



Creating A Single Global Electronic Market

1  
2  
3  
4

## **OASIS/ebXML Registry Information Model v2.1**

### **Approved Committee Specification**

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

**June 2002**

9

## **1 Status of this Document**

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20

Distribution of this document is unlimited.

***This version:***

<http://www.oasis-open.org/committees/regrep/documents/2.1/specs/ebRIM.pdf>

***Latest version:***

<http://www.oasis-open.org/committees/regrep/documents/2.1/specs/ebRIM.pdf>

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

21 This document has been approved by the OASIS ebXML Registry TC as version  
22 2.1.

23

24 At the time of v2.1 committee approval, the following were members of the  
25 OASIS/ebXML Registry Technical Committee:

26

27 Kathryn Breininger, Boeing (TC Chair)

28 Zachary Alexander, Individual Member

29 Lisa Carnahan, US NIST

30 Martin Chapman, Oracle

31 Joseph M. Chiusano, LMI

32 Suresh Damodaran, Sterling Commerce

33 Mike DeNicola Fujitsu

34 Anne Fischer, Drummond Group

35 Sally Fuger, Individual Member

36 Yan Guo, WebMethods

37 Brian Hopkins, Individual Member

38 Jong Kim, InnoDigital

39 Kyu-Chul Lee, Individual Member

40 Joel Munter, Intel

41 Farrukh Najmi, Sun Microsystems

42 Sanjay Patil, IONA

43 Nikola Stojanovic, Encoda Systems, Inc.

44

45 Contributors

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

48

49 Len Gallagher, NIST

50 Sekhar Vajjhala, Sun Microsystems

51

52

52 **Table of Contents**

53

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

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

135	8.1.1	<i>Attribute Summary</i> .....	28
136	8.1.2	<i>Attribute eventType</i> .....	28
137	8.1.3	<i>Attribute registryObject</i> .....	28
138	8.1.4	<i>Attribute timestamp</i> .....	28
139	8.1.5	<i>Attribute user</i> .....	28
140	8.2	CLASS USER .....	29
141	8.2.1	<i>Attribute Summary</i> .....	29
142	8.2.2	<i>Attribute address</i> .....	29
143	8.2.3	<i>Attribute emailAddresses</i> .....	29
144	8.2.4	<i>Attribute organization</i> .....	29
145	8.2.5	<i>Attribute personName</i> .....	29
146	8.2.6	<i>Attribute telephoneNumbers</i> .....	30
147	8.2.7	<i>Attribute url</i> .....	30
148	8.3	CLASS ORGANIZATION .....	30
149	8.3.1	<i>Attribute Summary</i> .....	30
150	8.3.2	<i>Attribute address</i> .....	30
151	8.3.3	<i>Attribute parent</i> .....	30
152	8.3.4	<i>Attribute primaryContact</i> .....	30
153	8.3.5	<i>Attribute telephoneNumbers</i> .....	30
154	8.4	CLASS POSTALADDRESS .....	31
155	8.4.1	<i>Attribute Summary</i> .....	31
156	8.4.2	<i>Attribute city</i> .....	31
157	8.4.3	<i>Attribute country</i> .....	31
158	8.4.4	<i>Attribute postalCode</i> .....	31
159	8.4.5	<i>Attribute state</i> .....	31
160	8.4.6	<i>Attribute street</i> .....	31
161	8.4.7	<i>Attribute streetNumber</i> .....	31
162	8.4.8	<i>Method Summary</i> .....	32
163	8.5	CLASS TELEPHONENUMBER .....	32
164	8.5.1	<i>Attribute Summary</i> .....	32
165	8.5.2	<i>Attribute areaCode</i> .....	32
166	8.5.3	<i>Attribute countryCode</i> .....	32
167	8.5.4	<i>Attribute extension</i> .....	32
168	8.5.5	<i>Attribute number</i> .....	33
169	8.5.6	<i>Attribute phoneType</i> .....	33
170	8.6	CLASS EMAILADDRESS .....	33
171	8.6.1	<i>Attribute Summary</i> .....	33
172	8.6.2	<i>Attribute address</i> .....	33
173	8.6.3	<i>Attribute type</i> .....	33
174	8.7	CLASS PERSONNAME .....	33
175	8.7.1	<i>Attribute Summary</i> .....	33
176	8.7.2	<i>Attribute firstName</i> .....	33
177	8.7.3	<i>Attribute lastName</i> .....	34
178	8.7.4	<i>Attribute middleName</i> .....	34
179	8.8	CLASS SERVICE .....	34
180	8.8.1	<i>Attribute Summary</i> .....	34

