is used to facilitate  
343 ebXML-based *Business to Business* (B2B) partnerships and transactions.

344 Submitted content may be *XML* schema and documents, process descriptions,  
345 ebXML *Core Components*, context descriptions, *UML* models, information about  
346 parties and even software components.

### 347 **5.2 Registry Services**

348 A set of *Registry Services* that provide access to *Registry* content to clients of the  
349 *Registry* is defined in the ebXML Registry Services Specification [ebRS]. This  
350 document does not provide details on these services but may occasionally refer  
351 to them.

### 352 **5.3 What the Registry Information Model Does**

353 The Registry Information Model provides a blueprint or high-level schema for the  
354 ebXML *Registry*. Its primary value is for implementers of ebXML *Registries*. It  
355 provides these implementers with information on the type of metadata that is  
356 stored in the *Registry* as well as the relationships among metadata *Classes*.

357 The Registry information model:

- 358 ○ Defines what types of objects are stored in the *Registry*
- 359 ○ Defines how stored objects are organized in the *Registry*

360

### 361 **5.4 How the Registry Information Model Works**

362 Implementers of the ebXML *Registry* MAY use the information model to  
363 determine which *Classes* to include in their *Registry Implementation* and what  
364 attributes and methods these *Classes* may have. They MAY also use it to  
365 determine what sort of database schema their *Registry Implementation* may  
366 need.

367 [Note]The information model is meant to be  
368 illustrative and does not prescribe any  
369 specific *Implementation* choices.

370

### 371 **5.5 Where the Registry Information Model May Be Implemented**

372 The Registry Information Model MAY be implemented within an ebXML *Registry*  
373 in the form of a relational database schema, object database schema or some

374 other physical schema. It MAY also be implemented as interfaces and *Classes*  
 375 within a *Registry Implementation*.

376 **5.6 Conformance to an ebXML Registry**

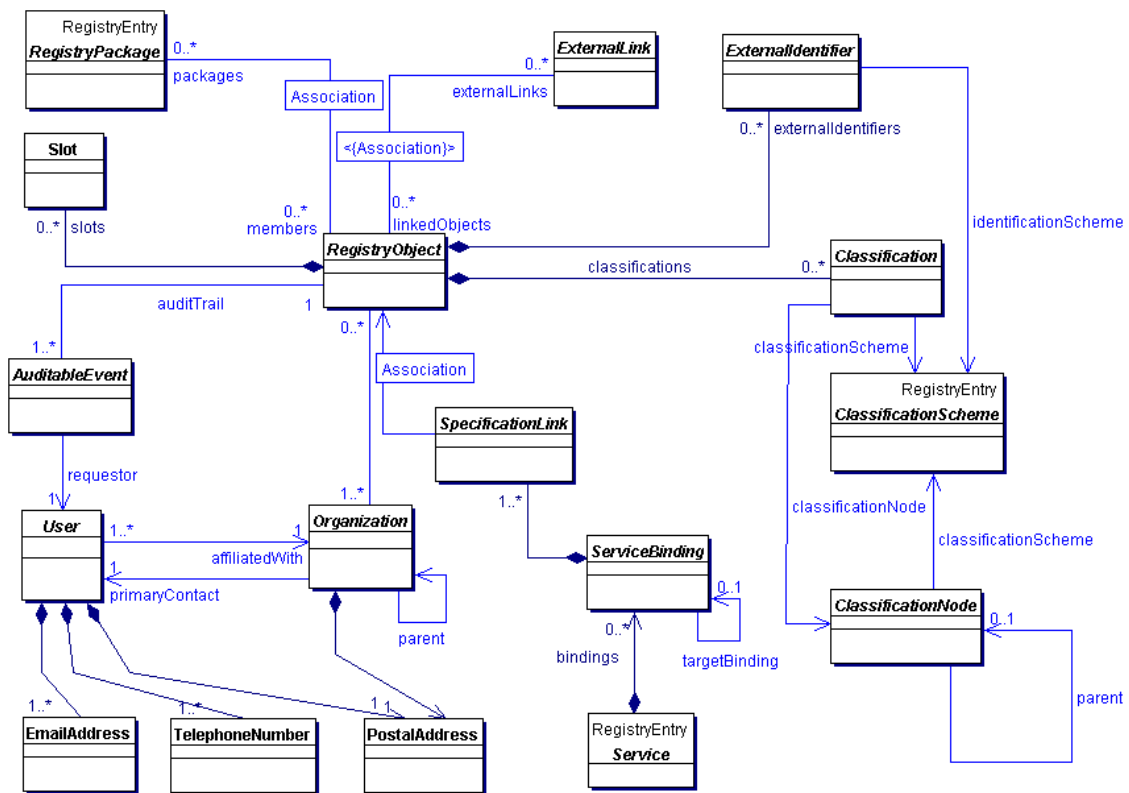
377 If an *Implementation* claims *Conformance* to this specification then it supports all  
 378 required information model *Classes* and interfaces, their attributes and their  
 379 semantic definitions that are visible through the ebXML *Registry Services*.

380 **6 Registry Information Model: High Level Public View**

381 This section provides a high level public view of the most visible objects in the  
 382 *Registry*.

383 **Figure 1** ~~Figure-4~~ shows the high level public view of the objects in the *Registry*  
 384 and their relationships as a *UML Class Diagram*. It does not show *Inheritance*,  
 385 *Class* attributes or *Class* methods.

387 The reader is again reminded that the information model is not modeling actual  
 388 repository items.  
 389



390

391

**Figure 1: Information Model High Level Public View**

## 392 **6.1 RegistryObject**

393 The RegistryObject class is an abstract base class used by most classes in the  
394 model. It provides minimal metadata for registry objects. It also provides methods  
395 for accessing related objects that provide additional dynamic metadata for the  
396 registry object.

## 397 **6.2 Slot**

398 Slot instances provide a dynamic way to add arbitrary attributes to  
399 RegistryObject instances. This ability to add attributes dynamically to  
400 RegistryObject instances enables extensibility within the Registry Information  
401 Model. For example, if a company wants to add a “copyright” attribute to each  
402 RegistryObject instance that it submits, it can do so by adding a slot with name  
403 “copyright” and value containing the copyrights statement.

## 404 **6.3 Association**

405 Association instances are RegistryObject instances that are used to define many-  
406 to-many associations between objects in the information model. Associations are  
407 described in detail in section 9.

## 408 **6.4 ExternalIdentifier**

409 ExternalIdentifier instances provide additional identifier information to a  
410 RegistryObject instance, such as DUNS number, Social Security Number, or an  
411 alias name of the organization.

## 412 **6.5 ExternalLink**

413 ExternalLink instances are RegistryObject instances that model a named URI to  
414 content that is not managed by the *Registry*. Unlike managed content, such  
415 external content may change or be deleted at any time without the knowledge of  
416 the *Registry*. A RegistryObject instance may be associated with any number of  
417 ExternalLinks.

418 Consider the case where a *Submitting Organization* submits a repository item  
419 (e.g., a *DTD*) and wants to associate some external content to that object (e.g.,  
420 the *Submitting Organization's* home page). The ExternalLink enables this  
421 capability. A potential use of the ExternalLink capability may be in a GUI tool that  
422 displays the ExternalLinks to a RegistryObject. The user may click on such links  
423 and navigate to an external web page referenced by the link.

## 424 **6.6 ClassificationScheme**

425 ClassificationScheme instances are RegistryEntry instances that describe a  
426 structured way to classify or categorize RegistryObject instances. The structure  
427 of the classification scheme may be defined internal or external to the registry,  
428 resulting in a distinction between internal and external classification schemes. A  
429 very common example of a classification scheme in science is the *Classification*  
430 *of living things* where living things are categorized in a tree like structure. Another

431 example is the Dewey Decimal system used in libraries to categorize books and  
432 other publications. ClassificationScheme is described in detail in section 10.

### 433 **6.7 ClassificationNode**

434 ClassificationNode instances are RegistryObject instances that are used to  
435 define tree structures under a ClassificationScheme, where each node in the tree  
436 is a ClassificationNode and the root is the ClassificationScheme. *Classification*  
437 trees constructed with ClassificationNodes are used to define the structure of  
438 *Classification* schemes or ontologies. ClassificationNode is described in detail in  
439 section 10.

### 440 **6.8 Classification**

441 Classification instances are RegistryObject instances that are used to classify  
442 other RegistryObject instances. A Classification instance identifies a  
443 ClassificationScheme instance and taxonomy value defined within the  
444 classification scheme. Classifications can be internal or external depending on  
445 whether the referenced classification scheme is internal or external.  
446 Classification is described in detail in section 10.

### 447 **6.9 RegistryPackage**

448 RegistryPackage instances are RegistryEntry instances that group logically  
449 related RegistryObject instances together.

### 450 **6.10 AuditableEvent**

451 AuditableEvent instances are RegistryObject instances that are used to provide  
452 an audit trail for RegistryObject instances. AuditableEvent is described in detail in  
453 section 8.

### 454 **6.11 User**

455 User instances are RegistryObject instances that are used to provide information  
456 about registered users within the *Registry*. User objects are used in audit trail for  
457 RegistryObject instances. User is described in detail in section 8.

### 458 **6.12 PostalAddress**

459 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal  
460 address.

### 461 **6.13 EmailAddress**

462 EmailAddress is a simple reusable *Entity Class* that defines attributes of an email  
463 address.

## 464 **6.14 Organization**

465 Organization instances are RegistryObject instances that provide information on  
466 organizations such as a *Submitting Organization*. Each Organization instance  
467 may have a reference to a parent Organization.

## 468 **6.15 Service**

469 Service instances are RegistryEntry instances that provide information on  
470 services (e.g., web services).

## 471 **6.16 ServiceBinding**

472 ServiceBinding instances are RegistryObject instances that represent technical  
473 information on a specific way to access a specific interface offered by a Service  
474 instance. A Service has a collection of ServiceBindings.  
475

## 476 **6.17 SpecificationLink**

477 A SpecificationLink provides the linkage between a ServiceBinding and one of its  
478 technical specifications that describes how to use the service with that  
479 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink  
480 instance that describes how to access the service using a technical specification  
481 in the form of a WSDL document or a CORBA IDL document.  
482

## 483 **7 Registry Information Model: Detail View**

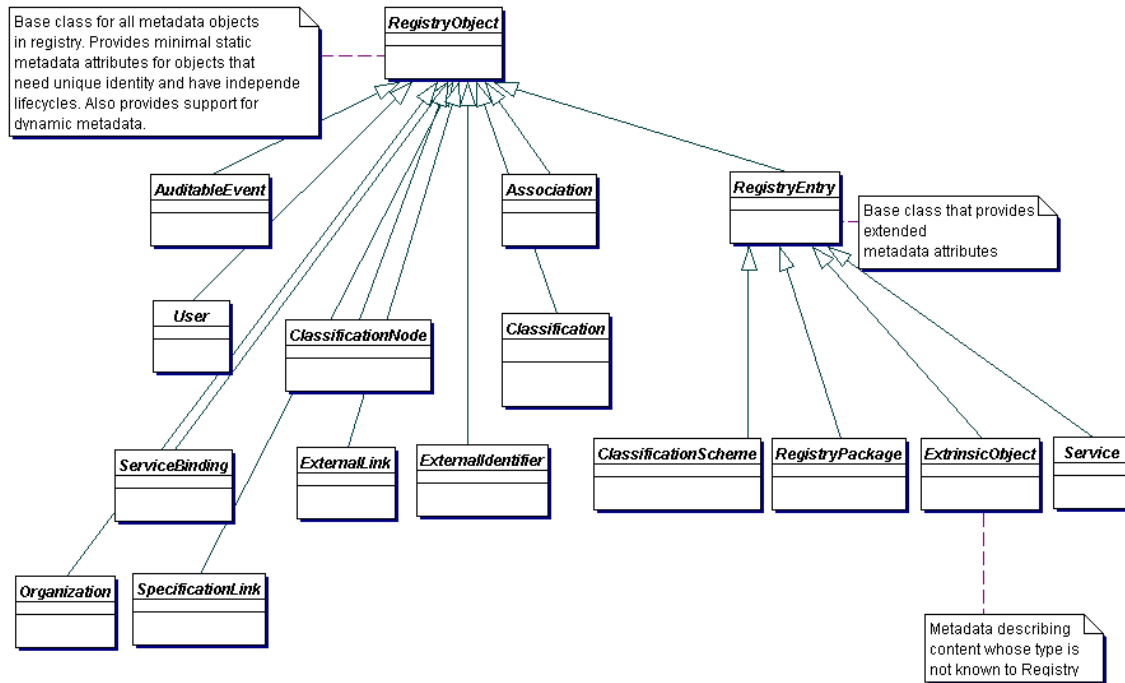
484 This section covers the information model *Classes* in more detail than the Public  
485 View. The detail view introduces some additional *Classes* within the model that  
486 were not described in the public view of the information model.  
487

488 [Figure 2](#) shows the *Inheritance* or “is a” relationships between the  
489 *Classes* in the information model. Note that it does not show the other types of  
490 relationships, such as “has a” relationships, since they have already been shown  
491 in a previous figure. *Class* attributes and *class* methods are also not shown.  
492 Detailed description of methods and attributes of most interfaces and *Classes* will  
493 be displayed in tabular form following the description of each *Class* in the model.  
494

495 The class Association will be covered in detail separately in section 9. The  
496 classes ClassificationScheme, Classification, and ClassificationNode will be  
497 covered in detail separately in section 10.  
498

499 The reader is again reminded that the information model is not modeling actual  
500 repository items.





501  
502  
503

Figure 2: Information Model *Inheritance View*

504 **7.1 Attribute and Methods of Information Model Classes**

505 Information model classes are defined primarily in terms of the attributes they  
506 carry. These attributes provide state information on instances of these classes.  
507 Implementations of a registry often map class attributes to attributes in an XML  
508 store or columns in a relational store.

