



Creating A Single Global Electronic Market

1  
2  
3  
4

## **OASIS/ebXML Registry Information Model v2.02**

### **Bug Fixes To Approved OASIS Standard**

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

**May 2002**

9

## **1 Status of this Document**

Distribution of this document is unlimited.

***This version:***

<http://www.oasis-open.org/committees/regrep/documents/2.02/specs/ebRIM.pdf>

***Latest version:***

<http://www.oasis-open.org/committees/regrep/documents/2.02/specs/ebRIM.pdf>

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

## 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 Breininger, Boeing (TC Chair)  
29 Lisa Carnahan, US NIST  
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 STATUS OF THIS DOCUMENT.....</b>	<b>1</b>
56	<b>2 OASIS/EBXML REGISTRY TECHNICAL COMMITTEE.....</b>	<b>2</b>
57	2.1 CONTRIBUTORS.....	2
58	<b>3 INTRODUCTION .....</b>	<b>8</b>
59	3.1 SUMMARY OF CONTENTS OF DOCUMENT.....	8
60	3.2 GENERAL CONVENTIONS .....	8
61	3.2.1 <i>Naming Conventions</i> .....	8
62	3.3 AUDIENCE.....	9
63	3.4 RELATED DOCUMENTS .....	9
64	<b>4 DESIGN OBJECTIVES.....</b>	<b>9</b>
65	4.1 GOALS .....	9
66	<b>5 SYSTEM OVERVIEW .....</b>	<b>10</b>
67	5.1 ROLE OF EBXML <i>REGISTRY</i> .....	10
68	5.2 REGISTRY SERVICES .....	10
69	5.3 WHAT THE REGISTRY INFORMATION MODEL DOES .....	10
70	5.4 HOW THE REGISTRY INFORMATION MODEL WORKS .....	10
71	5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED .....	10
72	5.6 CONFORMANCE TO AN EBXML <i>REGISTRY</i> .....	11
73	<b>6 REGISTRY INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW.....</b>	<b>11</b>
74	6.1 REGISTRYOBJECT .....	12
75	6.2 SLOT .....	12
76	6.3 ASSOCIATION.....	12
77	6.4 EXTERNALIDENTIFIER .....	12
78	6.5 EXTERNALLINK .....	12
79	6.6 CLASSIFICATIONSCHEME.....	12
80	6.7 CLASSIFICATIONNODE.....	13
81	6.8 CLASSIFICATION .....	13
82	6.9 REGISTRYPACKAGE.....	13
83	6.10 AUDITABLEEVENT.....	13
84	6.11 USER.....	13
85	6.12 POSTALADDRESS .....	13
86	6.13 EMAILADDRESS .....	13
87	6.14 ORGANIZATION.....	14
88	6.15 SERVICE.....	14
89	6.16 SERVICEBINDING.....	14
90	6.17 SPECIFICATIONLINK .....	14

91	<b>7</b>	<b>REGISTRY INFORMATION MODEL: DETAIL VIEW</b> .....	<b>14</b>
92	7.1	ATTRIBUTE AND METHODS OF INFORMATION MODEL CLASSES .....	15
93	7.2	DATA TYPES .....	16
94	7.3	INTERNATIONALIZATION (I18N) SUPPORT.....	16
95	7.3.1	Class <i>InternationalString</i> .....	16
96	7.3.2	Class <i>LocalizedString</i> .....	17
97	7.4	CLASS REGISTRYOBJECT .....	17
98	7.4.1	Attribute <i>Summary</i> .....	18
99	7.4.2	Attribute <i>accessControlPolicy</i> .....	18
100	7.4.3	Attribute <i>description</i> .....	18
101	7.4.4	Attribute <i>id</i> .....	19
102	7.4.5	Attribute <i>name</i> .....	19
103	7.4.6	Attribute <i>objectType</i> .....	19
104	7.4.7	Method <i>Summary</i> .....	20
105	7.5	CLASS REGISTRYENTRY .....	21
106	7.5.1	Attribute <i>Summary</i> .....	21
107	7.5.2	Attribute <i>expiration</i> .....	22
108	7.5.3	Attribute <i>majorVersion</i> .....	22
109	7.5.4	Attribute <i>minorVersion</i> .....	22
110	7.5.5	Attribute <i>stability</i> .....	22
111	7.5.6	Attribute <i>status</i> .....	23
112	7.5.7	Attribute <i>userVersion</i> .....	23
113	7.6	CLASS SLOT.....	23
114	7.6.1	Attribute <i>Summary</i> .....	23
115	7.6.2	Attribute <i>name</i> .....	24
116	7.6.3	Attribute <i>slotType</i> .....	24
117	7.6.4	Attribute <i>values</i> .....	24
118	7.7	CLASS EXTRINSICOBJECT .....	24
119	7.7.1	Attribute <i>Summary</i> .....	24
120	7.7.2	Attribute <i>isOpaque</i> .....	25
121	7.7.3	Attribute <i>contentType</i> .....	25
122	7.8	CLASS REGISTRYPACKAGE.....	25
123	7.8.1	Attribute <i>Summary</i> .....	25
124	7.8.2	Method <i>Summary</i> .....	25
125	7.9	CLASS EXTERNALIDENTIFIER .....	25
126	7.9.1	Attribute <i>Summary</i> .....	26
127	7.9.2	Attribute <i>identificationScheme</i> .....	26
128	7.9.3	Attribute <i>registryObject</i> .....	26
129	7.9.4	Attribute <i>value</i> .....	26
130	7.10	CLASS EXTERNALLINK .....	26
131	7.10.1	Attribute <i>Summary</i> .....	26
132	7.10.2	Attribute <i>externalURI</i> .....	27
133	7.10.3	Method <i>Summary</i> .....	27
134	<b>8</b>	<b>REGISTRY AUDIT TRAIL</b> .....	<b>27</b>
135	8.1	CLASS AUDITABLEEVENT.....	27

136	8.1.1	Attribute Summary.....	28
137	8.1.2	Attribute eventType.....	28
138	8.1.3	Attribute registryObject.....	28
139	8.1.4	Attribute timestamp.....	28
140	8.1.5	Attribute user.....	28
141	8.2	CLASS USER.....	29
142	8.2.1	Attribute Summary.....	29
143	8.2.2	Attribute address.....	29
144	8.2.3	Attribute emailAddresses.....	29
145	8.2.4	Attribute organization.....	29
146	8.2.5	Attribute personName.....	29
147	8.2.6	Attribute telephoneNumbers.....	30
148	8.2.7	Attribute url.....	30
149	8.3	CLASS ORGANIZATION.....	30
150	8.3.1	Attribute Summary.....	30
151	8.3.2	Attribute address.....	30
152	8.3.3	Attribute parent.....	30
153	8.3.4	Attribute primaryContact.....	30
154	8.3.5	Attribute telephoneNumbers.....	30
155	8.4	CLASS POSTALADDRESS.....	31
156	8.4.1	Attribute Summary.....	31
157	8.4.2	Attribute city.....	31
158	8.4.3	Attribute country.....	31
159	8.4.4	Attribute postalCode.....	31
160	8.4.5	Attribute state.....	31
161	8.4.6	Attribute street.....	31
162	8.4.7	Attribute streetNumber.....	31
163	8.4.8	Method Summary.....	32
164	8.5	CLASS TELEPHONENUMBER.....	32
165	8.5.1	Attribute Summary.....	32
166	8.5.2	Attribute areaCode.....	32
167	8.5.3	Attribute countryCode.....	32
168	8.5.4	Attribute extension.....	32
169	8.5.5	Attribute number.....	33
170	8.5.6	Attribute phoneType.....	33
171	8.6	CLASS EMAILADDRESS.....	33
172	8.6.1	Attribute Summary.....	33
173	8.6.2	Attribute address.....	33
174	8.6.3	Attribute type.....	33
175	8.7	CLASS PERSONNAME.....	33
176	8.7.1	Attribute Summary.....	33
177	8.7.2	Attribute firstName.....	33
178	8.7.3	Attribute lastName.....	34
179	8.7.4	Attribute middleName.....	34
180	8.8	CLASS SERVICE.....	34
181	8.8.1	Attribute Summary.....	34

182	8.8.2	<i>Method Summary</i> .....	34
183	8.9	CLASS SERVICEBINDING.....	34
184	8.9.1	<i>Attribute Summary</i> .....	35
185	8.9.2	<i>Attribute accessURI</i> .....	35
186	8.9.3	<i>Attribute targetBinding</i> .....	35
187	8.9.4	<i>Method Summary</i> .....	35
188	8.10	CLASS SPECIFICATIONLINK.....	35
189	8.10.1	<i>Attribute Summary</i> .....	36
190	8.10.2	<i>Attribute specificationObject</i> .....	36
191	8.10.3	<i>Attribute usageDescription</i> .....	36
192	8.10.4	<i>Attribute usageParameters</i> .....	36
193	<b>9</b>	<b>ASSOCIATION OF REGISTRY OBJECTS</b> .....	<b>37</b>
194	9.1	EXAMPLE OF AN ASSOCIATION .....	37
195	9.2	SOURCE AND TARGET OBJECTS .....	37
196	9.3	ASSOCIATION TYPES .....	37
197	9.4	INTRAMURAL ASSOCIATION .....	38
198	9.5	EXTRAMURAL ASSOCIATION .....	38
199	9.6	CONFIRMATION OF AN ASSOCIATION.....	39
200	9.6.1	<i>Confirmation of Intramural Associations</i> .....	39
201	9.6.2	<i>Confirmation of Extramural Associations</i> .....	40
202	9.6.3	<i>Deleting an Extramural Associations</i> .....	40
203	9.7	VISIBILITY OF UNCONFIRMED ASSOCIATIONS .....	40
204	9.8	POSSIBLE CONFIRMATION STATES.....	40
205	9.9	CLASS ASSOCIATION.....	41
206	9.9.1	<i>Attribute Summary</i> .....	41
207	9.9.2	<i>Attribute associationType</i> .....	41
208	9.9.3	<i>Attribute sourceObject</i> .....	42
209	9.9.4	<i>Attribute targetObject</i> .....	42
210	9.9.5	<i>Attribute isConfirmedBySourceOwner</i> .....	42
211	9.9.6	<i>Attribute isConfirmedByTargetOwner</i> .....	43
212	<b>10</b>	<b>CLASSIFICATION OF REGISTRYOBJECT</b> .....	<b>43</b>
213	10.1	CLASS CLASSIFICATIONSCHEME.....	46
214	10.1.1	<i>Attribute Summary</i> .....	46
215	10.1.2	<i>Attribute isInternal</i> .....	46
216	10.1.3	<i>Attribute nodeType</i> .....	46
217	10.2	CLASS CLASSIFICATIONNODE.....	47
218	10.2.1	<i>Attribute Summary</i> .....	47
219	10.2.2	<i>Attribute parent</i> .....	47
220	10.2.3	<i>Attribute code</i> .....	47
221	10.2.4	<i>Attribute path</i> .....	47
222	10.2.5	<i>Method Summary</i> .....	48
223	10.2.6	<i>Canonical Path Syntax</i> .....	48
224	10.3	CLASS CLASSIFICATION.....	49
225	10.3.1	<i>Attribute Summary</i> .....	49