509  
510 Information model classes may also have methods defined for them. These  
511 methods provide additional behavior for the class they are defined within.  
512 Methods are currently used in mapping to filter query and the SQL query  
513 capabilities defined in [ebRS].

514  
515 Since the model supports inheritance between classes, it is usually the case that  
516 a class in the model inherits attributes and methods from its base classes, in  
517 addition to defining its own specialized attributes and methods.

518



## 518 7.2 Data Types

519 The following table lists the various data types used by the attributes within  
 520 information model classes:  
 521

Data Type	XML Schema Data Type	Description	Length
Boolean	boolean	Used for a true or false value	
String4	string	Used for 4 character long strings	4 characters
String8	string	Used for 8 character long strings	8 characters
String16	string	Used for 16 character long strings	16 characters
String32	string	Used for 32 character long strings	32 characters
String	string	Used for unbounded Strings	unbounded
ShortName	string	A short text string	64 characters
LongName	string	A long text string	128 characters
FreeFormText	string	A very long text string for free-form text	256 characters
UUID	string	DCE 128 Bit Universally unique Ids used for referencing another object	64 characters
URI	string	Used for URL and URN values	256 characters
Integer	integer	Used for integer values	4 bytes
DateTime	dateTime	Used for a timestamp value such as Date	

522

## 523 7.3 Internationalization (I18N) Support

524 Some information model classes have String attributes that are I18N capable and  
 525 may be localized into multiple native languages. Examples include the name and  
 526 description attributes of the RegistryObject class in 7.4.

527

528 The information model defines the InternationalString and the LocalizedString  
 529 interfaces to support I18N capable attributes within the information model  
 530 classes. These classes are defined below.

### 531 7.3.1 Class InternationalString

532 This class is used as a replacement for the String type whenever a String  
 533 attribute needs to be I18N capable. An instance of the InternationalString class  
 534 composes within it a Collection of LocalizedString instances, where each String  
 535 is specific to a particular locale. The InternationalString class provides set/get

536 methods for adding or getting locale specific String values for the  
537 InternationalString instance.

#### 538 7.3.1.1 Attribute Summary

539

Attribute	Data Type	Required	Default Value	Specified By	Mutable
localized-Strings	Collection of Localized-String	No		Client	Yes

540

#### 541 7.3.1.2 Attribute localizedStrings

542 Each InternationalString instance may have localizedString attribute that is a  
543 Collection of zero or more LocalizedString instances.

### 544 7.3.2 Class LocalizedString

545 This class is used as a simple wrapper class that associates a String with its  
546 locale. The class is needed in the InternationalString class where a Collection of  
547 LocalizedString instances are kept. Each LocalizedString instance has a charset  
548 and lang attribute as well as a value attribute of type String.

#### 549 7.3.2.1 Attribute Summary

550

Attribute	Data Type	Required	Default Value	Specified By	Mutable
lang	language	No	en-us	Client	Yes
charset	string	No	UTF-8	Client	Yes
value	string	Yes		CLient	Yes

551

#### 552 7.3.2.2 Attribute lang

553 Each LocalizedString instance may have a lang attribute that specifies the  
554 language used by that LocalizedString.

#### 555 7.3.2.3 Attribute charset

556 Each LocalizedString instance may have a charset attribute that specifies the  
557 name of the character set used by that LocalizedString.

#### 558 7.3.2.4 Attribute value

559 Each LocalizedString instance must have a value attribute that specifies the  
560 string value used by that LocalizedString.

## 561 7.4 Class RegistryObject

### 562 Direct Known Subclasses:

563 [Association](#), [AuditableEvent](#), [Classification](#), [ClassificationNode](#),  
564 [ExternalIdentifier](#), [ExternalLink](#), [Organization](#), [RegistryEntry](#), [User](#),  
565 [Service](#), [ServiceBinding](#), [SpecificationLink](#)

566  
567 RegistryObject provides a common base class for almost all objects in the  
568 information model. Information model *Classes* whose instances have a unique  
569 identity are descendants of the RegistryObject *Class*.

570  
571 Note that Slot, PostalAddress, and a few other classes are not descendants of  
572 the RegistryObject Class because their instances do not have an independent  
573 existence and unique identity. They are always a part of some other Class's  
574 Instance (e.g., Organization has a PostalAddress).

#### 575 **7.4.1 Attribute Summary**

576 The following is the first of many tables that summarize the attributes of a class.  
577 The columns in the table are described as follows:

578

Column	Description
Attribute	The name of the attribute
Data Type	The data type for the attribute
Required	Specifies whether the attribute is required to be specified
Default	Specifies the default value in case the attribute is omitted
Specified By	Indicates whether the attribute is specified by the client or specified by the registry. In some cases it may be both
Mutable	Specifies whether an attribute may be changed once it has been set to a certain value

579

Attribute	Data Type	Required	Default Value	Specified By	Mutable
accessControlPolicy	UUID	No		Registry	No
description	International-String	No		Client	Yes
id	UUID	Yes		Client or registry	No
name	International-String	No		Client	Yes
objectType	LongName	Yes		Registry	No

#### 580 **7.4.2 Attribute accessControlPolicy**

581 Each RegistryObject instance may have an accessControlPolicy instance  
582 associated with it. An accessControlPolicy instance defines the *Security Model*  
583 associated with the RegistryObject in terms of "who is permitted to do what" with  
584 that RegistryObject.

#### 585 **7.4.3 Attribute description**

586 Each RegistryObject instance may have textual description in a human readable  
587 and user-friendly manner. This attribute is I18N capable and therefore of type  
588 InternationalString.

#### 589 **7.4.4 Attribute id**

590 Each RegistryObject instance must have a universally unique ID. Registry  
591 objects use the id of other RegistryObject instances for the purpose of  
592 referencing those objects.

593  
594 Note that some classes in the information model do not have a need for a unique  
595 id. Such classes do not inherit from RegistryObject class. Examples include  
596 Entity classes such as TelephoneNumber, PostalAddress, EmailAddress and  
597 PersonName.

598  
599 All classes derived from RegistryObject have an id that is a Universally Unique ID  
600 as defined by [UUID]. Such UUID based id attributes may be specified by the  
601 client. If the UUID based id is not specified, then it must be generated by the  
602 registry when a new RegistryObject instance is first submitted to the registry.

#### 603 **7.4.5 Attribute name**

604 Each RegistryObject instance may have human readable name. The name does  
605 not need to be unique with respect to other RegistryObject instances. This  
606 attribute is I18N capable and therefore of type InternationalString.

#### 607 **7.4.6 Attribute objectType**

608 Each RegistryObject instance has an objectType. The objectType for almost all  
609 objects in the information model is the name of their class. For example the  
610 objectType for a Classification is "Classification". The only exception to this rule  
611 is that the objectType for an ExtrinsicObject instance is user defined and  
612 indicates the type of repository item associated with the ExtrinsicObject.

##### 613 **7.4.6.1 Pre-defined Object Types**

614 The following table lists pre-defined object types. Note that for an ExtrinsicObject  
615 there are many types defined based on the type of repository item the  
616 ExtrinsicObject catalogs. In addition there are object types defined for all leaf  
617 sub-classes of RegistryObject.

618

619

620 These pre-defined object types are defined as a *ClassificationScheme*. While the  
621 scheme may easily be extended a *Registry* MUST support the object types listed  
622 below.

623

<b>Name</b>	<b>description</b>
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	An ExtrinsicObject of this type catalogues an <i>XML</i> document <i>Collaboration Protocol Agreement (CPA)</i> representing a

	technical agreement between two parties on how they plan to communicate with each other using a specific protocol.
CPP	An ExtrinsicObject of this type catalogues an document called <i>Collaboration Protocol Profile (CPP)</i> that provides information about a <i>Party</i> participating in a <i>Business</i> transaction. See [ebCPP] for details.
Process	An ExtrinsicObject of this type catalogues a process description document.
SoftwareComponent	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or <i>Class</i> library).
UMLModel	An ExtrinsicObject of this type catalogues a <i>UML</i> model.
XMLSchema	An ExtrinsicObject of this type catalogues an <i>XML</i> schema ( <i>DTD</i> , <i>XML</i> Schema, RELAX grammar, etc.).
RegistryPackage	A RegistryPackage object
ExternalLink	An ExternalLink object
ExternalIdentifier	An ExternalIdentifier object
Association	An Association object
ClassificationScheme	A ClassificationScheme object
Classification	A Classification object
ClassificationNode	A ClassificationNode object
AuditableEvent	An AuditableEvent object
User	A User object
Organization	An Organization object
Service	A Service object
ServiceBinding	A ServiceBinding object
SpecificationLink	A SpecificationLink object

624

625 **7.4.7 Method Summary**

626 In addition to its attributes, the RegistryObject class also defines the following  
 627 methods. These methods are used to navigate relationship links from a  
 628 RegistryObject instance to other objects.  
 629

<b>Method Summary for RegistryObject</b>	
Collection	<a href="#">getAuditTrail()</a> Gets the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.
Collection	<a href="#">getClassifications()</a> Gets the Classification that classify this object.

Collection	<a href="#">getExternalIdentifiers()</a> Gets the collection of ExternalIdentifiers associated with this object.
Collection	<a href="#">getExternalLinks()</a> Gets the ExternalLinks associated with this object.
Collection	<a href="#">getRegistryPackages()</a> Gets the RegistryPackages that this object is a member of.
Collection	<a href="#">getSlots()</a> Gets the Slots associated with this object.

630

631

## 632 7.5 Class RegistryEntry

### 633 Super Classes:

634 [RegistryObject](#)

635

### 636 Direct Known Subclasses:

637 [ClassificationScheme](#), [ExtrinsicObject](#), [RegistryPackage](#), [Service](#)

638

639 RegistryEntry is a common base *Class* for classes in the information model that  
 640 require additional metadata beyond the minimal metadata provided by  
 641 RegistryObject class. RegistryEntry is used as a base class for high level coarse  
 642 grained objects in the registry. Their life cycle typically requires more  
 643 management (e.g. may require approval, deprecation). They typically have  
 644 relatively fewer instances but serve as a root of a composition hierarchy  
 645 consisting of numerous objects that are sub-classes of RegistryObject but not  
 646 RegistryEntry.

647

648 The additional metadata is described by the attributes of the RegistryEntry class  
 649 below.

### 650 7.5.1 Attribute Summary

651

Attribute	Data Type	Required	Default Value	Specified By	Mutable
expiration	DateTime	No		Client	Yes
majorVersion	Integer	Yes	1	Registry	Yes
minorVersion	Integer	Yes	0	Registry	Yes
stability	LongName	No		Client	Yes
status	LongName	Yes		Registry	Yes
userVersion	ShortName	No		Client	Yes

652

653 Note that attributes inherited by RegistryEntry class from the RegistryObject  
 654 class are not shown in the table above.

## 655 **7.5.2 Attribute expiration**

656 Each RegistryEntry instance may have an expirationDate. This attribute defines a  
 657 time limit upon the stability indication provided by the stability attribute. Once the  
 658 expirationDate has been reached the stability attribute in effect becomes  
 659 STABILITY\_DYNAMIC implying that the repository item can change at any time  
 660 and in any manner. A null value implies that there is no expiration on stability  
 661 attribute.

## 662 **7.5.3 Attribute majorVersion**

663 Each RegistryEntry instance must have a major revision number for the current  
 664 version of the RegistryEntry instance. This number is assigned by the registry  
 665 when the object is created. This number may be updated by the registry when an  
 666 object is updated.

## 667 **7.5.4 Attribute minorVersion**

668 Each RegistryEntry instance must have a minor revision number for the current  
 669 version of the RegistryEntry instance. This number is assigned by the registry  
 670 when the object is created. This number may be updated by the registry when an  
 671 object is updated.

## 672 **7.5.5 Attribute stability**

673 Each RegistryEntry instance may have a stability indicator. The stability indicator  
 674 is provided by the submitter as an indication of the level of stability for the  
 675 repository item.

### 676 **7.5.5.1 Pre-defined RegistryEntry Stability Enumerations**

677 The following table lists pre-defined choices for RegistryEntry stability attribute.  
 678 These pre-defined stability types are defined as a *ClassificationScheme*. While  
 679 the scheme may easily be extended, a *Registry* MAY support the stability types  
 680 listed below.

681

<b>Name</b>	<b>Description</b>
Dynamic	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed arbitrarily by submitter at any time.
DynamicCompatible	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed in a backward compatible way by submitter at any time.
Static	Stability of a RegistryEntry that indicates that the content is static and will not be changed by submitter.

682



683 **7.5.6 Attribute status**

684 Each RegistryEntry instance must have a life cycle status indicator. The status is  
685 assigned by the registry.

686 **7.5.6.1 Pre-defined RegistryObject Status Types**

687 The following table lists pre-defined choices for RegistryObject status attribute.  
688 These pre-defined status types are defined as a *ClassificationScheme*.

689

Name	Description
Submitted	Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> .
Approved	Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> and has been subsequently approved.
Deprecated	Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> and has been subsequently deprecated.
Withdrawn	Status of a RegistryObject that catalogues content that has been withdrawn from the <i>Registry</i> .

690

691 **7.5.7 Attribute userVersion**

692 Each RegistryEntry instance may have a userVersion. The userVersion is similar  
693 to the majorVersion-minorVersion tuple. They both provide an indication of the  
694 version of the object. The majorVersion-minorVersion tuple is provided by the  
695 registry while userVersion provides a user specified version for the object.

696

697 **7.5.8 Method Summary**

698 ~~In addition to its attributes, the RegistryEntry class also defines the following~~  
699 ~~methods.~~

<b>Method Summary for RegistryEntry</b>	
<del>Organization</del>	<del><b>getSubmittingOrganization()</b> —— Gets the Organization instance of the organization that submitted the given RegistryEntry instance. This method returns a non-null result for every RegistryEntry. For privilege assignment, the organization returned by this method is regarded as the owner of the RegistryEntry instance.</del>
<del>Organization</del>	<del><b>getResponsibleOrganization()</b> —— Gets the Organization instance of the organization responsible for definition, approval, and/or maintenance of the repository item referenced by the given RegistryEntry instance. This method may return a null result if the submitting</del>



organization of this RegistryEntry does not identify a responsible organization or if the registration authority does not assign a responsible organization.

700

## 701 **7.6 Class Slot**

702 Slot instances provide a dynamic way to add arbitrary attributes to  
 703 RegistryObject instances. This ability to add attributes dynamically to  
 704 RegistryObject instances enables extensibility within the information model.  
 705

706 A RegistryObject may have 0 or more Slots. A slot is composed of a name, a  
 707 slotType and a collection of values.

### 708 **7.6.1 Attribute Summary**

709

Attribute	Data Type	Required	Default Value	Specified By	Mutable
name	LongName	Yes		Client	No
slotType	LongName	No		Client	No
values	Collection of LongName	Yes		Client	No

710

### 711 **7.6.2 Attribute name**

712 Each Slot instance must have a name. The name is the primary means for  
 713 identifying a Slot instance within a RegistryObject. Consequently, the name of a  
 714 Slot instance must be locally unique within the RegistryObject *Instance*.

### 715 **7.6.3 Attribute slotType**

716 Each Slot instance may have a slotType that allows different slots to be grouped  
 717 together.

### 718 **7.6.4 Attribute values**

719 A Slot instance must have a Collection of values. The collection of values may be  
 720 empty. Since a Slot represent an extensible attribute whose value may be a  
 721 collection, therefore a Slot is allowed to have a collection of values rather than a  
 722 single value.  
 723

## 724 **7.7 Class ExtrinsicObject**

725 **Super Classes:**

726 [RegistryEntry](#), [RegistryObject](#)

727

728

729 ExtrinsicObjects provide metadata that describes submitted content whose type  
 730 is not intrinsically known to the *Registry* and therefore MUST be described by  
 731 means of additional attributes (e.g., mime type).  
 732

733 Since the registry can contain arbitrary content without intrinsic knowledge about  
 734 that content, ExtrinsicObjects require special metadata attributes to provide some  
 735 knowledge about the object (e.g., mime type).  
 736

737 Examples of content described by ExtrinsicObject include *Collaboration Protocol*  
 738 *Profiles* [ebCPP], *Business Process* descriptions, and schemas.

### 739 **7.7.1 Attribute Summary**

740

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isOpaque	Boolean	No		Client	No
mimeType	LongName	No		Client	No

741

742 Note that attributes inherited from RegistryEntry and RegistryObject are not  
 743 shown in the table above.

### 744 **7.7.2 Attribute isOpaque**

745 Each ExtrinsicObject instance may have an isOpaque attribute defined. This  
 746 attribute determines whether the content catalogued by this ExtrinsicObject is  
 747 opaque to (not readable by) the *Registry*. In some situations, a *Submitting*  
 748 *Organization* may submit content that is encrypted and not even readable by the  
 749 *Registry*.

### 750 **7.7.3 Attribute mimeType**

751 Each ExtrinsicObject instance may have a mimeType attribute defined. The  
 752 mimeType provides information on the type of repository item catalogued by the  
 753 ExtrinsicObject instance.  
 754

## 755 **7.8 Class RegistryPackage**

### 756 **Super Classes:**

757 [RegistryEntry](#), [RegistryObject](#)

758

759 RegistryPackage instances allow for grouping of logically related RegistryObject  
 760 instances even if individual member objects belong to different Submitting  
 761 Organizations.

### 762 **7.8.1 Attribute Summary**

763

764 The RegistryPackage class defines no new attributes other than those that are  
 765 inherited from RegistryEntry and RegistryObject base classes. The inherited  
 766 attributes are not shown here.

## 767 7.8.2 Method Summary

768 In addition to its attributes, the RegistryPackage class also defines the following  
 769 methods.

770

Method Summary of RegistryPackage	
Collection	<a href="#">getMemberObjects()</a> Get the collection of RegistryObject instances that are members of this RegistryPackage.

771

## 772 7.9 Class ExternalIdentifier

### 773 Super Classes:

774 [RegistryObject](#)

775

776 ExternalIdentifier instances provide the additional identifier information to  
 777 RegistryObject such as DUNS number, Social Security Number, or an alias  
 778 name of the organization. The attribute *identificationScheme* is used to  
 779 reference the identification scheme (e.g., "DUNS", "Social Security #"), and the  
 780 attribute *value* contains the actual information (e.g., the DUNS number, the social  
 781 security number). Each RegistryObject may contain 0 or more ExternalIdentifier  
 782 instances.

### 783 7.9.1 Attribute Summary

784

Attribute	Data Type	Required	Default Value	Specified By	Mutable
identificationScheme	UUID	Yes		Client	Yes
registryObject	UUID	Yes		Client	No
value	ShortName	Yes		Client	Yes

785 Note that attributes inherited from the base classes of this class are not shown.

### 786 7.9.2 Attribute identificationScheme

787 Each ExternalIdentifier instance must have an identificationScheme attribute that  
 788 references a ClassificationScheme. This ClassificationScheme defines the  
 789 namespace within which an identifier is defined using the value attribute for the  
 790 RegistryObject referenced by the RegistryObject attribute.

### 791 **7.9.3 Attribute registryObject**

792 Each ExternalIdentifier instance must have a RegistryObject attribute that  
793 references the parent RegistryObject for which this is an ExternalIdentifier.

### 794 **7.9.4 Attribute value**

795 Each ExternalIdentifier instance must have a value attribute that provides the  
796 identifier value for this ExternalIdentifier (e.g., the actual social security number).

## 797 **7.10 Class ExternalLink**

### 798 **Super Classes:**

799 [RegistryObject](#)

800

801 ExternalLinks use URIs to associate content in the *Registry* with content that may  
802 reside outside the *Registry*. For example, an organization submitting a *DTD*  
803 could use an ExternalLink to associate the *DTD* with the organization's home  
804 page.

### 805 **7.10.1 Attribute Summary**

806

Attribute	Data Type	Required	Default Value	Specified By	Mutable
externalURI	URI	Yes		Client	Yes

807

### 808 **7.10.2 Attribute externalURI**

809 Each ExternalLink instance must have an externalURI attribute defined. The  
810 externalURI attribute provides a URI to the external resource pointed to by this  
811 ExternalLink instance. If the URI is a URL then a registry must validate the URL  
812 to be resolvable at the time of submission before accepting an ExternalLink  
813 submission to the registry.

### 814 **7.10.3 Method Summary**

815 In addition to its attributes, the ExternalLink class also defines the following  
816 methods.

817

#### Method Summary of ExternalLink

Collection	<a href="#">getLinkedObjects</a> () Gets the collection of RegistryObjects that are linked by this ExternalLink to content outside the registry.
------------	---

818

## 819 **8 Registry Audit Trail**

820 This section describes the information model *Elements* that support the audit trail  
 821 capability of the *Registry*. Several *Classes* in this section are *Entity Classes* that  
 822 are used as wrappers to model a set of related attributes. They are analogous to  
 823 the “struct” construct in the C programming language.

824

825 The `getAuditTrail()` method of a `RegistryObject` returns an ordered `Collection` of  
 826 `AuditableEvents`. These `AuditableEvents` constitute the audit trail for the  
 827 `RegistryObject`. `AuditableEvents` include a timestamp for the *Event*. Each  
 828 `AuditableEvent` has a reference to a `User` identifying the specific user that  
 829 performed an action that resulted in an `AuditableEvent`. Each `User` is affiliated  
 830 with an `Organization`, which is usually the *Submitting Organization*.

### 831 **8.1 Class AuditableEvent**

832 **Super Classes:**

833 [RegistryObject](#)

834

---

835 `AuditableEvent` instances provide a long-term record of *Events* that effect a  
 836 change in a `RegistryObject`. A `RegistryObject` is associated with an ordered  
 837 `Collection` of `AuditableEvent` instances that provide a complete audit trail for that  
 838 `RegistryObject`.

839

840 `AuditableEvents` are usually a result of a client-initiated request. `AuditableEvent`  
 841 instances are generated by the *Registry Service* to log such *Events*.

842

843 Often such *Events* effect a change in the life cycle of a `RegistryObject`. For  
 844 example a client request could Create, Update, Deprecate or Delete a  
 845 `RegistryObject`. An `AuditableEvent` is created if and only if a request creates or  
 846 alters the content or ownership of a `RegistryObject`. Read-only requests do not  
 847 generate an `AuditableEvent`. No `AuditableEvent` is generated for a  
 848 `RegistryObject` when it is classified, assigned to a `RegistryPackage` or associated  
 849 with another `RegistryObject`.

#### 850 **8.1.1 Attribute Summary**

851

Attribute	Data Type	Required	Default Value	Specified By	Mutable
eventType	LongName	Yes		Registry	No
registryObject	UUID	Yes		Registry	No
timestamp	DateTime	Yes		Registry	No
user	UUID	Yes		Registry	No

852

853 **8.1.2 Attribute eventType**

854 Each AuditableEvent must have an eventType attribute which identifies the type  
855 of event recorded by the AuditableEvent.

856 **8.1.2.1 Pre-defined Auditable Event Types**

857 The following table lists pre-defined auditable event types. These pre-defined  
858 event types are defined as a pre-defined *ClassificationScheme* with name  
859 "EventType". A Registry MUST support the event types listed below.

860

<b>Name</b>	<b>description</b>
Created	An <i>Event</i> that created a RegistryObject.
Deleted	An <i>Event</i> that deleted a RegistryObject.
Deprecated	An <i>Event</i> that deprecated a RegistryObject.
Updated	An <i>Event</i> that updated the state of a RegistryObject.
Versioned	An <i>Event</i> that versioned a RegistryObject.

861 **8.1.3 Attribute registryObject**

862 Each AuditableEvent must have a registryObject attribute that identifies the  
863 RegistryObject instance that was affected by this event.

864 **8.1.4 Attribute timestamp**

865 Each AuditableEvent must have a timestamp attribute that records the date and  
866 time that this event occurred.

867 **8.1.5 Attribute user**

868 Each AuditableEvent must have a user attribute that identifies the User that sent  
869 the request that generated this event affecting the RegistryObject instance.

870

871

872 **8.2 Class User**873 **Super Classes:**874 [RegistryObject](#)

875

876 User instances are used in an AuditableEvent to keep track of the identity of the  
877 requestor that sent the request that generated the AuditableEvent.