226	10.3.2	<i>Attribute classificationScheme</i> .....	50
227	10.3.3	<i>Attribute classificationNode</i> .....	50
228	10.3.4	<i>Attribute classifiedObject</i> .....	50
229	10.3.5	<i>Attribute nodeRepresentation</i> .....	50
230	10.3.6	<i>Context Sensitive Classification</i> .....	50
231	10.3.7	<i>Method Summary</i> .....	52
232	10.4	EXAMPLE OF <i>CLASSIFICATION</i> SCHEMES.....	52
233	<b>11</b>	<b>INFORMATION MODEL: SECURITY VIEW</b> .....	<b>53</b>
234	11.1	CLASS ACCESSCONTROLPOLICY .....	54
235	11.2	CLASS PERMISSION .....	55
236	11.3	CLASS PRIVILEGE .....	55
237	11.4	CLASS PRIVILEGEATTRIBUTE.....	56
238	11.5	CLASS ROLE .....	56
239	11.5.1	<i>A security Role PrivilegeAttribute</i> .....	56
240	11.6	CLASS GROUP.....	56
241	11.6.1	<i>A security Group PrivilegeAttribute</i> .....	56
242	11.7	CLASS IDENTITY .....	57
243	11.7.1	<i>A security Identity PrivilegeAttribute</i> .....	57
244	11.8	CLASS PRINCIPAL .....	57
245	<b>12</b>	<b>REFERENCES</b> .....	<b>58</b>
246	<b>13</b>	<b>DISCLAIMER</b> .....	<b>58</b>
247	<b>14</b>	<b>CONTACT INFORMATION</b> .....	<b>59</b>
248		<b>COPYRIGHT STATEMENT</b> .....	<b>60</b>

## 249 **Table of Figures**

250	Figure 1: Information Model High Level Public View.....	11
251	Figure 2: Information Model <i>Inheritance</i> View.....	15
252	Figure 3: Example of RegistryObject Association.....	37
253	Figure 4: Example of Intramural Association.....	38
254	Figure 5: Example of Extramural Association .....	39
255	Figure 6: Example showing a <i>Classification</i> Tree.....	44
256	Figure 7: Information Model <i>Classification</i> View.....	45
257	Figure 8: Classification <i>Instance</i> Diagram .....	45
258	Figure 9: Context Sensitive <i>Classification</i> .....	51
259	Figure 10: Information Model: Security View.....	54

## 260 **Table of Tables**

261	Table 1: Sample <i>Classification</i> Schemes.....	53
-----	--	----

262

263

## 263 **3 Introduction**

### 264 **3.1 Summary of Contents of Document**

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

266

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

### 270 **3.2 General Conventions**

271 The following conventions are used throughout this document:

272

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

275

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

279

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

282

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

286

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

288

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

292

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

#### 295 **3.2.1 Naming Conventions**

296

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

299 Capitalization styles are used in the following conventions:

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

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

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

### 311 **3.3 Audience**

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

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

### 316 **3.4 Related Documents**

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

- 319  
320 a) ebXML Registry Services Specification [ebRS] - defines the actual  
321 *Registry Services* based on this information model  
322 b) ebXML Collaboration-Protocol Profile and Agreement Specification  
323 [ebCPP] - defines how profiles can be defined for a *Party* and how two  
324 *Parties'* profiles may be used to define a *Party* agreement  
325

## 326 **4 Design Objectives**

### 327 **4.1 Goals**

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

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

## 337 **5 System Overview**

### 338 **5.1 Role of ebXML Registry**

339

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

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

### 346 **5.2 Registry Services**

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

### 351 **5.3 What the Registry Information Model Does**

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

356 The Registry information model:

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

### 360 **5.4 How the Registry Information Model Works**

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

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

### 370 **5.5 Where the Registry Information Model May Be Implemented**

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



## 391 **6.1 RegistryObject**

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

## 396 **6.2 Slot**

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

## 403 **6.3 Association**

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

## 407 **6.4 ExternalIdentifier**

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

## 411 **6.5 ExternalLink**

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

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

## 423 **6.6 ClassificationScheme**

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

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

## 432 **6.7 ClassificationNode**

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

## 439 **6.8 Classification**

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

## 446 **6.9 RegistryPackage**

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

## 449 **6.10 AuditableEvent**

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

## 453 **6.11 User**

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

## 457 **6.12 PostalAddress**

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

## 460 **6.13 EmailAddress**

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

## 463 **6.14 Organization**

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

## 467 **6.15 Service**

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

## 470 **6.16 ServiceBinding**

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

## 475 **6.17 SpecificationLink**

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

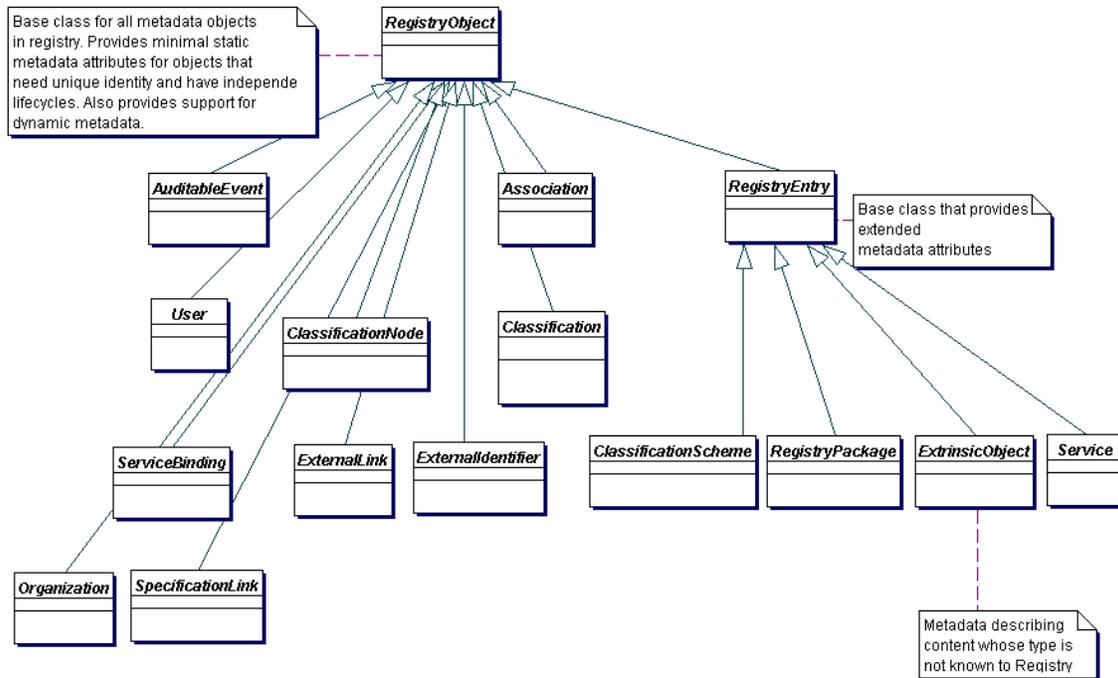
## 482 **7 Registry Information Model: Detail View**

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

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

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

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



500  
501  
502

Figure 2: Information Model *Inheritance View*

503 **7.1 Attribute and Methods of Information Model Classes**

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

508  
509 Information model classes may also have methods defined for them. These  
510 methods provide additional behavior for the class they are defined within.  
511 Methods are currently used in mapping to filter query and the SQL query  
512 capabilities defined in [ebRS].

513  
514 Since the model supports inheritance between classes, it is usually the case that  
515 a class in the model inherits attributes and methods from its base classes, in  
516 addition to defining its own specialized attributes and methods.

517

## 517 **7.2 Data Types**

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

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	

521

## 522 **7.3 Internationalization (I18N) Support**

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

526

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

### 530 **7.3.1 Class InternationalString**

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

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

#### 537 7.3.1.1 Attribute Summary

538

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

539

#### 540 7.3.1.2 Attribute localizedStrings

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

### 543 7.3.2 Class LocalizedString

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

#### 548 7.3.2.1 Attribute Summary

549

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

550

#### 551 7.3.2.2 Attribute lang

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

#### 554 7.3.2.3 Attribute charset

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

#### 557 7.3.2.4 Attribute value

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

## 560 7.4 Class RegistryObject

### 561 Direct Known Subclasses:

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

565  
566 RegistryObject provides a common base class for almost all objects in the  
567 information model. Information model *Classes* whose instances have a unique  
568 identity are descendants of the RegistryObject *Class*.

569  
570 Note that Slot, PostalAddress, and a few other classes are not descendants of  
571 the RegistryObject Class because their instances do not have an independent  
572 existence and unique identity. They are always a part of some other Class's  
573 Instance (e.g., Organization has a PostalAddress).

#### 574 **7.4.1 Attribute Summary**

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

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

578

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

#### 579 **7.4.2 Attribute accessControlPolicy**

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

#### 584 **7.4.3 Attribute description**

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

#### 588 7.4.4 Attribute id

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

592  
593 Note that some classes in the information model do not have a need for a unique  
594 id. Such classes do not inherit from RegistryObject class. Examples include  
595 Entity classes such as TelephoneNumber, PostalAddress, EmailAddress and  
596 PersonName.