878 **8.2.1 Attribute Summary**

879

<b>Attribute</b>	<b>Data Type</b>	<b>Required</b>	<b>Default Value</b>	<b>Specified By</b>	<b>Mutable</b>
------------------	------------------	-----------------	----------------------	---------------------	----------------

address	PostalAddress	Yes		Client	Yes
emailAddresses	Collection of EmailAddress	Yes		Client	Yes
organization	UUID	Yes		Client	No
personName	PersonName	Yes		Client	No
telephoneNumbers	Collection of TelephoneNumber	Yes		Client	Yes
url	URI	No		Client	Yes

880

881 **8.2.2 Attribute address**

882 Each User instance must have an address attribute that provides the postal  
883 address for that user.

884 **8.2.3 Attribute emailAddresses**

885 Each User instance has an attribute emailAddresses that is a Collection of  
886 EmailAddress instances. Each EmailAddress provides an email address for that  
887 user. A User must have at least one email address.

888 **8.2.4 Attribute organization**

889 Each User instance must have an organization attribute that references the  
890 Organization instance for the organization that the user is affiliated with.

891 **8.2.5 Attribute personName**

892 Each User instance must have a personName attribute that provides the human  
893 name for that user.

894 **8.2.6 Attribute telephoneNumbers**

895 Each User instance must have a telephoneNumbers attribute that contains the  
896 Collection of TelephoneNumber instances for each telephone number defined for  
897 that user. A User must have at least one telephone number.

898 **8.2.7 Attribute url**

899 Each User instance may have a url attribute that provides the URL address for the web  
900 page associated with that user.

901 **8.3 Class Organization**902 **Super Classes:**903 [RegistryObject](#)

904

---

905 Organization instances provide information on organizations such as a  
906 *Submitting Organization*. Each Organization *Instance* may have a reference to a  
907 parent Organization.

908 **8.3.1 Attribute Summary**

909

Attribute	Data Type	Required	Default Value	Specified By	Mutable
address	PostalAddress	Yes		Client	Yes
parent	UUID	No		Client	Yes
primaryContact	UUID	Yes		Client	No
telephoneNumbers	Collection of TelephoneNumber	Yes		Client	Yes

910

911 **8.3.2 Attribute address**

912 Each Organization instance must have an address attribute that provides the  
913 postal address for that organization.

914 **8.3.3 Attribute parent**

915 Each Organization instance may have a parent attribute that references the  
916 parent Organization instance, if any, for that organization.

917 **8.3.4 Attribute primaryContact**

918 Each Organization instance must have a primaryContact attribute that references  
919 the User instance for the user that is the primary contact for that organization.

920 **8.3.5 Attribute telephoneNumbers**

921 Each Organization instance must have a telephoneNumbers attribute that  
922 contains the Collection of TelephoneNumber instances for each telephone  
923 number defined for that organization. An Organization must have at least one  
924 telephone number.

925 **8.4 Class PostalAddress**

926 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal  
927 address.

928 **8.4.1 Attribute Summary**

929

Attribute	Data Type	Required	Default Value	Specified By	Mutable
city	ShortName	No		Client	Yes
country	ShortName	No		Client	Yes
postalCode	ShortName	No		Client	Yes
state	ShortName	No		Client	Yes
street	ShortName	No		Client	Yes
streetNumber	String32	No		Client	Yes



930

931 **8.4.2 Attribute city**

932 Each PostalAddress may have a city attribute identifying the city for that address.

933 **8.4.3 Attribute country**934 Each PostalAddress may have a country attribute identifying the country for that  
935 address.936 **8.4.4 Attribute postalCode**937 Each PostalAddress may have a postalCode attribute identifying the postal code  
938 (e.g., zip code) for that address.939 **8.4.5 Attribute state**940 Each PostalAddress may have a state attribute identifying the state, province or  
941 region for that address.942 **8.4.6 Attribute street**943 Each PostalAddress may have a street attribute identifying the street name for  
944 that address.945 **8.4.7 Attribute streetNumber**946 Each PostalAddress may have a streetNumber attribute identifying the street  
947 number (e.g., 65) for the street address.948 **8.4.8 Method Summary**949 In addition to its attributes, the PostalAddress class also defines the following  
950 methods.

951

Method Summary of ExternalLink	
Collection	<a href="#">getSlots()</a> Gets the collection of Slots for this object. Each PostalAddress may have multiple Slot instances where a Slot is a dynamically defined attribute. The use of Slots allows the client to extend PostalAddress class by defining additional dynamic attributes using slots to handle locale specific needs.

952

953 **8.5 Class TelephoneNumber**954 A simple reusable *Entity Class* that defines attributes of a telephone number.955 **8.5.1 Attribute Summary**

956

Attribute	Data Type	Required	Default Value	Specified By	Mutable
areaCode	String4	No		Client	Yes
countryCode	String4	No		Client	Yes
extension	String8	No		Client	Yes
number	String16	No		Client	Yes
phoneType	String32	No		Client	Yes
url	URI	No		Client	Yes

957

958 **8.5.2 Attribute areaCode**

959 Each TelephoneNumber instance may have an areaCode attribute that provides  
960 the area code for that telephone number.

961 **8.5.3 Attribute countryCode**

962 Each TelephoneNumber instance may have an countryCode attribute that  
963 provides the country code for that telephone number.

964 **8.5.4 Attribute extension**

965 Each TelephoneNumber instance may have an extension attribute that provides  
966 the extension number, if any, for that telephone number.

967 **8.5.5 Attribute number**

968 Each TelephoneNumber instance may have a number attribute that provides the  
969 local number (without area code, country code and extension) for that telephone  
970 number.

971 **8.5.6 Attribute phoneType**

972 Each TelephoneNumber instance may have phoneType attribute that provides  
973 the type for the TelephoneNumber. Some examples of phoneType are "home",  
974 "office".

975 **8.6 Class EmailAddress**

976 A simple reusable *Entity Class* that defines attributes of an email address.

977 **8.6.1 Attribute Summary**

Attribute	Data Type	Required	Default Value	Specified By	Mutable
address	ShortName	Yes		Client	Yes
type	String32	No		Client	Yes

978 **8.6.2 Attribute address**

979 Each EmailAddress instance must have an address attribute that provides the  
980 actual email address.

981 **8.6.3 Attribute type**

982 Each EmailAddress instance may have a type attribute that provides the type for  
983 that email address. This is an arbitrary value. Examples include “home”, “work”  
984 etc.

985 **8.7 Class PersonName**

986 A simple *Entity Class* for a person’s name.

987 **8.7.1 Attribute Summary**

988

Attribute	Data Type	Required	Default Value	Specified By	Mutable
firstName	ShortName	No		Client	Yes
lastName	ShortName	No		Client	Yes
middleName	ShortName	No		Client	Yes

989 **8.7.2 Attribute firstName**

990 Each PersonName may have a firstName attribute that is the first name of the  
991 person.

992 **8.7.3 Attribute lastName**

993 Each PersonName may have a lastName attribute that is the last name of the  
994 person.

995 **8.7.4 Attribute middleName**

996 Each PersonName may have a middleName attribute that is the middle name of the  
997 person.

998 **8.8 Class Service**

999 **Super Classes:**

1000 [RegistryEntry](#), [RegistryObject](#)

1001

1002 Service instances provide information on services, such as web services.

1003 **8.8.1 Attribute Summary**

1004 The Service class does not define any specialized attributes other than its  
1005 inherited attributes.

## 1006 8.8.2 Method Summary

1007 In addition to its attributes, the Service class also defines the following methods.  
1008

Method Summary of Service	
Collection	<a href="#">getServiceBindings()</a> Gets the collection of ServiceBinding instances defined for this Service.

## 1009 8.9 Class ServiceBinding

### 1010 Super Classes:

1011 [RegistryObject](#)

1012

1013 ServiceBinding instances are RegistryObjects that represent technical  
1014 information on a specific way to access a specific interface offered by a Service  
1015 instance. A Service has a Collection of ServiceBindings.

1016 The description attribute of ServiceBinding provides details about the relationship  
1017 between several specification links comprising the Service Binding. This  
1018 description can be useful for human understanding such that the runtime system  
1019 can be appropriately configured by the human being. There is possibility of  
1020 enforcing a structure on this description for enabling machine processing of the  
1021 Service Binding, which is however not addressed by the current document.

1022

1023

### 1024 8.9.1 Attribute Summary

1025

Attribute	Data Type	Required	Default Value	Specified By	Mutable
accessURI	URI	No		Client	Yes
targetBinding	UUID	No		Client	Yes

1026

### 1027 8.9.2 Attribute accessURI

1028 A ServiceBinding may have an accessURI attribute that defines the URI to  
1029 access that ServiceBinding. This attribute is ignored if a targetBinding attribute is  
1030 specified for the ServiceBinding. If the URI is a URL then a registry must validate  
1031 the URL to be resolvable at the time of submission before accepting a  
1032 ServiceBinding submission to the registry.

### 1033 8.9.3 Attribute targetBinding

1034 A ServiceBinding may have a targetBinding attribute defined which references  
1035 another ServiceBinding. A targetBinding may be specified when a service is  
1036 being redirected to another service. This allows the rehosting of a service by  
1037 another service provider.

## 1038 **8.9.4 Method Summary**

1039 In addition to its attributes, the ServiceBinding class also defines the following  
1040 methods.  
1041

Method Summary of ServiceBinding	
Collection	<a href="#">getSpecificationLinks()</a> Get the collection of SpecificationLink instances defined for this ServiceBinding.

1042  
1043  
1044

## 1045 **8.10 Class SpecificationLink**

### 1046 **Super Classes:**

1047 [RegistryObject](#)

1048

1049 A SpecificationLink provides the linkage between a ServiceBinding and one of its  
1050 technical specifications that describes how to use the service using the  
1051 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink  
1052 instances that describe how to access the service using a technical specification  
1053 in form of a WSDL document or a CORBA IDL document.

### 1054 **8.10.1 Attribute Summary**

1055

Attribute	Data Type	Required	Default Value	Specified By	Mutable
specificationObject	UUID	Yes		Client	Yes
usageDescription	InternationalString	No		Client	Yes
usageParameters	Collection of FreeFormText	No		Client	Yes

1056

### 1057 **8.10.2 Attribute specificationObject**

1058 A SpecificationLink instance must have a specificationObject attribute that  
1059 provides a reference to a RegistryObject instance that provides a technical  
1060 specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject  
1061 instance representing the technical specification (e.g., a WSDL document).

### 1062 **8.10.3 Attribute usageDescription**

1063 A SpecificationLink instance may have a usageDescription attribute that provides  
1064 a textual description of how to use the optional usageParameters attribute  
1065 described next. The usageDescription is of type InternationalString, thus allowing  
1066 the description to be in multiple languages.

1067 **8.10.4 Attribute usageParameters**

1068 A SpecificationLink instance may have a usageParameters attribute that provides  
1069 a collection of Strings representing the instance specific parameters needed to  
1070 use the technical specification (e.g., a WSDL document) specified by this  
1071 SpecificationLink object.  
1072

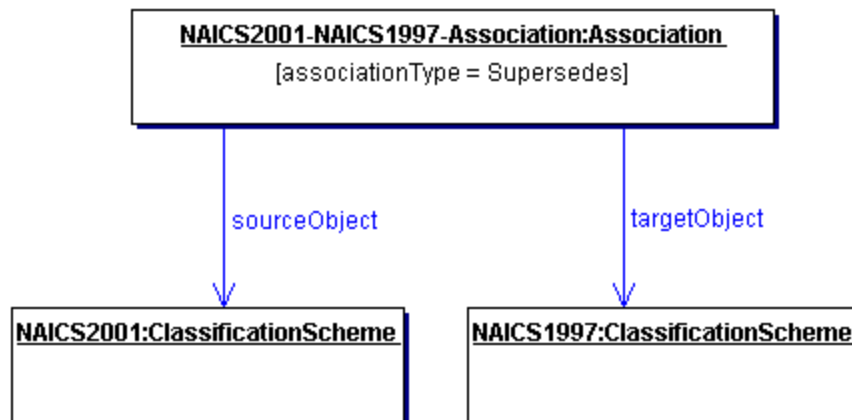
## 1072 9 Association of Registry Objects

1073 A RegistryObject instance may be *associated* with zero or more RegistryObject  
 1074 instances. The information model defines an Association class, an instance of  
 1075 which may be used to associate any two RegistryObject instances.

### 1076 9.1 Example of an Association

1077 One example of such an association is between two ClassificationScheme  
 1078 instances, where one ClassificationScheme supersedes the other  
 1079 ClassificationScheme as shown in [Figure 3](#). This may be the case when  
 1080 a new version of a ClassificationScheme is submitted.

1081 In [Figure 3](#), we see how an Association is defined between a new  
 1082 version of the NAICS ClassificationScheme and an older version of the NAICS  
 1083 ClassificationScheme.  
 1084



1085

1086

Figure 3: Example of RegistryObject Association

### 1087 9.2 Source and Target Objects

1088 An Association instance represents an association between a *source*  
 1089 RegistryObject and a *target* RegistryObject. These are referred to as  
 1090 *sourceObject* and *targetObject* for the Association instance. It is important which  
 1091 object is the *sourceObject* and which is the *targetObject* as it determines the  
 1092 directional semantics of an Association.