597  
598 All classes derived from RegistryObject have an id that is a Universally Unique ID  
599 as defined by [UUID]. Such UUID based id attributes may be specified by the  
600 client. If the UUID based id is not specified, then it must be generated by the  
601 registry when a new RegistryObject instance is first submitted to the registry.

#### 602 7.4.5 Attribute name

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

#### 606 7.4.6 Attribute objectType

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

##### 612 7.4.6.1 Pre-defined Object Types

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

617  
618  
619 These pre-defined object types are defined as a *ClassificationScheme*. While the  
620 scheme may easily be extended a *Registry* MUST support the object types listed  
621 below.

622

Name	description
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	An ExtrinsicObject of this type catalogues an XML 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

623

624 **7.4.7 Method Summary**

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

628

Method Summary for RegistryObject	
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.

629

630

## 631 **7.5 Class RegistryEntry**

632 **Super Classes:**

633 [RegistryObject](#)

634

635 **Direct Known Subclasses:**

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

637

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

646

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

### 649 **7.5.1 Attribute Summary**

650

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

651

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

## 654 **7.5.2 Attribute expiration**

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

## 661 **7.5.3 Attribute majorVersion**

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

## 666 **7.5.4 Attribute minorVersion**

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

## 671 **7.5.5 Attribute stability**

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

### 675 **7.5.5.1 Pre-defined RegistryEntry Stability Enumerations**

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

680

<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.

681

682 **7.5.6 Attribute status**

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

685 **7.5.6.1 Pre-defined RegistryObject Status Types**

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

688

<b>Name</b>	<b>Description</b>
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> .

689

690 **7.5.7 Attribute userVersion**

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

695 **7.6 Class Slot**

696 Slot instances provide a dynamic way to add arbitrary attributes to  
697 RegistryObject instances. This ability to add attributes dynamically to  
698 RegistryObject instances enables extensibility within the information model.  
699

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

702 **7.6.1 Attribute Summary**

703

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

704

705 **7.6.2 Attribute name**

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

709 **7.6.3 Attribute slotType**

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

712 **7.6.4 Attribute values**

713 A Slot instance must have a Collection of values. The collection of values may be  
 714 empty. Since a Slot represent an extensible attribute whose value may be a  
 715 collection, therefore a Slot is allowed to have a collection of values rather than a  
 716 single value.  
 717

718 **7.7 Class ExtrinsicObject**719 **Super Classes:**720 [RegistryEntry](#), [RegistryObject](#)

721

722

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

726

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

730

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

733 **7.7.1 Attribute Summary**

734

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

735

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

### 738 **7.7.2 Attribute isOpaque**

739 Each ExtrinsicObject instance may have an isOpaque attribute defined. This  
 740 attribute determines whether the content catalogued by this ExtrinsicObject is  
 741 opaque to (not readable by) the *Registry*. In some situations, a *Submitting*  
 742 *Organization* may submit content that is encrypted and not even readable by the  
 743 *Registry*.

### 744 **7.7.3 Attribute mimeType**

745 Each ExtrinsicObject instance may have a mimeType attribute defined. The  
 746 mimeType provides information on the type of repository item catalogued by the  
 747 ExtrinsicObject instance.  
 748

## 749 **7.8 Class RegistryPackage**

### 750 **Super Classes:**

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

752

---

753 RegistryPackage instances allow for grouping of logically related RegistryObject  
 754 instances even if individual member objects belong to different Submitting  
 755 Organizations.

### 756 **7.8.1 Attribute Summary**

757

758 The RegistryPackage class defines no new attributes other than those that are  
 759 inherited from RegistryEntry and RegistryObject base classes. The inherited  
 760 attributes are not shown here.

### 761 **7.8.2 Method Summary**

762 In addition to its attributes, the RegistryPackage class also defines the following  
 763 methods.

764

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

765

## 766 **7.9 Class ExternalIdentifier**

### 767 **Super Classes:**

768 [RegistryObject](#)

769

---

770 ExternalIdentifier instances provide the additional identifier information to  
 771 RegistryObject such as DUNS number, Social Security Number, or an alias

772 name of the organization. The attribute *identificationScheme* is used to  
 773 reference the identification scheme (e.g., "DUNS", "Social Security #"), and the  
 774 attribute *value* contains the actual information (e.g., the DUNS number, the social  
 775 security number). Each RegistryObject may contain 0 or more ExternalIdentifier  
 776 instances.

### 777 **7.9.1 Attribute Summary**

778

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

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

### 780 **7.9.2 Attribute identificationScheme**

781 Each ExternalIdentifier instance must have an identificationScheme attribute that  
 782 references a ClassificationScheme. This ClassificationScheme defines the  
 783 namespace within which an identifier is defined using the value attribute for the  
 784 RegistryObject referenced by the RegistryObject attribute.

### 785 **7.9.3 Attribute registryObject**

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

### 788 **7.9.4 Attribute value**

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

## 791 **7.10 Class ExternalLink**

### 792 **Super Classes:**

793 [RegistryObject](#)

794

795 ExternalLinks use URIs to associate content in the *Registry* with content that may  
 796 reside outside the *Registry*. For example, an organization submitting a *DTD*  
 797 could use an ExternalLink to associate the *DTD* with the organization's home  
 798 page.

### 799 **7.10.1 Attribute Summary**

800

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

801

### 802 **7.10.2 Attribute externalURI**

803 Each ExternalLink instance must have an externalURI attribute defined. The  
 804 externalURI attribute provides a URI to the external resource pointed to by this  
 805 ExternalLink instance. If the URI is a URL then a registry must validate the URL  
 806 to be resolvable at the time of submission before accepting an ExternalLink  
 807 submission to the registry.

### 808 **7.10.3 Method Summary**

809 In addition to its attributes, the ExternalLink class also defines the following  
 810 methods.

811

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.

812

## 813 **8 Registry Audit Trail**

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

818

819 The getAuditTrail() method of a RegistryObject returns an ordered Collection of  
 820 AuditableEvents. These AuditableEvents constitute the audit trail for the  
 821 RegistryObject. AuditableEvents include a timestamp for the *Event*. Each  
 822 AuditableEvent has a reference to a User identifying the specific user that  
 823 performed an action that resulted in an AuditableEvent. Each User is affiliated  
 824 with an Organization, which is usually the *Submitting Organization*.

### 825 **8.1 Class AuditableEvent**

826 **Super Classes:**

827 [RegistryObject](#)

828

---

829 AuditableEvent instances provide a long-term record of *Events* that effect a  
 830 change in a RegistryObject. A RegistryObject is associated with an ordered  
 831 Collection of AuditableEvent instances that provide a complete audit trail for that  
 832 RegistryObject.

833

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

836

837 Often such *Events* effect a change in the life cycle of a RegistryObject. For  
 838 example a client request could Create, Update, Deprecate or Delete a

839 RegistryObject. An AuditableEvent is created if and only if a request creates or  
 840 alters the content or ownership of a RegistryObject. Read-only requests do not  
 841 generate an AuditableEvent. No AuditableEvent is generated for a  
 842 RegistryObject when it is classified, assigned to a RegistryPackage or associated  
 843 with another RegistryObject.

### 844 8.1.1 Attribute Summary

845

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

846

### 847 8.1.2 Attribute eventType

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

#### 850 8.1.2.1 Pre-defined Auditable Event Types

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

854

Name	description
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.

### 855 8.1.3 Attribute registryObject

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

### 858 8.1.4 Attribute timestamp

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

### 861 8.1.5 Attribute user

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

864  
865

## 866 **8.2 Class User**

### 867 **Super Classes:**

868 [RegistryObject](#)

869

---

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

### 872 **8.2.1 Attribute Summary**

873

Attribute	Data Type	Required	Default Value	Specified By	Mutable
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

874

### 875 **8.2.2 Attribute address**

876 Each User instance must have an address attribute that provides the postal  
877 address for that user.

### 878 **8.2.3 Attribute emailAddresses**

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

### 882 **8.2.4 Attribute organization**

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

### 885 **8.2.5 Attribute personName**

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

## 888 **8.2.6 Attribute telephoneNumbers**

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

## 892 **8.2.7 Attribute url**

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

## 895 **8.3 Class Organization**

### 896 **Super Classes:**

897 [RegistryObject](#)

898

---

899 Organization instances provide information on organizations such as a  
900 *Submitting Organization*. Each Organization *Instance* may have a reference to a  
901 parent Organization.

### 902 **8.3.1 Attribute Summary**

903

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

904

### 905 **8.3.2 Attribute address**

906 Each Organization instance must have an address attribute that provides the  
907 postal address for that organization.

### 908 **8.3.3 Attribute parent**

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

### 911 **8.3.4 Attribute primaryContact**

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

### 914 **8.3.5 Attribute telephoneNumbers**

915 Each Organization instance must have a telephoneNumbers attribute that  
916 contains the Collection of TelephoneNumber instances for each telephone

917 number defined for that organization. An Organization must have at least one  
918 telephone number.

## 919 **8.4 Class PostalAddress**

920 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal  
921 address.

### 922 **8.4.1 Attribute Summary**

923

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

924

### 925 **8.4.2 Attribute city**

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

### 927 **8.4.3 Attribute country**

928 Each PostalAddress may have a country attribute identifying the country for that  
929 address.

### 930 **8.4.4 Attribute postalCode**

931 Each PostalAddress may have a postalCode attribute identifying the postal code  
932 (e.g., zip code) for that address.

### 933 **8.4.5 Attribute state**

934 Each PostalAddress may have a state attribute identifying the state, province or  
935 region for that address.

### 936 **8.4.6 Attribute street**

937 Each PostalAddress may have a street attribute identifying the street name for  
938 that address.

### 939 **8.4.7 Attribute streetNumber**

940 Each PostalAddress may have a streetNumber attribute identifying the street  
941 number (e.g., 65) for the street address.

942 **8.4.8 Method Summary**

943 In addition to its attributes, the PostalAddress class also defines the following  
944 methods.

945

Method Summary of ExternalLink	
Collection	<p><a href="#">getSlots()</a></p> <p>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.</p>

946

947 **8.5 Class TelephoneNumber**

948 A simple reusable *Entity Class* that defines attributes of a telephone number.

949 **8.5.1 Attribute Summary**

950

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

951

952 **8.5.2 Attribute areaCode**

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

955 **8.5.3 Attribute countryCode**

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

958 **8.5.4 Attribute extension**

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

961 **8.5.5 Attribute number**

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

965 **8.5.6 Attribute phoneType**

966 Each TelephoneNumber instance may have phoneType attribute that provides  
 967 the type for the TelephoneNumber. Some examples of phoneType are “home”,  
 968 “office”.