1093 In the example in [Figure 3](#), it is important to make the newer version of  
 1094 NAICS ClassificationScheme be the *sourceObject* and the older version of  
 1095 NAICS be the *targetObject* because the *associationType* implies that the  
 1096 *sourceObject* supersedes the *targetObject* (and not the other way around).

### 1097 9.3 Association Types

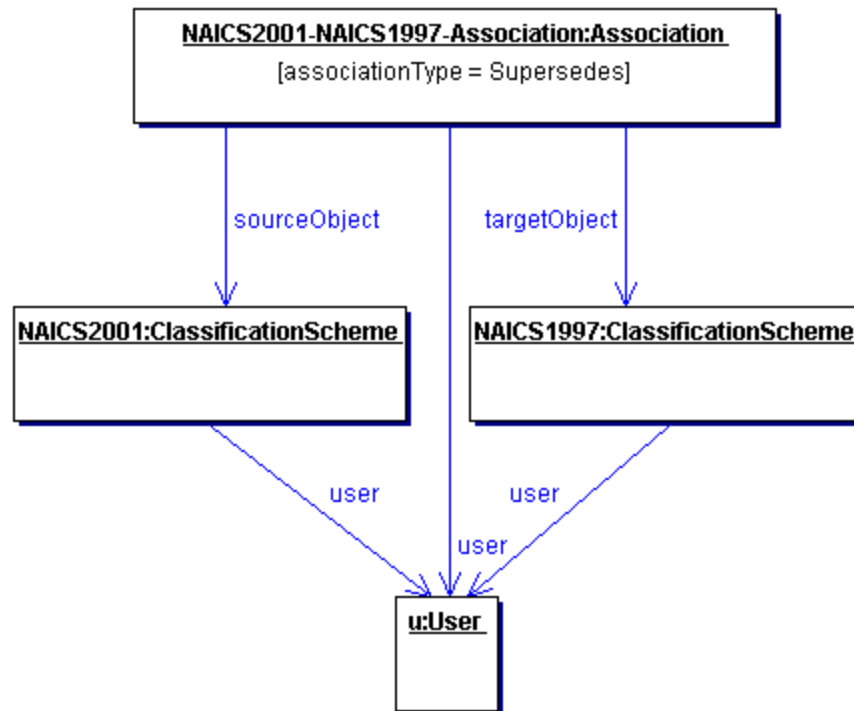
1098 Each Association must have an *associationType* attribute that identifies the type  
 1099 of that association.

## 1100 9.4 Intramural Association

1101 A common use case for the Association class is when a User “u” creates an  
 1102 Association “a” between two RegistryObjects “o1” and “o2” where association “a”  
 1103 and RegistryObjects “o1” and “o2” are objects that were created by the same  
 1104 User “u.” This is the simplest use case, where the association is between two  
 1105 objects that are owned by the same User that is defining the Association. Such  
 1106 associations are referred to as *intramural associations*.

1107 [Figure 4](#) below, extends the previous example in [Figure 3](#) for the  
 1108 intramural association case.

1109



1110

1111

Figure 4: Example of Intramural Association

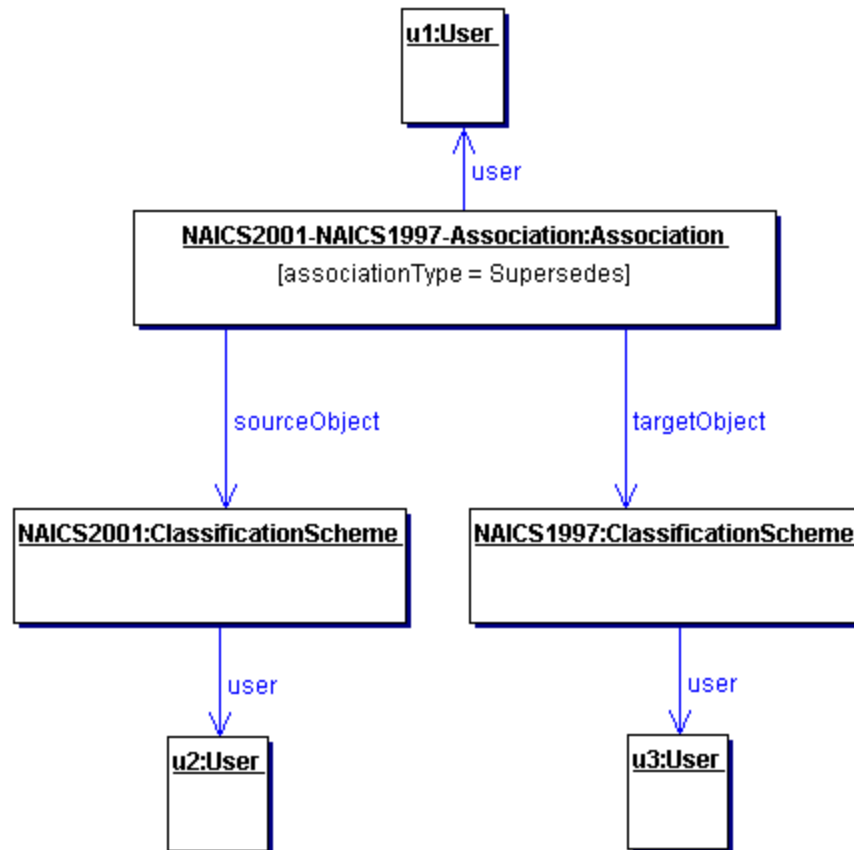
## 1112 9.5 Extramural Association

1113 The information model also allows more sophisticated use cases. For example, a  
 1114 User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”  
 1115 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2”  
 1116 are owned by User “u2” and User “u3” respectively.

1117 In this use case an Association is defined where either or both objects that are  
 1118 being associated are owned by a User different from the User defining the  
 1119 Association. Such associations are referred to as *extramural associations*. The  
 1120 Association class provides a convenience method called `isExtramural` that  
 1121 returns "true" if the Association instance is an extramural Association.



1122 [Figure 5](#) below, extends the previous example in [Figure 3](#) for the  
 1123 extramural association case. Note that it is possible for an extramural association  
 1124 to have two distinct Users rather than three distinct Users as shown in [Figure](#)  
 1125 [5](#). In such case, one of the two users owns two of the three objects  
 1126 involved (Association, sourceObject and targetObject).  
 1127



1128  
 1129

Figure 5: Example of Extramural Association

## 1130 9.6 Confirmation of an Association

1131 An association may need to be confirmed by the parties whose objects are  
 1132 involved in that Association as the sourceObject or targetObject. This section  
 1133 describes the semantics of confirmation of an association by the parties involved.

### 1134 9.6.1 Confirmation of Intramural Associations

1135 Intramural associations may be viewed as declarations of truth and do not  
 1136 require any explicit steps to confirm that Association as being true. In other  
 1137 words, intramural associations are implicitly considered confirmed.

## 1138 **9.6.2 Confirmation of Extramural Associations**

1139 Extramural associations may be thought of as a unilateral assertion that may not  
1140 be viewed as truth until it has been confirmed by the other (extramural) parties  
1141 involved (Users “u2” and “u3” in the example in section 9.5).  
1142 To confirm an extramural association, each of the extramural parties (parties that  
1143 own the source or target object but do not own the Association) must submit an  
1144 identical Association (clone Association) as the Association they are intending to  
1145 confirm using a SubmitObjectsRequest. The clone Association must have the  
1146 same id as the original Association.

## 1147 **9.6.3 Deleting an Extramural Associations**

1148 An Extramural Association is deleted like any other type of RegistryObject, using  
1149 the RemoveObjectsRequest as defined in [ebRS]. However, in some cases  
1150 deleting an extramural Association may not actually delete it but instead only  
1151 revert a confirmed association to unconfirmed state.

1152  
1153 An Association must always be deleted when deleted by the owner of that  
1154 Association, irrespective of its confirmation state. An extramural Association must  
1155 become unconfirmed by the owner of its source/target object when deleted by  
1156 the owner of its source/target object when the requestor is not the owner of the  
1157 Association itself.

## 1158 **9.7 Visibility of Unconfirmed Associations**

1159 Extramural associations require each extramural party to confirm the assertion  
1160 being made by the extramural Association before the Association is visible to  
1161 third parties that are not involved in the Association. This ensures that  
1162 unconfirmed Associations are not visible to third party registry clients.

## 1163 **9.8 Possible Confirmation States**

1164 Assume the most general case where there are three distinct User instances as  
1165 shown in [Figure 5](#) for an extramural Association. The extramural  
1166 Association needs to be confirmed by both the other (extramural) parties (Users  
1167 “u2” and “u3” in example) in order to be fully confirmed. The methods  
1168 `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the  
1169 Association class provide access to the confirmation state for both the  
1170 sourceObject and targetObject. A third convenience method called  
1171 `isConfirmed` provides a way to determine whether the Association is fully  
1172 confirmed or not. So there are the following four possibilities related to the  
1173 confirmation state of an extramural Association:

- 1174 ○ The Association is confirmed neither by the owner of the sourceObject nor  
1175 by the owner of the targetObject.
- 1176 ○ The Association is confirmed by the owner of the sourceObject but it is not  
1177 confirmed by the owner of the targetObject.
- 1178 ○ The Association is not confirmed by the owner of the sourceObject but it is  
1179 confirmed by the owner of the targetObject.

- 1180       ○ The Association is confirmed by both the owner of the sourceObject and  
 1181       the owner of the targetObject. This is the only state where the Association  
 1182       is fully confirmed.  
 1183

## 1184   **9.9 Class Association**

### 1185   **Super Classes:**

1186       [RegistryObject](#)

1187

1188

1189   Association instances are used to define many-to-many associations among  
 1190   RegistryObjects in the information model.

1191

1192   An *Instance* of the Association *Class* represents an association between two  
 1193   RegistryObjects.

### 1194   **9.9.1 Attribute Summary**

1195

Attribute	Data Type	Required	Default Value	Specified By	Mutable
associationType	LongName	Yes		Client	No
sourceObject	UUID	Yes		Client	No
targetObject	UUID	Yes		Client	No
IsConfirmedBy-SourceOwner	boolean	No	false	Registry	No
IsConfirmedBy-TargetOwner	boolean	No	false	Registry	No

1196

### 1197   **9.9.2 Attribute associationType**

1198   Each Association must have an associationType attribute that identifies the type  
 1199   of that association.

#### 1200   **9.9.2.1 Pre-defined Association Types**

1201   The following table lists pre-defined association types. These pre-defined  
 1202   association types are defined as a *Classification* scheme. While the scheme may  
 1203   easily be extended a *Registry* MUST support the association types listed below.  
 1204

name	description
RelatedTo	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source RegistryPackage object has the target RegistryObject object as a member. Reserved for use in Packaging of RegistryEntries.

<b>ExternallyLinks</b>	Defines that the source ExternalLink object externally links the target RegistryObject object. Reserved for use in associating ExternalLinks with RegistryEntries.
<b>Contains</b>	Defines that source RegistryObject contains the target RegistryObject. The details of the containment relationship are specific to the usage. For example a parts catalog may define an Engine object to have a contains relationship with a Transmission object.
<b>EquivalentTo</b>	Defines that source RegistryObject is equivalent to the target RegistryObject.
<b>Extends</b>	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
<b>Implements</b>	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
<b>InstanceOf</b>	Defines that source RegistryObject is an <i>Instance</i> of target RegistryObject.
<b>Supersedes</b>	Defines that the source RegistryObject supersedes the target RegistryObject.
<b>Uses</b>	Defines that the source RegistryObject uses the target RegistryObject in some manner.
<b>Replaces</b>	Defines that the source RegistryObject replaces the target RegistryObject in some manner.
<b>SubmitterOf</b>	Defines that the source Organization is the submitter of the target RegistryObject.
<b>ResponsibleFor</b>	Defines that the source Organization is responsible for the ongoing maintenance of the target RegistryObject.
<b>OffersService</b>	Defines that the source Organization object offers the target Service object as a service. Reserved for use in indicating that an Organization offers a Service.

1205

### 1206 **9.9.3 Attribute sourceObject**

1207 Each Association must have a sourceObject attribute that references the  
1208 RegistryObject instance that is the source of that association.

### 1209 **9.9.4 Attribute targetObject**

1210 Each Association must have a targetObject attribute that references the  
1211 RegistryObject instance that is the target of that association.

### 1212 **9.9.5 Attribute isConfirmedBySourceOwner**

1213 Each Association may have an isConfirmedBySourceOwner attribute that is set  
1214 by the registry to be true if the association has been confirmed by the owner of

1215 the sourceObject. For intramural Associations this attribute is always true. This  
 1216 attribute must be present when the object is retrieved from the registry. This  
 1217 attribute must be ignored if specified by the client when the object is submitted to  
 1218 the registry.

### 1219 **9.9.6 Attribute isConfirmedByTargetOwner**

1220 Each Association may have an isConfirmedByTargetOwner attribute that is set  
 1221 by the registry to be true if the association has been confirmed by the owner of  
 1222 the targetObject. For intramural Associations this attribute is always true. This  
 1223 attribute must be present when the object is retrieved from the registry. This  
 1224 attribute must be ignored if specified by the client when the object is submitted to  
 1225 the registry.  
 1226

Method Summary of Association	
Boolean	<p><a href="#">isConfirmed</a> ( )</p> <p>Returns true if isConfirmedBySourceOwner and isConfirmedByTargetOwner attributes are both true. For intramural Associations always return true. An association should only be visible to third parties (not involved with the Association) if isConfirmed returns true.</p>
Boolean	<p><a href="#">isExtramural</a> ( )</p> <p>Returns true if the sourceObject and/or the targetObject are owned by a User that is different from the User that created the Association.</p>

1227

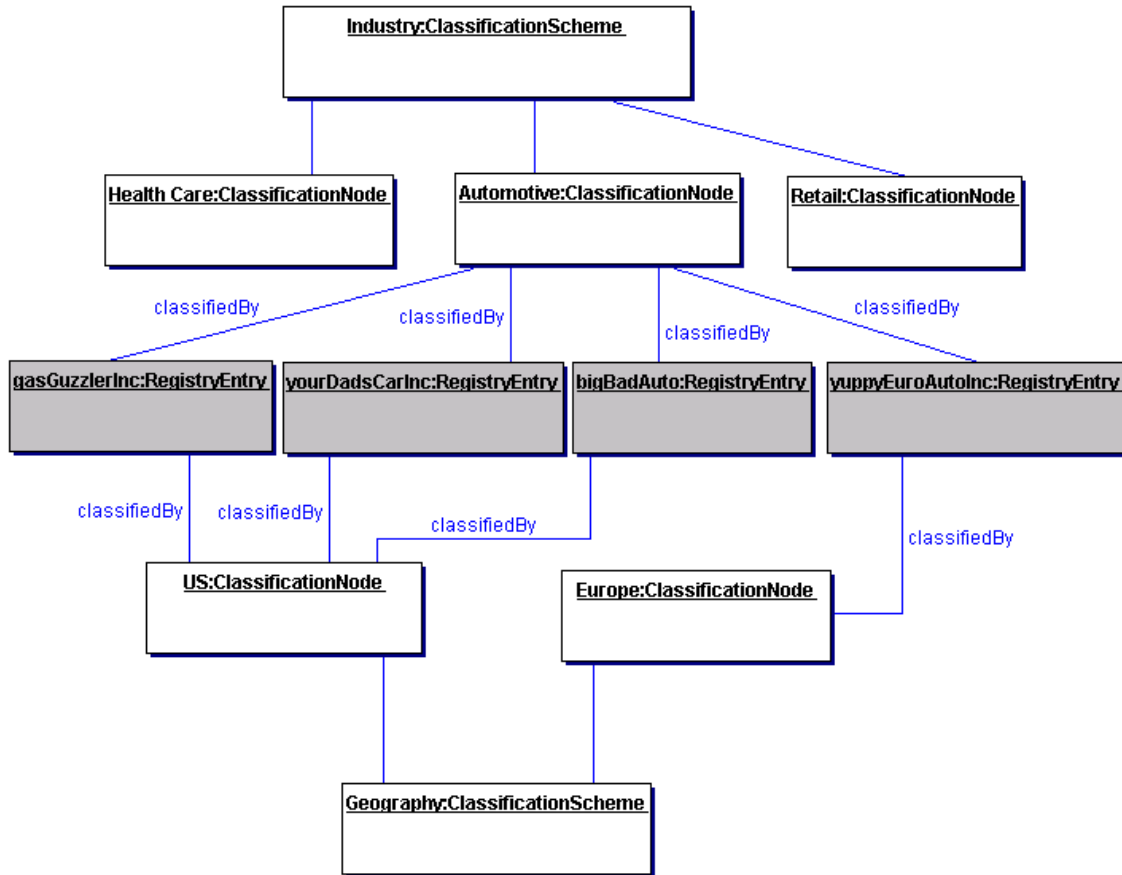
## 1228 **10 Classification of RegistryObject**

1229 This section describes the how the information model supports *Classification of*  
 1230 *RegistryObject*. It is a simplified version of the *OASIS* classification model [OAS].  
 1231

1232 A RegistryObject may be classified in many ways. For example the  
 1233 RegistryObject for the same *Collaboration Protocol Profile (CPP)* may be  
 1234 classified by its industry, by the products it sells and by its geographical location.  
 1235

1236 A general *ClassificationScheme* can be viewed as a *Classification* tree. In the  
 1237 example shown in [Figure 6](#)~~Figure-6~~, RegistryObject instances representing  
 1238 *Collaboration Protocol Profiles* are shown as shaded boxes. Each *Collaboration*  
 1239 *Protocol Profile* represents an automobile manufacturer. Each *Collaboration*  
 1240 *Protocol Profile* is classified by the ClassificationNode named "Automotive" under  
 1241 the ClassificationScheme instance with name "Industry." Furthermore, the US  
 1242 Automobile manufacturers are classified by the US ClassificationNode under the  
 1243 ClassificationScheme with name "Geography." Similarly, a European automobile  
 1244 manufacturer is classified by the "Europe" ClassificationNode under the  
 1245 ClassificationScheme with name "Geography."  
 1246

1247 The example shows how a RegistryObject may be classified by multiple  
 1248 ClassificationNode instances under multiple ClassificationScheme instances  
 1249 (e.g., Industry, Geography).  
 1250



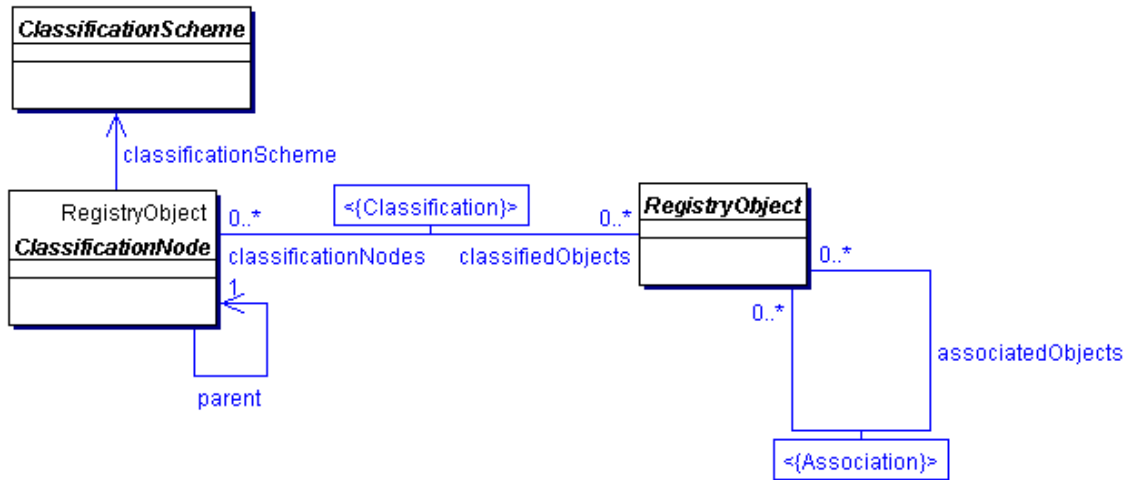
1251  
 1252

**Figure 6: Example showing a Classification Tree**

1253 [Note]It is important to point out that the dark  
 1254 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are  
 1255 not part of the Classification tree. The leaf  
 1256 nodes of the Classification tree are Health  
 1257 Care, Automotive, Retail, US and Europe. The  
 1258 dark nodes are associated with the  
 1259 Classification tree via a Classification  
 1260 Instance that is not shown in the picture

1261  
 1262  
 1263  
 1264

In order to support a general Classification scheme that can support single level as well as multi-level Classifications, the information model defines the Classes and relationships shown in [Figure 7](#).



1265

1266

Figure 7: Information Model Classification View

1267

1268

1269

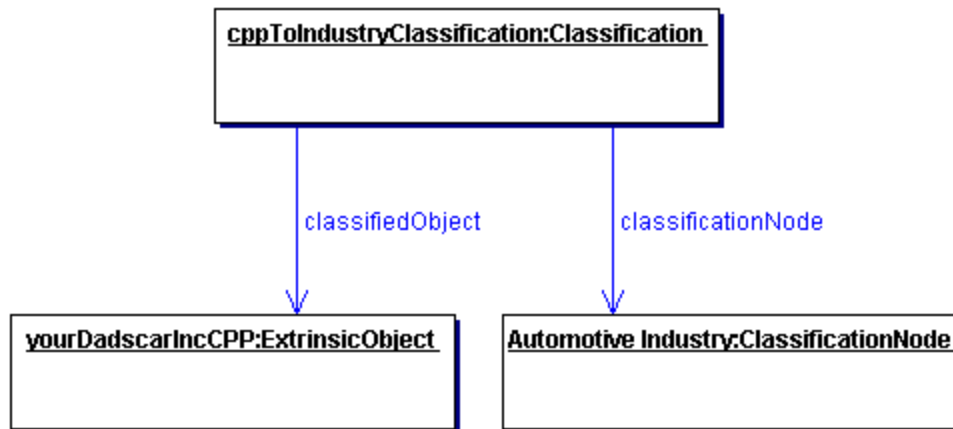
1270

1271

1272

1273

A Classification is somewhat like a specialized form of an Association. [Figure 8](#) shows an example of an ExtrinsicObject Instance for a Collaboration Protocol Profile (CPP) object that is classified by a ClassificationNode representing the Industry that it belongs to.



1274

1275

Figure 8: Classification Instance Diagram

1276

1277

1278

1279

1280

1281

## 1282 10.1 Class ClassificationScheme

### 1283 Base classes:

1284 [RegistryEntry](#), [RegistryObject](#)

1285

1286 A ClassificationScheme instance is metadata that describes a registered  
 1287 taxonomy. The taxonomy hierarchy may be defined internally to the  
 1288 Registry by instances of ClassificationNode or it may be defined externally  
 1289 to the Registry, in which case the structure and values of the taxonomy  
 1290 elements are not known to the Registry.

1291 In the first case the classification scheme is defined to be *internal* and in  
 1292 the second case the classification scheme is defined to be *external*.

1293 The ClassificationScheme class inherits attributes and methods from the  
 1294 RegistryObject and RegistryEntry classes.

1295

### 1296 10.1.1 Attribute Summary

1297

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isInternal	Boolean	Yes		Client	No
nodeType	String32	Yes		Client	No

1298 Note that attributes inherited by ClassificationScheme class from the  
 1299 RegistryEntry class are not shown.

1300

### 1301 10.1.2 Attribute isInternal

1302 When submitting a ClassificationScheme instance the Submitting Organization  
 1303 needs to declare whether the ClassificationScheme instance represents an  
 1304 internal or an external taxonomy. This allows the registry to validate the  
 1305 subsequent submissions of ClassificationNode and Classification instances in  
 1306 order to maintain the type of ClassificationScheme consistent throughout its  
 1307 lifecycle.

1308

### 1309 10.1.3 Attribute nodeType

1310 When submitting a ClassificationScheme instance the Submitting Organization  
 1311 needs to declare what is the structure of taxonomy nodes that this  
 1312 ClassificationScheme instance will represent. This attribute is an enumeration  
 1313 with the following values:

- 1314 - UniqueCode. This value says that each node of the taxonomy has  
 1315 a unique code assigned to it.
- 1316 - EmbeddedPath. This value says that a unique code assigned to  
 1317 each node of the taxonomy at the same time encodes its path. This  
 1318 is the case in the NAICS taxonomy.



1319 - NonUniqueCode. In some cases nodes are not unique, and it is  
 1320 necessary to nominate the full path in order to identify the node. For  
 1321 example, in a geography taxonomy Moscow could be under both  
 1322 Russia and the USA, where there are five cities of that name in  
 1323 different states.

1324 This enumeration might expand in the future with some new values. An example  
 1325 for possible future values for this enumeration might be NamedPathElements for  
 1326 support of Named-Level taxonomies such as Genus/Species.  
 1327

## 1328 **10.2 Class ClassificationNode**

1329 **Base classes:**

1330 [RegistryObject](#)

---

1331 ClassificationNode instances are used to define tree structures where  
 1332 each node in the tree is a ClassificationNode. Such *Classification* trees  
 1333 are constructed with ClassificationNode instances under a  
 1334 ClassificationScheme instance, and are used to define *Classification*  
 1335 schemes or ontologies.  
 1336  
 1337

### 1338 **10.2.1 Attribute Summary**

1339

Attribute	Data Type	Required	Default Value	Specified By	Mutable
parent	UUID	No		Client	No
code	ShortName	No		Client	No
path	String	No		Registry	No

1340

### 1341 **10.2.2 Attribute parent**

1342 Each ClassificationNode may have a parent attribute. The parent attribute either  
 1343 references a parent ClassificationNode or a ClassificationScheme instance in  
 1344 case of first level ClassificationNode instances.  
 1345

### 1346 **10.2.3 Attribute code**

1347 Each ClassificationNode may have a code attribute. The code attribute contains  
 1348 a code within a standard coding scheme.