969 **8.6 Class EmailAddress**

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

971 **8.6.1 Attribute Summary**

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

972 **8.6.2 Attribute address**

973 Each EmailAddress instance must have an address attribute that provides the  
 974 actual email address.

975 **8.6.3 Attribute type**

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

979 **8.7 Class PersonName**

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

981 **8.7.1 Attribute Summary**

982

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

983 **8.7.2 Attribute firstName**

984 Each PersonName may have a firstName attribute that is the first name of the  
 985 person.

986 **8.7.3 Attribute lastName**

987 Each PersonName may have a lastName attribute that is the last name of the  
988 person.

989 **8.7.4 Attribute middleName**

990 Each PersonName may have a middleName attribute that is the middle name of the  
991 person.

992 **8.8 Class Service**

993 **Super Classes:**

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

995

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

997 **8.8.1 Attribute Summary**

998 The Service class does not define any specialized attributes other than its  
999 inherited attributes.

1000 **8.8.2 Method Summary**

1001 In addition to its attributes, the Service class also defines the following methods.

1002

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

1003 **8.9 Class ServiceBinding**

1004 **Super Classes:**

1005 [RegistryObject](#)

1006

1007 ServiceBinding instances are RegistryObjects that represent technical  
1008 information on a specific way to access a specific interface offered by a Service  
1009 instance. A Service has a Collection of ServiceBindings.

1010 The description attribute of ServiceBinding provides details about the relationship  
1011 between several specification links comprising the Service Binding. This  
1012 description can be useful for human understanding such that the runtime system  
1013 can be appropriately configured by the human being. There is possibility of  
1014 enforcing a structure on this description for enabling machine processing of the  
1015 Service Binding, which is however not addressed by the current document.

1016

1017

1018 **8.9.1 Attribute Summary**

1019

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

1020

1021 **8.9.2 Attribute accessURI**

1022 A ServiceBinding may have an accessURI attribute that defines the URI to  
 1023 access that ServiceBinding. This attribute is ignored if a targetBinding attribute is  
 1024 specified for the ServiceBinding. If the URI is a URL then a registry must validate  
 1025 the URL to be resolvable at the time of submission before accepting a  
 1026 ServiceBinding submission to the registry.

1027 **8.9.3 Attribute targetBinding**

1028 A ServiceBinding may have a targetBinding attribute defined which references  
 1029 another ServiceBinding. A targetBinding may be specified when a service is  
 1030 being redirected to another service. This allows the rehosting of a service by  
 1031 another service provider.

1032 **8.9.4 Method Summary**

1033 In addition to its attributes, the ServiceBinding class also defines the following  
 1034 methods.

1035

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

1036

1037

1038

1039 **8.10 Class SpecificationLink**1040 **Super Classes:**1041 [RegistryObject](#)

1042

1043 A SpecificationLink provides the linkage between a ServiceBinding and one of its  
 1044 technical specifications that describes how to use the service using the  
 1045 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink  
 1046 instances that describe how to access the service using a technical specification  
 1047 in form of a WSDL document or a CORBA IDL document.

1048 **8.10.1 Attribute Summary**

1049

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

1050

1051 **8.10.2 Attribute specificationObject**

1052 A SpecificationLink instance must have a specificationObject attribute that  
 1053 provides a reference to a RegistryObject instance that provides a technical  
 1054 specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject  
 1055 instance representing the technical specification (e.g., a WSDL document).

1056 **8.10.3 Attribute usageDescription**

1057 A SpecificationLink instance may have a usageDescription attribute that provides  
 1058 a textual description of how to use the optional usageParameters attribute  
 1059 described next. The usageDescription is of type InternationalString, thus allowing  
 1060 the description to be in multiple languages.

1061 **8.10.4 Attribute usageParameters**

1062 A SpecificationLink instance may have a usageParameters attribute that provides  
 1063 a collection of Strings representing the instance specific parameters needed to  
 1064 use the technical specification (e.g., a WSDL document) specified by this  
 1065 SpecificationLink object.

1066

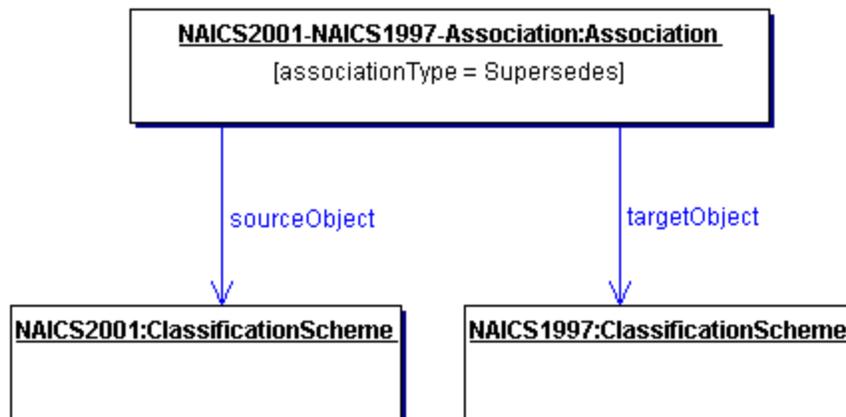
## 1066 9 Association of Registry Objects

1067 A RegistryObject instance may be *associated* with zero or more RegistryObject  
 1068 instances. The information model defines an Association class, an instance of  
 1069 which may be used to associate any two RegistryObject instances.

### 1070 9.1 Example of an Association

1071 One example of such an association is between two ClassificationScheme  
 1072 instances, where one ClassificationScheme supersedes the other  
 1073 ClassificationScheme as shown in Figure 3. This may be the case when a new  
 1074 version of a ClassificationScheme is submitted.

1075 In Figure 3, we see how an Association is defined between a new version of the  
 1076 NAICS ClassificationScheme and an older version of the NAICS  
 1077 ClassificationScheme.  
 1078



1079

1080

Figure 3: Example of RegistryObject Association

### 1081 9.2 Source and Target Objects

1082 An Association instance represents an association between a *source*  
 1083 RegistryObject and a *target* RegistryObject. These are referred to as  
 1084 *sourceObject* and *targetObject* for the Association instance. It is important which  
 1085 object is the *sourceObject* and which is the *targetObject* as it determines the  
 1086 directional semantics of an Association.

1087 In the example in Figure 3, it is important to make the newer version of NAICS  
 1088 ClassificationScheme be the *sourceObject* and the older version of NAICS be the  
 1089 *targetObject* because the *associationType* implies that the *sourceObject*  
 1090 supersedes the *targetObject* (and not the other way around).

### 1091 9.3 Association Types

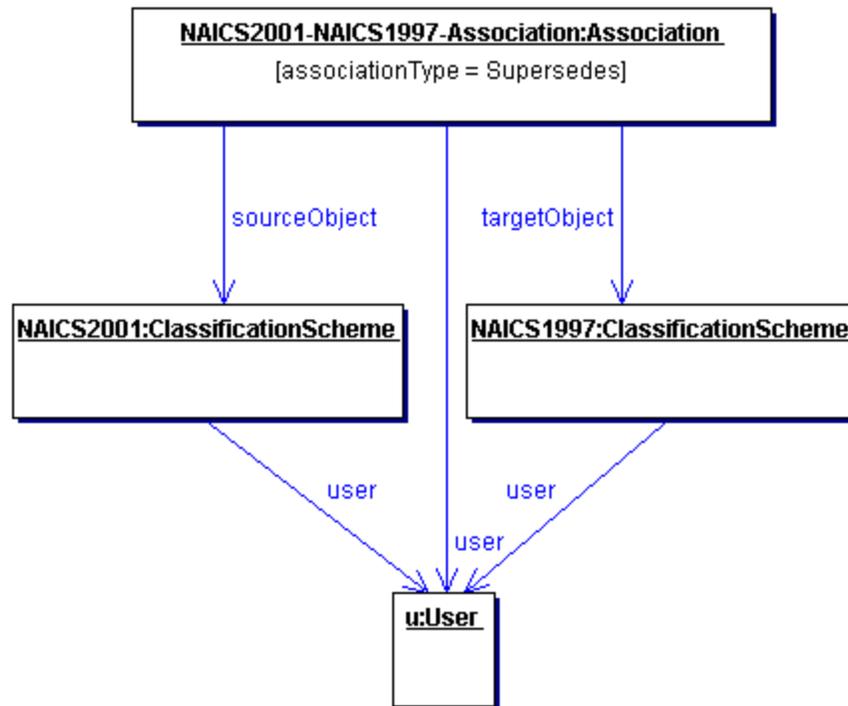
1092 Each Association must have an *associationType* attribute that identifies the type  
 1093 of that association.

## 1094 9.4 Intramural Association

1095 A common use case for the Association class is when a User “u” creates an  
 1096 Association “a” between two RegistryObjects “o1” and “o2” where association “a”  
 1097 and RegistryObjects “o1” and “o2” are objects that were created by the same  
 1098 User “u.” This is the simplest use case, where the association is between two  
 1099 objects that are owned by the same User that is defining the Association. Such  
 1100 associations are referred to as *intramural associations*.

1101 Figure 4 below, extends the previous example in Figure 3 for the intramural  
 1102 association case.

1103



1104

1105

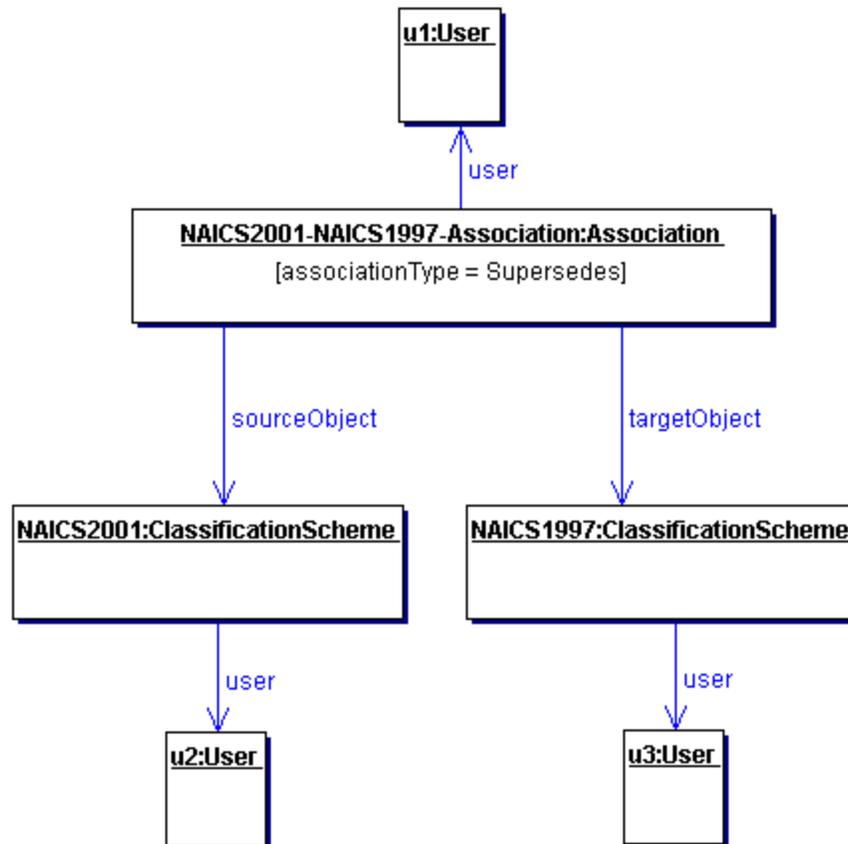
Figure 4: Example of Intramural Association

## 1106 9.5 Extramural Association

1107 The information model also allows more sophisticated use cases. For example, a  
 1108 User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”  
 1109 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2”  
 1110 are owned by User “u2” and User “u3” respectively.

1111 In this use case an Association is defined where either or both objects that are  
 1112 being associated are owned by a User different from the User defining the  
 1113 Association. Such associations are referred to as *extramural associations*. The  
 1114 Association class provides a convenience method called `isExtramural` that  
 1115 returns "true" if the Association instance is an extramural Association.

1116 Figure 5 below, extends the previous example in Figure 3 for the extramural  
 1117 association case. Note that it is possible for an extramural association to have  
 1118 two distinct Users rather than three distinct Users as shown in Figure 5. In such  
 1119 case, one of the two users owns two of the three objects involved (Association,  
 1120 sourceObject and targetObject).  
 1121



1122  
 1123

Figure 5: Example of Extramural Association

## 1124 9.6 Confirmation of an Association

1125 An association may need to be confirmed by the parties whose objects are  
 1126 involved in that Association as the sourceObject or targetObject. This section  
 1127 describes the semantics of confirmation of an association by the parties involved.

### 1128 9.6.1 Confirmation of Intramural Associations

1129 Intramural associations may be viewed as declarations of truth and do not  
 1130 require any explicit steps to confirm that Association as being true. In other  
 1131 words, intramural associations are implicitly considered confirmed.

## 1132 **9.6.2 Confirmation of Extramural Associations**

1133 Extramural associations may be thought of as a unilateral assertion that may not  
1134 be viewed as truth until it has been confirmed by the other (extramural) parties  
1135 involved (Users “u2” and “u3” in the example in section 9.5).  
1136 To confirm an extramural association, each of the extramural parties (parties that  
1137 own the source or target object but do not own the Association) must submit an  
1138 identical Association (clone Association) as the Association they are intending to  
1139 confirm using a SubmitObjectsRequest. The clone Association must have the  
1140 same id as the original Association.

## 1141 **9.6.3 Deleting an Extramural Associations**

1142 An Extramural Association is deleted like any other type of RegistryObject, using  
1143 the RemoveObjectsRequest as defined in [ebRS]. However, in some cases  
1144 deleting an extramural Association may not actually delete it but instead only  
1145 revert a confirmed association to unconfirmed state.  
1146

1147 An Association must always be deleted when deleted by the owner of that  
1148 Association, irrespective of its confirmation state. An extramural Association must  
1149 become unconfirmed by the owner of its source/target object when deleted by  
1150 the owner of its source/target object when the requestor is not the owner of the  
1151 Association itself.

## 1152 **9.7 Visibility of Unconfirmed Associations**

1153 Extramural associations require each extramural party to confirm the assertion  
1154 being made by the extramural Association before the Association is visible to  
1155 third parties that are not involved in the Association. This ensures that  
1156 unconfirmed Associations are not visible to third party registry clients.

## 1157 **9.8 Possible Confirmation States**

1158 Assume the most general case where there are three distinct User instances as  
1159 shown in Figure 5 for an extramural Association. The extramural Association  
1160 needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3”  
1161 in example) in order to be fully confirmed. The methods  
1162 `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the  
1163 Association class provide access to the confirmation state for both the  
1164 `sourceObject` and `targetObject`. A third convenience method called  
1165 `isConfirmed` provides a way to determine whether the Association is fully  
1166 confirmed or not. So there are the following four possibilities related to the  
1167 confirmation state of an extramural Association:

- 1168 ○ The Association is confirmed neither by the owner of the `sourceObject` nor  
1169 by the owner of the `targetObject`.
- 1170 ○ The Association is confirmed by the owner of the `sourceObject` but it is not  
1171 confirmed by the owner of the `targetObject`.
- 1172 ○ The Association is not confirmed by the owner of the `sourceObject` but it is  
1173 confirmed by the owner of the `targetObject`.

- 1174       ○ The Association is confirmed by both the owner of the sourceObject and  
 1175       the owner of the targetObject. This is the only state where the Association  
 1176       is fully confirmed.  
 1177

## 1178   **9.9 Class Association**

### 1179   **Super Classes:**

1180       [RegistryObject](#)

1181

1182

1183   Association instances are used to define many-to-many associations among  
 1184   RegistryObjects in the information model.

1185

1186   An *Instance* of the Association *Class* represents an association between two  
 1187   RegistryObjects.

### 1188   **9.9.1 Attribute Summary**

1189

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

1190

### 1191   **9.9.2 Attribute associationType**

1192   Each Association must have an associationType attribute that identifies the type  
 1193   of that association.

#### 1194   **9.9.2.1 Pre-defined Association Types**

1195   The following table lists pre-defined association types. These pre-defined  
 1196   association types are defined as a *Classification* scheme. While the scheme may  
 1197   easily be extended a *Registry* MUST support the association types listed below.

1198

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.

1199

1200 **9.9.3 Attribute sourceObject**

1201 Each Association must have a sourceObject attribute that references the  
1202 RegistryObject instance that is the source of that association.

1203 **9.9.4 Attribute targetObject**

1204 Each Association must have a targetObject attribute that references the  
1205 RegistryObject instance that is the target of that association.

1206 **9.9.5 Attribute isConfirmedBySourceOwner**

1207 Each Association may have an isConfirmedBySourceOwner attribute that is set  
1208 by the registry to be true if the association has been confirmed by the owner of

1209 the sourceObject. For intramural Associations this attribute is always true. This  
 1210 attribute must be present when the object is retrieved from the registry. This  
 1211 attribute must be ignored if specified by the client when the object is submitted to  
 1212 the registry.

### 1213 **9.9.6 Attribute isConfirmedByTargetOwner**

1214 Each Association may have an isConfirmedByTargetOwner attribute that is set  
 1215 by the registry to be true if the association has been confirmed by the owner of  
 1216 the targetObject. For intramural Associations this attribute is always true. This  
 1217 attribute must be present when the object is retrieved from the registry. This  
 1218 attribute must be ignored if specified by the client when the object is submitted to  
 1219 the registry.  
 1220

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>

1221

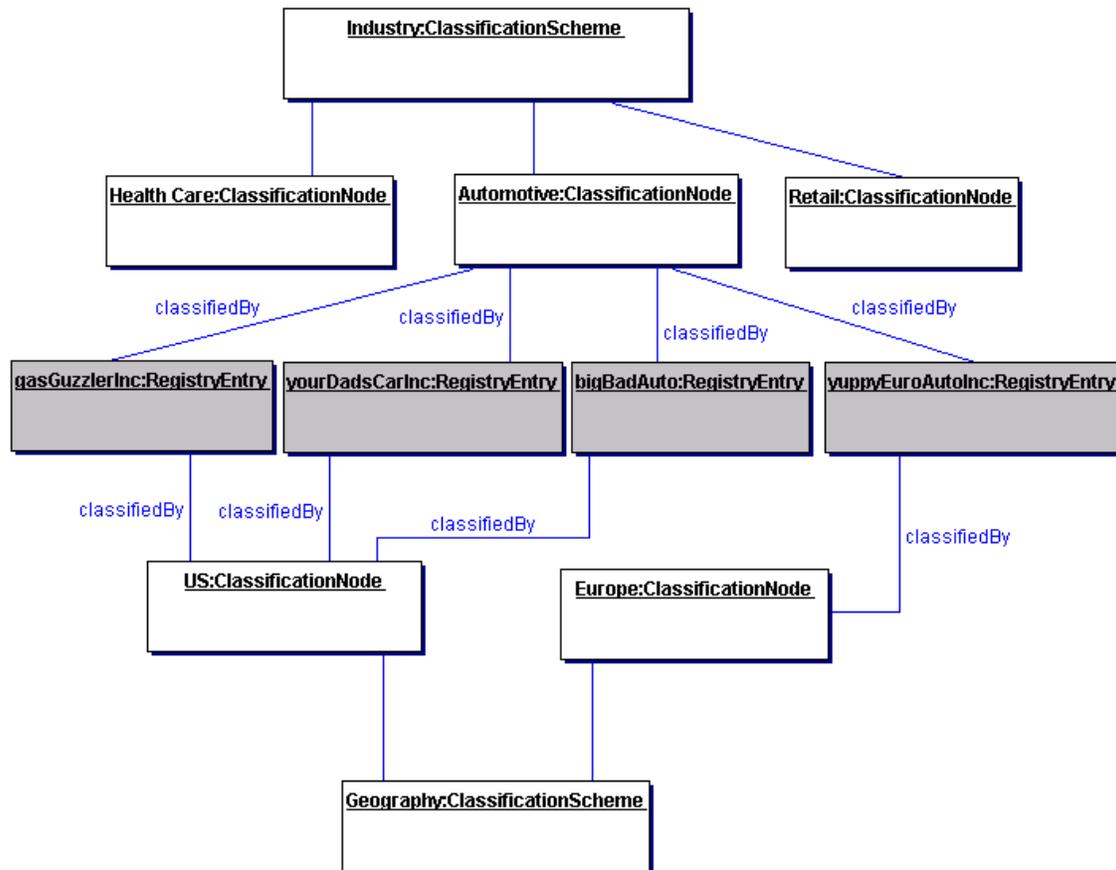
## 1222 **10 Classification of RegistryObject**