### 1349 **10.2.4 Attribute path**

1350 Each ClassificationNode may have a path attribute. The path attribute must be  
 1351 present when a ClassificationNode is retrieved from the registry. The path  
 1352 attribute must be ignored when the path is specified by the client when the object

1353 is submitted to the registry. The path attribute contains the canonical path from  
 1354 the ClassificationScheme of this ClassificationNode. The path syntax is defined  
 1355 in 10.2.6.

### 1356 10.2.5 Method Summary

1357 In addition to its attributes, the ClassificationNode class also defines the following  
 1358 methods.

1359

Method Summary of ClassificationNode	
ClassificationScheme	<a href="#">getClassificationScheme</a> () Get the ClassificationScheme that this ClassificationNode belongs to.
Collection	<a href="#">getClassifiedObjects</a> () Get the collection of RegistryObjects classified by this ClassificationNode.
Integer	<a href="#">getLevelNumber</a> () Gets the level number of this ClassificationNode in the classification scheme hierarchy. This method returns a positive integer and is defined for every node instance.

1360

1361 In ~~Figure 6~~ [Figure 6](#), several instances of ClassificationNode are defined (all light  
 1362 colored boxes). A ClassificationNode has zero or one parent and zero or more  
 1363 ClassificationNodes for its immediate children. The parent of a  
 1364 ClassificationNode may be another ClassificationNode or a ClassificationScheme  
 1365 in case of first level ClassificationNodes.

1366

### 1367 10.2.6 Canonical Path Syntax

1368 The path attribute of the ClassificationNode class contains an absolute path in a  
 1369 canonical representation that uniquely identifies the path leading from the  
 1370 ClassificationScheme to that ClassificationNode.

1371 The canonical path representation is defined by the following BNF grammar:

1372

```

1373 canonicalPath ::= '/' schemeld nodePath
1374 nodePath    ::= '/' nodeCode
1375             | '/' nodeCode ( nodePath )?
  
```

1376

1377 In the above grammar, schemeld is the id attribute of the ClassificationScheme  
 1378 instance, and nodeCode is defined by NCName production as defined by  
 1379 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1380

### 1381 10.2.6.1 Example of Canonical Path Representation

1382 The following canonical path represents what the path attribute would contain for  
 1383 the ClassificationNode with code 'United States' in the sample Geography  
 1384 scheme in section 10.2.6.2.

```
1385  

  1386 /Geography-id/NorthAmerica/UnitedStates
```

### 1387 10.2.6.2 Sample Geography Scheme

1388 Note that in the following examples, the ID attributes have been chosen for ease  
 1389 of readability and are therefore not valid URN or UUID values.

```
1390  

  1391 <ClassificationScheme id='Geography-id' name="Geography"/>  

  1392  

  1393 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />  

  1394 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />  

  1395  

  1396 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />  

  1397 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />  

  1398 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />  

  1399
```

## 1400 10.3 Class Classification

### 1401 Base Classes:

1402 [RegistryObject](#)

1403  
 1404 A Classification instance classifies a RegistryObject instance by referencing a  
 1405 node defined within a particular classification scheme. An internal classification  
 1406 will always reference the node directly, by its id, while an external classification  
 1407 will reference the node indirectly by specifying a representation of its value that is  
 1408 unique within the external classification scheme.

1409  
 1410 The attributes and methods for the Classification class are intended to allow for  
 1411 representation of both internal and external classifications in order to minimize  
 1412 the need for a submission or a query to distinguish between internal and external  
 1413 classifications.

1414  
 1415 In [Figure 6](#), Classification instances are not explicitly shown but are  
 1416 implied as associations between the RegistryObject instances (shaded leaf node)  
 1417 and the associated ClassificationNode.

### 1418 10.3.1 Attribute Summary

1419

Attribute	Data Type	Required	Default Value	Specified By	Mutable
classificationScheme	UUID	for external classifications	null	Client	No
classificationNode	UUID	for internal	null	Client	No

		classifications			
classifiedObject	UUID	Yes		Client	No
nodeRepresentation	LongName	for external classifications	null	Client	No

1420 Note that attributes inherited from the base classes of this class are not shown.

1421

### 1422 **10.3.2 Attribute classificationScheme**

1423 If the Classification instance represents an external classification, then the  
1424 classificationScheme attribute is required. The classificationScheme value must  
1425 reference a ClassificationScheme instance.

1426

### 1427 **10.3.3 Attribute classificationNode**

1428 If the Classification instance represents an internal classification, then the  
1429 classificationNode attribute is required. The classificationNode value must  
1430 reference a ClassificationNode instance.

### 1431 **10.3.4 Attribute classifiedObject**

1432 For both internal and external classifications, the ClassifiedObject attribute is  
1433 required and it references the RegistryObject instance that is classified by this  
1434 Classification.

1435

### 1436 **10.3.5 Attribute nodeRepresentation**

1437 If the Classification instance represents an external classification, then the  
1438 nodeRepresentation attribute is required. It is a representation of a taxonomy  
1439 element from a classification scheme. It is the responsibility of the registry to  
1440 distinguish between different types of nodeRepresentation, like between the  
1441 classification scheme node code and the classification scheme node canonical  
1442 path. This allows client to transparently use different syntaxes for  
1443 nodeRepresentation.

### 1444 **10.3.6 Context Sensitive Classification**

1445 Consider the case depicted in [Figure 9](#) where a *Collaboration Protocol*  
1446 *Profile* for ACME Inc. is classified by the Japan ClassificationNode under the  
1447 *Geography Classification* scheme. In the absence of the context for this  
1448 *Classification* its meaning is ambiguous. Does it mean that ACME is located in  
1449 Japan, or does it mean that ACME ships products to Japan, or does it have some  
1450 other meaning? To address this ambiguity a Classification may optionally be  
1451 associated with another ClassificationNode (in this example named isLocatedIn)  
1452 that provides the missing context for the Classification. Another *Collaboration*  
1453 *Protocol Profile* for MyParcelService may be classified by the Japan  
1454 ClassificationNode where this Classification is associated with a different

1455 ClassificationNode (e.g., named shipsTo) to indicate a different context than the  
 1456 one used by ACME Inc.

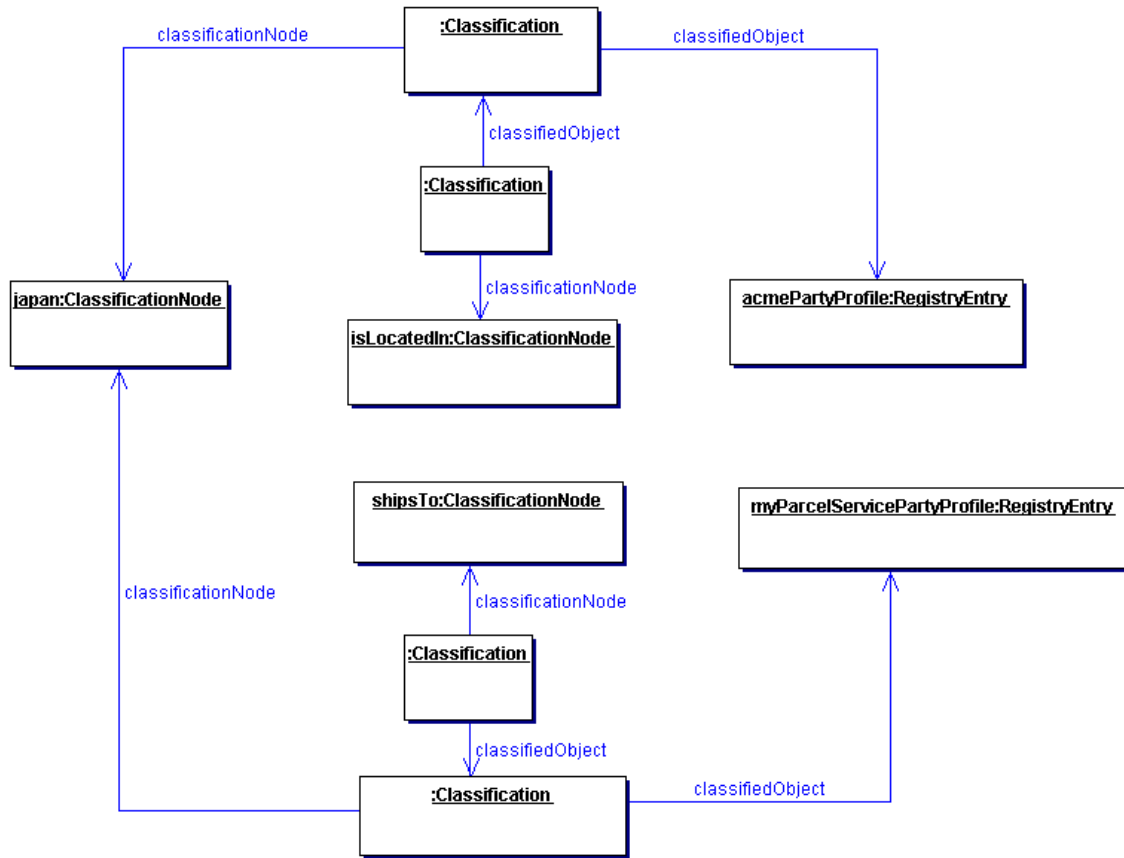


Figure 9: Context Sensitive Classification

1457  
 1458

1459  
 1460  
 1461

1462 Thus, in order to support the possibility of Classification within multiple contexts,  
 1463 a Classification is itself classified by any number of Classifications that bind the  
 1464 first Classification to ClassificationNodes that provide the missing contexts.  
 1465

1466 In summary, the generalized support for *Classification* schemes in the  
 1467 information model allows:

- 1468 ○ A RegistryObject to be classified by defining an internal Classification that  
 1469 associates it with a ClassificationNode in a *ClassificationScheme*.
- 1470 ○ A RegistryObject to be classified by defining an external Classification that  
 1471 associates it with a value in an external *ClassificationScheme*.
- 1472 ○ A RegistryObject to be classified along multiple facets by having multiple  
 1473 *Classifications* that associate it with multiple ClassificationNodes or value  
 1474 within a *ClassificationScheme*.
- 1475 ○ A *Classification* defined for a RegistryObject to be qualified by the  
 1476 contexts in which it is being classified.

1477

1478

1479 **10.3.7 Method Summary**

1480 In addition to its attributes, the Classification class also defines the following  
 1481 methods:

Return Type	Method
UUID	<p><a href="#">getClassificationScheme()</a></p> <p>For an external classification, returns the scheme identified by the classificationScheme attribute.            For an internal classification, returns the scheme identified by the same method applied to the ClassificationNode instance</p>
String	<p><a href="#">getPath()</a></p> <p>For an external classification returns a string that conforms <del>to the string structure specified for the path attribute in the ClassificationNode class.</del> <u>to the canonical path syntax as specified in 10.2.6</u>            For an internal classification, returns the value contained in the path attribute of the ClassificationNode instance identified by the classificationNode attribute.</p>
ShortName	<p><a href="#">getCode()</a></p> <p>For an external classification, returns a string that represents the declared value of the taxonomy element. It will not necessarily uniquely identify that node.            For an internal classification, returns the value of the code attribute of the ClassificationNode instance identified by the classificationNode attribute.</p>

1482

1483

1484

1485

1486

1487

1488

1489

1490

1491

1492

1493

1494 **10.4 Example of Classification Schemes**

1495 The following table lists some examples of possible *Classification* schemes  
 1496 enabled by the information model. These schemes are based on a subset of

1497 contextual concepts identified by the ebXML Business Process and Core  
 1498 Components Project Teams. This list is meant to be illustrative not prescriptive.  
 1499  
 1500

<b>Classification Scheme</b>	<b>Usage Example</b>	<b>Standard Classification Schemes</b>
Industry	Find all Parties in Automotive industry	NAICS
Process	Find a ServiceInterface that implements a Process	
Product / Services	Find a <i>Business</i> that sells a product or offers a service	UNSPSC
Locale	Find a Supplier located in Japan	ISO 3166
Temporal	Find Supplier that can ship with 24 hours	
Role	Find All Suppliers that have a <i>Role</i> of "Seller"	

1501 **Table 1: Sample Classification Schemes**

1502  
 1503  
 1504  
 1505  
 1506  
 1507  
 1508

## 1509 **11 Information Model: Security View**

1510 This section describes the aspects of the information model that relate to the  
 1511 security features of the *Registry*.