1223 This section describes the how the information model supports *Classification of*  
 1224 *RegistryObject*. It is a simplified version of the *OASIS* classification model [OAS].  
 1225

1226 A RegistryObject may be classified in many ways. For example the  
 1227 RegistryObject for the same *Collaboration Protocol Profile (CPP)* may be  
 1228 classified by its industry, by the products it sells and by its geographical location.  
 1229

1230 A general *ClassificationScheme* can be viewed as a *Classification* tree. In the  
 1231 example shown in Figure 6, RegistryObject instances representing *Collaboration*  
 1232 *Protocol Profiles* are shown as shaded boxes. Each *Collaboration Protocol*  
 1233 *Profile* represents an automobile manufacturer. Each *Collaboration Protocol*  
 1234 *Profile* is classified by the ClassificationNode named "Automotive" under the  
 1235 ClassificationScheme instance with name "Industry." Furthermore, the US  
 1236 Automobile manufacturers are classified by the US ClassificationNode under the  
 1237 ClassificationScheme with name "Geography." Similarly, a European automobile  
 1238 manufacturer is classified by the "Europe" ClassificationNode under the  
 1239 ClassificationScheme with name "Geography."  
 1240

1241 The example shows how a RegistryObject may be classified by multiple  
 1242 ClassificationNode instances under multiple ClassificationScheme instances  
 1243 (e.g., Industry, Geography).  
 1244



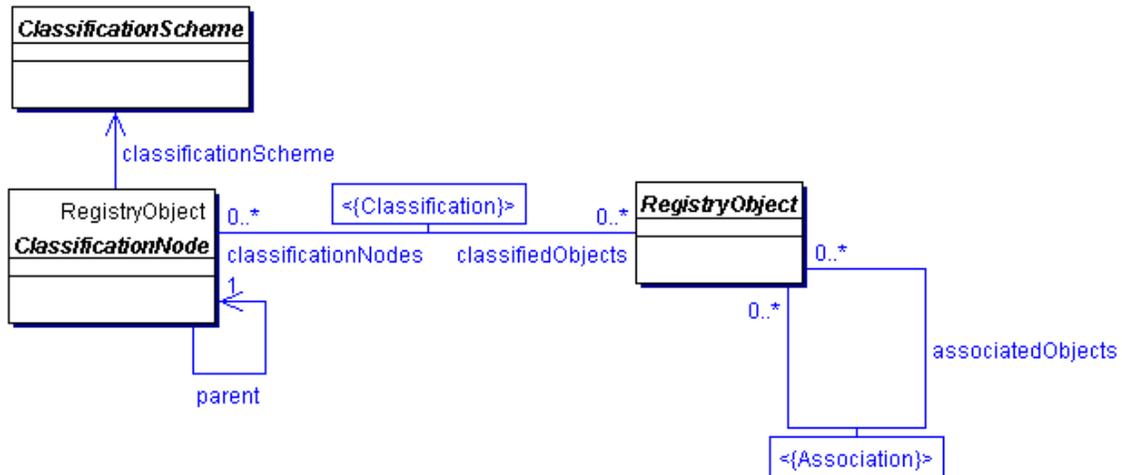
1245  
 1246

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

1247 [Note]It is important to point out that the dark  
 1248 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are  
 1249 not part of the *Classification tree*. The leaf  
 1250 nodes of the *Classification tree* are Health  
 1251 Care, Automotive, Retail, US and Europe. The  
 1252 dark nodes are associated with the  
 1253 *Classification tree* via a *Classification*  
 1254 *Instance* that is not shown in the picture

1255  
 1256  
 1257  
 1258

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.



1259

1260

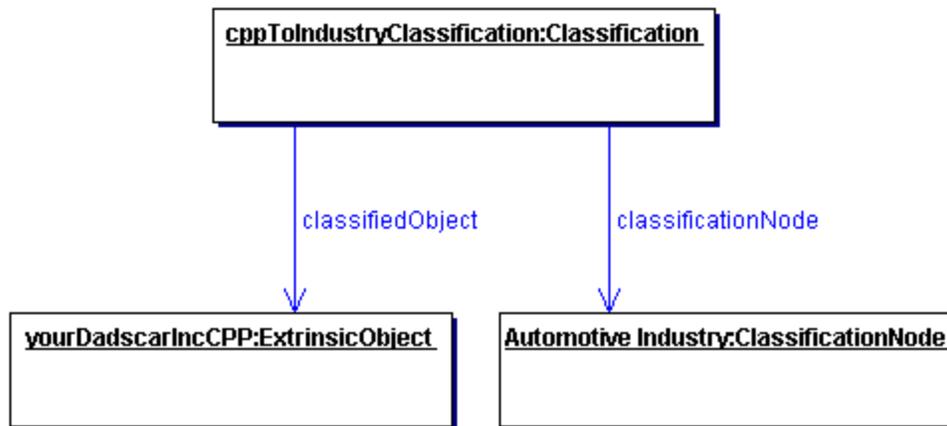
Figure 7: Information Model *Classification View*

1261

1262

1263 A Classification is somewhat like a specialized form of an Association. Figure 8  
 1264 shows an example of an ExtrinsicObject *Instance* for a *Collaboration Protocol*  
 1265 *Profile (CPP)* object that is classified by a ClassificationNode representing the  
 1266 Industry that it belongs to.

1267



1268

1269

Figure 8: Classification *Instance Diagram*

1270

1271

1272

1273

1274

1275

## 1276 **10.1 Class ClassificationScheme**

### 1277 **Base classes:**

1278 [RegistryEntry](#), [RegistryObject](#)

1279

1280 A ClassificationScheme instance is metadata that describes a registered  
1281 taxonomy. The taxonomy hierarchy may be defined internally to the  
1282 Registry by instances of ClassificationNode or it may be defined externally  
1283 to the Registry, in which case the structure and values of the taxonomy  
1284 elements are not known to the Registry.

1285 In the first case the classification scheme is defined to be *internal* and in  
1286 the second case the classification scheme is defined to be *external*.

1287 The ClassificationScheme class inherits attributes and methods from the  
1288 RegistryObject and RegistryEntry classes.

1289

### 1290 **10.1.1 Attribute Summary**

1291

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

1292 Note that attributes inherited by ClassificationScheme class from the  
1293 RegistryEntry class are not shown.

1294

### 1295 **10.1.2 Attribute isInternal**

1296 When submitting a ClassificationScheme instance the Submitting Organization  
1297 needs to declare whether the ClassificationScheme instance represents an  
1298 internal or an external taxonomy. This allows the registry to validate the  
1299 subsequent submissions of ClassificationNode and Classification instances in  
1300 order to maintain the type of ClassificationScheme consistent throughout its  
1301 lifecycle.

1302

### 1303 **10.1.3 Attribute nodeType**

1304 When submitting a ClassificationScheme instance the Submitting Organization  
1305 needs to declare what is the structure of taxonomy nodes that this  
1306 ClassificationScheme instance will represent. This attribute is an enumeration  
1307 with the following values:

- 1308 - UniqueCode. This value says that each node of the taxonomy has  
1309 a unique code assigned to it.
- 1310 - EmbeddedPath. This value says that a unique code assigned to  
1311 each node of the taxonomy at the same time encodes its path. This  
1312 is the case in the NAICS taxonomy.

1313 - NonUniqueCode. In some cases nodes are not unique, and it is  
 1314 necessary to nominate the full path in order to identify the node. For  
 1315 example, in a geography taxonomy Moscow could be under both  
 1316 Russia and the USA, where there are five cities of that name in  
 1317 different states.

1318 This enumeration might expand in the future with some new values. An example  
 1319 for possible future values for this enumeration might be NamedPathElements for  
 1320 support of Named-Level taxonomies such as Genus/Species.  
 1321

## 1322 10.2 Class ClassificationNode

1323 **Base classes:**

1324 [RegistryObject](#)

---

1325 ClassificationNode instances are used to define tree structures where  
 1326 each node in the tree is a ClassificationNode. Such *Classification* trees  
 1327 are constructed with ClassificationNode instances under a  
 1328 ClassificationScheme instance, and are used to define *Classification*  
 1329 schemes or ontologies.  
 1330  
 1331

### 1332 10.2.1 Attribute Summary

1333

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

1334

### 1335 10.2.2 Attribute parent

1336 Each ClassificationNode may have a parent attribute. The parent attribute either  
 1337 references a parent ClassificationNode or a ClassificationScheme instance in  
 1338 case of first level ClassificationNode instances.  
 1339

### 1340 10.2.3 Attribute code

1341 Each ClassificationNode may have a code attribute. The code attribute contains  
 1342 a code within a standard coding scheme.

### 1343 10.2.4 Attribute path

1344 Each ClassificationNode may have a path attribute. The path attribute must be  
 1345 present when a ClassificationNode is retrieved from the registry. The path  
 1346 attribute must be ignored when the path is specified by the client when the object

1347 is submitted to the registry. The path attribute contains the canonical path from  
 1348 the ClassificationScheme of this ClassificationNode. The path syntax is defined  
 1349 in 10.2.6.

### 1350 **10.2.5 Method Summary**

1351 In addition to its attributes, the ClassificationNode class also defines the following  
 1352 methods.

1353

<b>Method Summary of ClassificationNode</b>	
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.

1354

1355 In Figure 6, several instances of ClassificationNode are defined (all light colored  
 1356 boxes). A ClassificationNode has zero or one parent and zero or more  
 1357 ClassificationNodes for its immediate children. The parent of a  
 1358 ClassificationNode may be another ClassificationNode or a ClassificationScheme  
 1359 in case of first level ClassificationNodes.

1360

### 1361 **10.2.6 Canonical Path Syntax**

1362 The path attribute of the ClassificationNode class contains an absolute path in a  
 1363 canonical representation that uniquely identifies the path leading from the  
 1364 ClassificationScheme to that ClassificationNode.

1365 The canonical path representation is defined by the following BNF grammar:

1366

```

1367 canonicalPath ::= '/' schemeld nodePath
1368 nodePath     ::= '/' nodeCode
1369             | '/' nodeCode ( nodePath )?
  
```

1370

1371 In the above grammar, schemeld is the id attribute of the ClassificationScheme  
 1372 instance, and nodeCode is defined by NCName production as defined by  
 1373 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1374

### 1375 10.2.6.1 Example of Canonical Path Representation

1376 The following canonical path represents what the path attribute would contain for  
 1377 the ClassificationNode with code 'United States' in the sample Geography  
 1378 scheme in section 10.2.6.2.

1379  
 1380 `/Geography-id/NorthAmerica/UnitedStates`

### 1381 10.2.6.2 Sample Geography Scheme

1382 Note that in the following examples, the ID attributes have been chosen for ease  
 1383 of readability and are therefore not valid URN or UUID values.

```
1384
1385 <ClassificationScheme id='Geography-id' name="Geography"/>
1386
1387 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
1388 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
1389
1390 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
1391 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
1392 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
1393
```

## 1394 10.3 Class Classification

### 1395 Base Classes:

1396 [RegistryObject](#)

1397  
 1398 A Classification instance classifies a RegistryObject instance by referencing a  
 1399 node defined within a particular classification scheme. An internal classification  
 1400 will always reference the node directly, by its id, while an external classification  
 1401 will reference the node indirectly by specifying a representation of its value that is  
 1402 unique within the external classification scheme.

1403  
 1404 The attributes and methods for the Classification class are intended to allow for  
 1405 representation of both internal and external classifications in order to minimize  
 1406 the need for a submission or a query to distinguish between internal and external  
 1407 classifications.

1408  
 1409 In Figure 6, Classification instances are not explicitly shown but are implied as  
 1410 associations between the RegistryObject instances (shaded leaf node) and the  
 1411 associated ClassificationNode.

### 1412 10.3.1 Attribute Summary

1413

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

1414 Note that attributes inherited from the base classes of this class are not shown.  
1415

### 1416 **10.3.2 Attribute classificationScheme**

1417 If the Classification instance represents an external classification, then the  
1418 classificationScheme attribute is required. The classificationScheme value must  
1419 reference a ClassificationScheme instance.  
1420

### 1421 **10.3.3 Attribute classificationNode**

1422 If the Classification instance represents an internal classification, then the  
1423 classificationNode attribute is required. The classificationNode value must  
1424 reference a ClassificationNode instance.

### 1425 **10.3.4 Attribute classifiedObject**

1426 For both internal and external classifications, the ClassifiedObject attribute is  
1427 required and it references the RegistryObject instance that is classified by this  
1428 Classification.  
1429

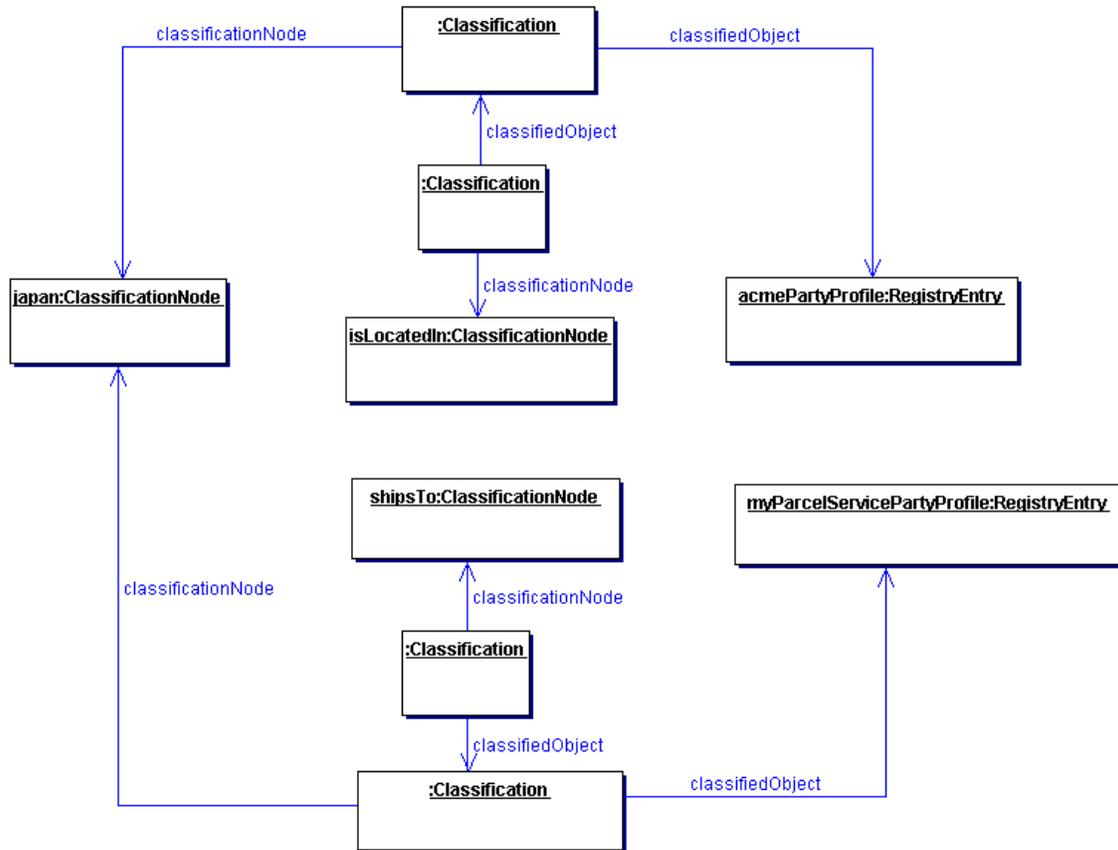
### 1430 **10.3.5 Attribute nodeRepresentation**

1431 If the Classification instance represents an external classification, then the  
1432 nodeRepresentation attribute is required. It is a representation of a taxonomy  
1433 element from a classification scheme. It is the responsibility of the registry to  
1434 distinguish between different types of nodeRepresentation, like between the  
1435 classification scheme node code and the classification scheme node canonical  
1436 path. This allows client to transparently use different syntaxes for  
1437 nodeRepresentation.

### 1438 **10.3.6 Context Sensitive Classification**

1439 Consider the case depicted in Figure 9 where a *Collaboration Protocol Profile* for  
1440 ACME Inc. is classified by the Japan ClassificationNode under the Geography  
1441 *Classification* scheme. In the absence of the context for this *Classification* its  
1442 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it  
1443 mean that ACME ships products to Japan, or does it have some other meaning?  
1444 To address this ambiguity a Classification may optionally be associated with  
1445 another ClassificationNode (in this example named isLocatedIn) that provides the  
1446 missing context for the Classification. Another *Collaboration Protocol Profile* for  
1447 MyParcelService may be classified by the Japan ClassificationNode where this

1448 Classification is associated with a different ClassificationNode (e.g., named  
 1449 shipsTo) to indicate a different context than the one used by ACME Inc.



1450  
 1451

**Figure 9: Context Sensitive Classification**

1452  
 1453  
 1454

1455 Thus, in order to support the possibility of Classification within multiple contexts,  
 1456 a Classification is itself classified by any number of Classifications that bind the  
 1457 first Classification to ClassificationNodes that provide the missing contexts.  
 1458

1459 In summary, the generalized support for *Classification* schemes in the  
 1460 information model allows:

- 1461 ○ A RegistryObject to be classified by defining an internal Classification that
- 1462 associates it with a ClassificationNode in a *ClassificationScheme*.
- 1463 ○ A RegistryObject to be classified by defining an external Classification that
- 1464 associates it with a value in an external *ClassificationScheme*.
- 1465 ○ A RegistryObject to be classified along multiple facets by having multiple
- 1466 *Classifications* that associate it with multiple ClassificationNodes or value
- 1467 within a *ClassificationScheme*.
- 1468 ○ A *Classification* defined for a RegistryObject to be qualified by the
- 1469 contexts in which it is being classified.

1470  
1471

### 1472 **10.3.7 Method Summary**

1473 In addition to its attributes, the Classification class also defines the following  
1474 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 to the canonical path syntax as specified in 10.2.6. 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>

1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486

### 1487 **10.4 Example of Classification Schemes**

1488 The following table lists some examples of possible *Classification* schemes  
1489 enabled by the information model. These schemes are based on a subset of  
1490 contextual concepts identified by the ebXML Business Process and Core  
1491 Components Project Teams. This list is meant to be illustrative not prescriptive.  
1492

1493

<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"	