1512  
 1513 ~~Figure 10~~Figure 10 shows the view of the objects in the *Registry* from a security  
 1514 perspective. It shows object relationships as a *UML Class* diagram. It does not  
 1515 show *Class* attributes or *Class* methods that will be described in subsequent  
 1516 sections. It is meant to be illustrative not prescriptive.  
 1517

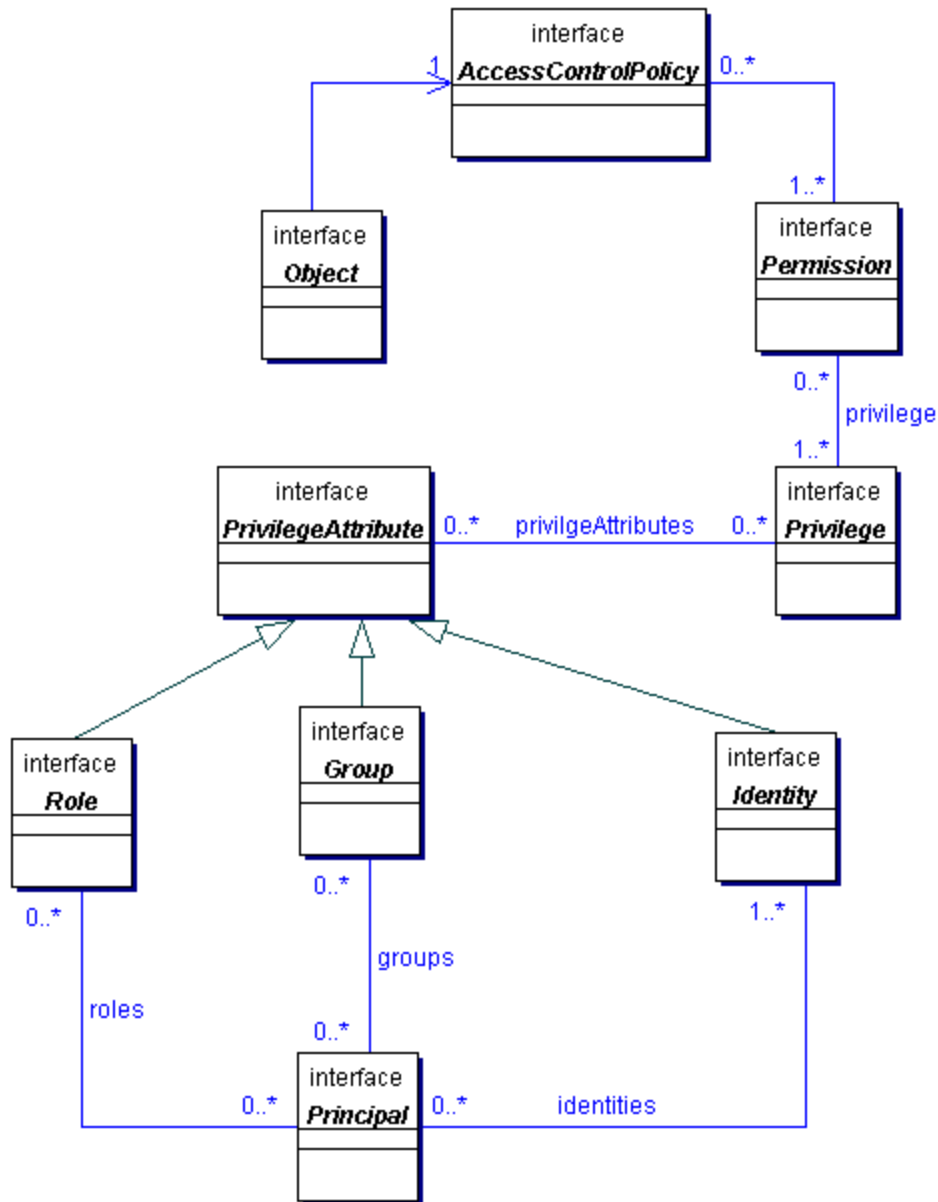


Figure 10: Information Model: Security View

1518  
 1519  
 1520  
 1521  
 1522  
 1523  
 1524  
 1525  
 1526  
 1527  
 1528

### 11.1 Class AccessControlPolicy

Every RegistryObject may be associated with exactly one AccessControlPolicy, which defines the policy rules that govern access to operations or methods performed on that RegistryObject. Such policy rules are defined as a collection of Permissions.



1529

Method Summary of AccessControlPolicy	
Collection	<a href="#">getPermissions()</a> Gets the Permissions defined for this AccessControlPolicy. Maps to attribute named <code>permissions</code> .

1530

## 1531 11.2 Class Permission

1532

1533 The Permission object is used for authorization and access control to  
 1534 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are  
 1535 defined in an AccessControlPolicy object.

1536

1537 A Permission object authorizes access to a method in a RegistryObject if the  
 1538 requesting Principal has any of the Privileges defined in the Permission.

1539 **See Also:**

1540 [Privilege](#), [AccessControlPolicy](#)

1541

Method Summary of Permission	
String	<a href="#">getMethodname()</a> Gets the method name that is accessible to a Principal with specified Privilege by this Permission. Maps to attribute named <code>methodName</code> .
Collection	<a href="#">getPrivileges()</a> Gets the Privileges associated with this Permission. Maps to attribute named <code>privileges</code> .

1542

## 1543 11.3 Class Privilege

1544

1545 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute  
 1546 can be a Group, a Role, or an Identity.

1547

1548 A requesting Principal MUST have all of the PrivilegeAttributes specified in a  
 1549 Privilege in order to gain access to a method in a protected RegistryObject.  
 1550 Permissions defined in the RegistryObject's AccessControlPolicy define the  
 1551 Privileges that can authorize access to specific methods.

1552

1553 This mechanism enables the flexibility to have object access control policies that  
 1554 are based on any combination of Roles, Identities or Groups.

1555 **See Also:**

1556 [PrivilegeAttribute](#), [Permission](#)

1557

1558

1559

Method Summary of Privilege	
Collection	<a href="#">getPrivilegeAttributes()</a> Gets the PrivilegeAttributes associated with this Privilege. Maps to attribute named <code>privilegeAttributes</code> .

1560

## 1561 11.4 Class PrivilegeAttribute

1562 **All Known Subclasses:**

1563 [Group](#), [Identity](#), [Role](#)

1564

1565 PrivilegeAttribute is a common base *Class* for all types of security attributes that  
1566 are used to grant specific access control privileges to a Principal. A Principal may  
1567 have several different types of PrivilegeAttributes. Specific combination of  
1568 PrivilegeAttributes may be defined as a Privilege object.

1569 **See Also:**

1570 [Principal](#), [Privilege](#)

## 1571 11.5 Class Role

1572 **All Superclasses:**

1573 [PrivilegeAttribute](#)

1574

### 1575 11.5.1 A security Role PrivilegeAttribute

1576 For example a hospital may have *Roles* such as Nurse, Doctor, Administrator  
1577 etc. Roles are used to grant Privileges to Principals. For example a Doctor *Role*  
1578 may be allowed to write a prescription but a Nurse *Role* may not.

## 1579 11.6 Class Group

1580 **All Superclasses:**

1581 [PrivilegeAttribute](#)

1582

### 1583 11.6.1 A security Group PrivilegeAttribute

1584 A Group is an aggregation of users that may have different Roles. For example  
1585 a hospital may have a Group defined for Nurses and Doctors that are  
1586 participating in a specific clinical trial (e.g., AspirinTrial group). Groups are used  
1587 to grant Privileges to Principals. For example the members of the AspirinTrial  
1588 group may be allowed to write a prescription for Aspirin (even though Nurse Role  
1589 as a rule may not be allowed to write prescriptions).

1590

1591

1592 **11.7 Class Identity**1593 **All Superclasses:**1594 [PrivilegeAttribute](#)

1595

1596 **11.7.1 A security Identity PrivilegeAttribute**

1597 This is typically used to identify a person, an organization, or software service.

1598 Identity attribute may be in the form of a digital certificate.

1599 **11.8 Class Principal**

1600

1601 Principal is a generic term used by the security community to include both people  
1602 and software systems. The Principal object is an entity that has a set of

1603 PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and

1604 optionally a set of role memberships, group memberships or security clearances.

1605 A principal is used to authenticate a requestor and to authorize the requested

1606 action based on the PrivilegeAttributes associated with the Principal.

1607 **See Also:**1608 PrivilegeAttributes, [Privilege](#), [Permission](#)

1609

Method Summary of Principal	
Collection	<a href="#">getGroups()</a> Gets the Groups associated with this Principal. Maps to attribute named <code>groups</code> .
Collection	<a href="#">getIdentities()</a> Gets the Identities associated with this Principal. Maps to attribute named <code>identities</code> .
Collection	<a href="#">getRoles()</a> Gets the Roles associated with this Principal. Maps to attribute named <code>roles</code> .

1610

1611

## 1611 **12 References**

- 1612 [ebGLOSS] ebXML Glossary,  
1613 [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)  
1614 [OAS] OASIS Information Model  
1615 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>  
1616 [ISO] ISO 11179 Information Model  
1617 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>  
1618  
1619 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use  
1620 in RFCs to Indicate Requirement Levels  
1621 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>  
1622 [ebRS] ebXML Registry Services Specification  
1623 <http://www.oasisopen.org/committees/regrep/documents/2.1/specs/ebRS.pdf>  
1624  
1625 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification  
1626 <http://www.ebxml.org/specrafts/>  
1627  
1628 [UUID] DCE 128 bit Universal Unique Identifier  
1629 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
1630 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>  
1631  
1632  
1633 [XPath] XML Path Language (XPath) Version 1.0  
1634 <http://www.w3.org/TR/xpath>  
1635  
1636 [NCName] Namespaces in XML 19990114  
1637 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

## 1638 **13 Disclaimer**

- 1639 The views and specification expressed in this document are those of the authors  
1640 and are not necessarily those of their employers. The authors and their  
1641 employers specifically disclaim responsibility for any problems arising from  
1642 correct or incorrect implementation or use of this design.  
1643

1643 **14 Contact Information**

1644

1645 Team Leader

1646 Name: Lisa Carnahan  
1647 Company: NIST  
1648 Street: 100 Bureau Drive STOP 8970  
1649 City, State, Postal Code: Gaithersburg, MD 20899-8970  
1650 Country: USA  
1651 Phone: (301) 975-3362  
1652 Email: lisa.carnahan@nist.gov

1653

1654 Editor

1655 Name: Sally Fuger  
1656 Company: Automotive Industry Action Group  
1657 Street: 26200 Lahser Road, Suite 200  
1658 City, State, Postal Code: Southfield, MI 48034  
1659 Country: USA  
1660 Phone: (248) 358-9744  
1661 Email: sfuger@aiag.org

1662

1663 Technical Editor

1664 Name: Farrukh S. Najmi  
1665 Company: Sun Microsystems  
1666 Street: 1 Network Dr., MS BUR02-302  
1667 City, State, Postal Code: Burlington, MA, 01803-0902  
1668 Country: USA  
1669 Phone: (781) 442-0703  
1670 Email: najmi@east.sun.com

1671

1672

**1672 Copyright Statement**

1673 OASIS takes no position regarding the validity or scope of any intellectual  
1674 property or other rights that might be claimed to pertain to the implementation or  
1675 use of the technology described in this document or the extent to which any  
1676 license under such rights might or might not be available; neither does it  
1677 represent that it has made any effort to identify any such rights. Information on  
1678 OASIS's procedures with respect to rights in OASIS specifications can be found  
1679 at the OASIS website. Copies of claims of rights made available for publication  
1680 and any assurances of licenses to be made available, or the result of an attempt  
1681 made to obtain a general license or permission for the use of such proprietary  
1682 rights by implementors or users of this specification, can be obtained from the  
1683 OASIS Executive Director.

1684  
1685 OASIS invites any interested party to bring to its attention any copyrights, patents  
1686 or patent applications, or other proprietary rights which may cover technology  
1687 that may be required to implement this specification. Please address the  
1688 information to the OASIS Executive Director.

1689  
1690 Copyright ©The Organization for the Advancement of Structured Information  
1691 Standards [OASIS] 2002. All Rights Reserved.

1692 This document and translations of it may be copied and furnished to others, and  
1693 derivative works that comment on or otherwise explain it or assist in its  
1694 implementation may be prepared, copied, published and distributed, in whole or  
1695 in part, without restriction of any kind, provided that the above copyright notice  
1696 and this paragraph are included on all such copies and derivative works.  
1697 However, this document itself may not be modified in any way, such as by  
1698 removing the copyright notice or references to OASIS, except as needed for the  
1699 purpose of developing OASIS specifications, in which case the procedures for  
1700 copyrights defined in the OASIS Intellectual Property Rights document must be  
1701 followed, or as required to translate it into languages other than English.

1702 The limited permissions granted above are perpetual and will not be revoked by  
1703 OASIS or its successors or assigns.

1704 This document and the information contained herein is provided on an "AS IS"  
1705 basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
1706 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
1707 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
1708 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
1709 PURPOSE."