1494

Table 1: Sample *Classification Schemes*

1495

1496

1497

1498

1499

1500

1501

1502

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

1503

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

1504

1505

1506

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

1507

1508

1509

1510

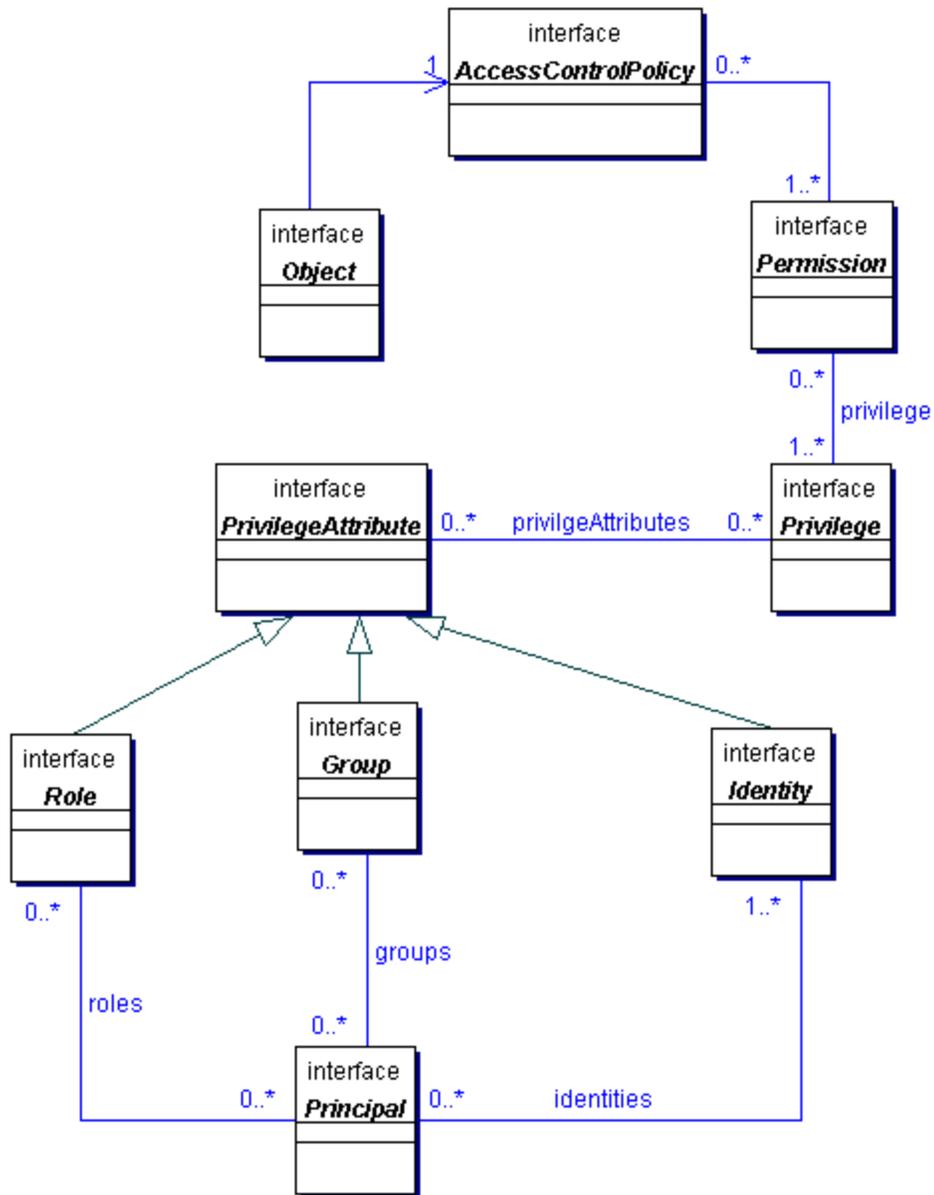


Figure 10: Information Model: Security View

1511  
 1512  
 1513  
 1514  
 1515  
 1516  
 1517  
 1518  
 1519  
 1520  
 1521

### 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.

1522

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

1523

## 1524 11.2 Class Permission

1525

1526 The Permission object is used for authorization and access control to  
1527 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are  
1528 defined in an AccessControlPolicy object.

1529

1530 A Permission object authorizes access to a method in a RegistryObject if the  
1531 requesting Principal has any of the Privileges defined in the Permission.

1532 **See Also:**

1533 [Privilege](#), [AccessControlPolicy](#)

1534

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> .

1535

## 1536 11.3 Class Privilege

1537

1538 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute  
1539 can be a Group, a Role, or an Identity.

1540

1541 A requesting Principal MUST have all of the PrivilegeAttributes specified in a  
1542 Privilege in order to gain access to a method in a protected RegistryObject.  
1543 Permissions defined in the RegistryObject's AccessControlPolicy define the  
1544 Privileges that can authorize access to specific methods.

1545

1546 This mechanism enables the flexibility to have object access control policies that  
1547 are based on any combination of Roles, Identities or Groups.

1548 **See Also:**

1549 [PrivilegeAttribute](#), [Permission](#)

1550

1551

1552

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

1553

## 1554 11.4 Class PrivilegeAttribute

1555 **All Known Subclasses:**

1556 [Group](#), [Identity](#), [Role](#)

1557

1558 PrivilegeAttribute is a common base *Class* for all types of security attributes that  
1559 are used to grant specific access control privileges to a Principal. A Principal may  
1560 have several different types of PrivilegeAttributes. Specific combination of  
1561 PrivilegeAttributes may be defined as a Privilege object.

1562 **See Also:**

1563 [Principal](#), [Privilege](#)

## 1564 11.5 Class Role

1565 **All Superclasses:**

1566 [PrivilegeAttribute](#)

1567

### 1568 11.5.1 A security Role PrivilegeAttribute

1569 For example a hospital may have *Roles* such as Nurse, Doctor, Administrator  
1570 etc. Roles are used to grant Privileges to Principals. For example a Doctor *Role*  
1571 may be allowed to write a prescription but a Nurse *Role* may not.

## 1572 11.6 Class Group

1573 **All Superclasses:**

1574 [PrivilegeAttribute](#)

1575

### 1576 11.6.1 A security Group PrivilegeAttribute

1577 A Group is an aggregation of users that may have different Roles. For example  
1578 a hospital may have a Group defined for Nurses and Doctors that are  
1579 participating in a specific clinical trial (e.g., AspirinTrial group). Groups are used  
1580 to grant Privileges to Principals. For example the members of the AspirinTrial  
1581 group may be allowed to write a prescription for Aspirin (even though Nurse Role  
1582 as a rule may not be allowed to write prescriptions).

1583

1584

1585 **11.7 Class Identity**1586 **All Superclasses:**1587 [PrivilegeAttribute](#)

1588

1589 **11.7.1 A security Identity PrivilegeAttribute**

1590 This is typically used to identify a person, an organization, or software service.  
 1591 Identity attribute may be in the form of a digital certificate.

1592 **11.8 Class Principal**

1593

1594 Principal is a generic term used by the security community to include both people  
 1595 and software systems. The Principal object is an entity that has a set of  
 1596 PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and  
 1597 optionally a set of role memberships, group memberships or security clearances.  
 1598 A principal is used to authenticate a requestor and to authorize the requested  
 1599 action based on the PrivilegeAttributes associated with the Principal.

1600 **See Also:**1601 PrivilegeAttributes, [Privilege](#), [Permission](#)

1602

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> .

1603

1604

## 1604 **12 References**

- 1605 [ebGLOSS] ebXML Glossary,  
1606 [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)  
1607 [OAS] OASIS Information Model  
1608 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>  
1609 [ISO] ISO 11179 Information Model  
1610 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>  
1611  
1612 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use  
1613 in RFCs to Indicate Requirement Levels  
1614 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>  
1615 [ebRS] ebXML Registry Services Specification  
1616 <http://www.oasisopen.org/committees/regrep/documents/2.1/specs/ebRS.pdf>  
1617  
1618 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification  
1619 <http://www.ebxml.org/specrafts/>  
1620  
1621 [UUID] DCE 128 bit Universal Unique Identifier  
1622 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
1623 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>  
1624  
1625  
1626 [XPath] XML Path Language (XPath) Version 1.0  
1627 <http://www.w3.org/TR/xpath>  
1628  
1629 [NCName] Namespaces in XML 19990114  
1630 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

## 1631 **13 Disclaimer**

- 1632 The views and specification expressed in this document are those of the authors  
1633 and are not necessarily those of their employers. The authors and their  
1634 employers specifically disclaim responsibility for any problems arising from  
1635 correct or incorrect implementation or use of this design.  
1636

1636 **14 Contact Information**

1637

1638 Team Leader

1639 Name: Lisa Carnahan  
1640 Company: NIST  
1641 Street: 100 Bureau Drive STOP 8970  
1642 City, State, Postal Code: Gaithersburg, MD 20899-8970  
1643 Country: USA  
1644 Phone: (301) 975-3362  
1645 Email: lisa.carnahan@nist.gov

1646

1647 Editor

1648 Name: Sally Fuger  
1649 Company: Automotive Industry Action Group  
1650 Street: 26200 Lahser Road, Suite 200  
1651 City, State, Postal Code: Southfield, MI 48034  
1652 Country: USA  
1653 Phone: (248) 358-9744  
1654 Email: sfuger@aiag.org

1655

1656 Technical Editor

1657 Name: Farrukh S. Najmi  
1658 Company: Sun Microsystems  
1659 Street: 1 Network Dr., MS BUR02-302  
1660 City, State, Postal Code: Burlington, MA, 01803-0902  
1661 Country: USA  
1662 Phone: (781) 442-0703  
1663 Email: najmi@east.sun.com

1664

1665

**1665 Copyright Statement**

1666 OASIS takes no position regarding the validity or scope of any intellectual  
1667 property or other rights that might be claimed to pertain to the implementation or  
1668 use of the technology described in this document or the extent to which any  
1669 license under such rights might or might not be available; neither does it  
1670 represent that it has made any effort to identify any such rights. Information on  
1671 OASIS's procedures with respect to rights in OASIS specifications can be found  
1672 at the OASIS website. Copies of claims of rights made available for publication  
1673 and any assurances of licenses to be made available, or the result of an attempt  
1674 made to obtain a general license or permission for the use of such proprietary  
1675 rights by implementors or users of this specification, can be obtained from the  
1676 OASIS Executive Director.

1677  
1678 OASIS invites any interested party to bring to its attention any copyrights, patents  
1679 or patent applications, or other proprietary rights which may cover technology  
1680 that may be required to implement this specification. Please address the  
1681 information to the OASIS Executive Director.

1682  
1683 Copyright ©The Organization for the Advancement of Structured Information  
1684 Standards [OASIS] 2002. All Rights Reserved.

1685 This document and translations of it may be copied and furnished to others, and  
1686 derivative works that comment on or otherwise explain it or assist in its  
1687 implementation may be prepared, copied, published and distributed, in whole or  
1688 in part, without restriction of any kind, provided that the above copyright notice  
1689 and this paragraph are included on all such copies and derivative works.

1690 However, this document itself may not be modified in any way, such as by  
1691 removing the copyright notice or references to OASIS, except as needed for the  
1692 purpose of developing OASIS specifications, in which case the procedures for  
1693 copyrights defined in the OASIS Intellectual Property Rights document must be  
1694 followed, or as required to translate it into languages other than English.

1695 The limited permissions granted above are perpetual and will not be revoked by  
1696 OASIS or its successors or assigns.

1697 This document and the information contained herein is provided on an "AS IS"  
1698 basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
1699 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
1700 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
1701 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
1702 PURPOSE."