181	8.8.2	<i>Method Summary</i> .....	34
182	8.9	CLASS SERVICEBINDING .....	34
183	8.9.1	<i>Attribute Summary</i> .....	35
184	8.9.2	<i>Attribute accessURI</i> .....	35
185	8.9.3	<i>Attribute targetBinding</i> .....	35
186	8.9.4	<i>Method Summary</i> .....	35
187	8.10	CLASS SPECIFICATIONLINK .....	35
188	8.10.1	<i>Attribute Summary</i> .....	36
189	8.10.2	<i>Attribute specificationObject</i> .....	36
190	8.10.3	<i>Attribute usageDescription</i> .....	36
191	8.10.4	<i>Attribute usageParameters</i> .....	36
192	<b>9</b>	<b>ASSOCIATION OF REGISTRY OBJECTS</b> .....	<b>37</b>
193	9.1	EXAMPLE OF AN ASSOCIATION .....	37
194	9.2	SOURCE AND TARGET OBJECTS .....	37
195	9.3	ASSOCIATION TYPES .....	37
196	9.4	INTRAMURAL ASSOCIATION.....	38
197	9.5	EXTRAMURAL ASSOCIATION.....	38
198	9.6	CONFIRMATION OF AN ASSOCIATION .....	39
199	9.6.1	<i>Confirmation of Intramural Associations</i> .....	39
200	9.6.2	<i>Confirmation of Extramural Associations</i> .....	40
201	9.6.3	<i>Deleting an Extramural Associations</i> .....	40
202	9.7	VISIBILITY OF UNCONFIRMED ASSOCIATIONS.....	40
203	9.8	POSSIBLE CONFIRMATION STATES .....	40
204	9.9	CLASS ASSOCIATION .....	41
205	9.9.1	<i>Attribute Summary</i> .....	41
206	9.9.2	<i>Attribute associationType</i> .....	41
207	9.9.3	<i>Attribute sourceObject</i> .....	42
208	9.9.4	<i>Attribute targetObject</i> .....	42
209	9.9.5	<i>Attribute isConfirmedBySourceOwner</i> .....	42
210	9.9.6	<i>Attribute isConfirmedByTargetOwner</i> .....	43
211	<b>10</b>	<b>CLASSIFICATION OF REGISTRYOBJECT</b> .....	<b>43</b>
212	10.1	CLASS CLASSIFICATIONSCHEME.....	46
213	10.1.1	<i>Attribute Summary</i> .....	46
214	10.1.2	<i>Attribute isInternal</i> .....	46
215	10.1.3	<i>Attribute nodeType</i> .....	46
216	10.2	CLASS CLASSIFICATIONNODE.....	47
217	10.2.1	<i>Attribute Summary</i> .....	47
218	10.2.2	<i>Attribute parent</i> .....	47
219	10.2.3	<i>Attribute code</i> .....	47
220	10.2.4	<i>Attribute path</i> .....	47
221	10.2.5	<i>Method Summary</i> .....	48
222	10.2.6	<i>Canonical Path Syntax</i> .....	48
223	10.3	CLASS CLASSIFICATION .....	49
224	10.3.1	<i>Attribute Summary</i> .....	49

225	10.3.2	<i>Attribute classificationScheme</i> .....	50
226	10.3.3	<i>Attribute classificationNode</i> .....	50
227	10.3.4	<i>Attribute classifiedObject</i> .....	50
228	10.3.5	<i>Attribute nodeRepresentation</i> .....	50
229	10.3.6	<i>Context Sensitive Classification</i> .....	50
230	10.3.7	<i>Method Summary</i> .....	52
231	10.4	EXAMPLE OF <i>CLASSIFICATION</i> SCHEMES.....	52
232	<b>11</b>	<b>INFORMATION MODEL: SECURITY VIEW</b> .....	<b>53</b>
233	11.1	CLASS ACCESSCONTROLPOLICY .....	54
234	11.2	CLASS PERMISSION .....	55
235	11.3	CLASS PRIVILEGE .....	55
236	11.4	CLASS PRIVILEGEATTRIBUTE .....	56
237	11.5	CLASS ROLE .....	56
238	11.5.1	<i>A security Role PrivilegeAttribute</i> .....	56
239	11.6	CLASS GROUP.....	56
240	11.6.1	<i>A security Group PrivilegeAttribute</i> .....	56
241	11.7	CLASS IDENTITY .....	57
242	11.7.1	<i>A security Identity PrivilegeAttribute</i> .....	57
243	11.8	CLASS PRINCIPAL .....	57
244	<b>12</b>	<b>REFERENCES</b> .....	<b>58</b>
245	<b>13</b>	<b>DISCLAIMER</b> .....	<b>58</b>
246	<b>14</b>	<b>CONTACT INFORMATION</b> .....	<b>59</b>
247		<b>COPYRIGHT STATEMENT</b> .....	<b>60</b>

## 248 **Table of Figures**

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

## 259 **Table of Tables**

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

261

262

## 262 **3 Introduction**

### 263 **3.1 Summary of Contents of Document**

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

265

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

### 269 **3.2 General Conventions**

270 The following conventions are used throughout this document:

271

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

274

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

278

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

281

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

285

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

287

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

291

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

#### 294 **3.2.1 Naming Conventions**

295

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

298 Capitalization styles are used in the following conventions:

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



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

308

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

### 310 **3.3 Audience**

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

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

### 315 **3.4 Related Documents**

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

318

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

## 325 **4 Design Objectives**

### 326 **4.1 Goals**

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

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

## 336 **5 System Overview**

### 337 **5.1 Role of ebXML Registry**

338

339 The *Registry* provides a stable store where information submitted by a  
340 *Submitting Organization* is made persistent. Such information is used to facilitate  
341 ebXML-based *Business to Business* (B2B) partnerships and transactions.  
342 Submitted content may be *XML* schema and documents, process descriptions,  
343 ebXML *Core Components*, context descriptions, *UML* models, information about  
344 parties and even software components.

### 345 **5.2 Registry Services**

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

### 350 **5.3 What the Registry Information Model Does**

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

355 The Registry information model:

- 356     o Defines what types of objects are stored in the *Registry*
- 357     o Defines how stored objects are organized in the *Registry*

358

### 359 **5.4 How the Registry Information Model Works**

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

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

368

### 369 **5.5 Where the Registry Information Model May Be Implemented**

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

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

374 **5.6 Conformance to an ebXML Registry**

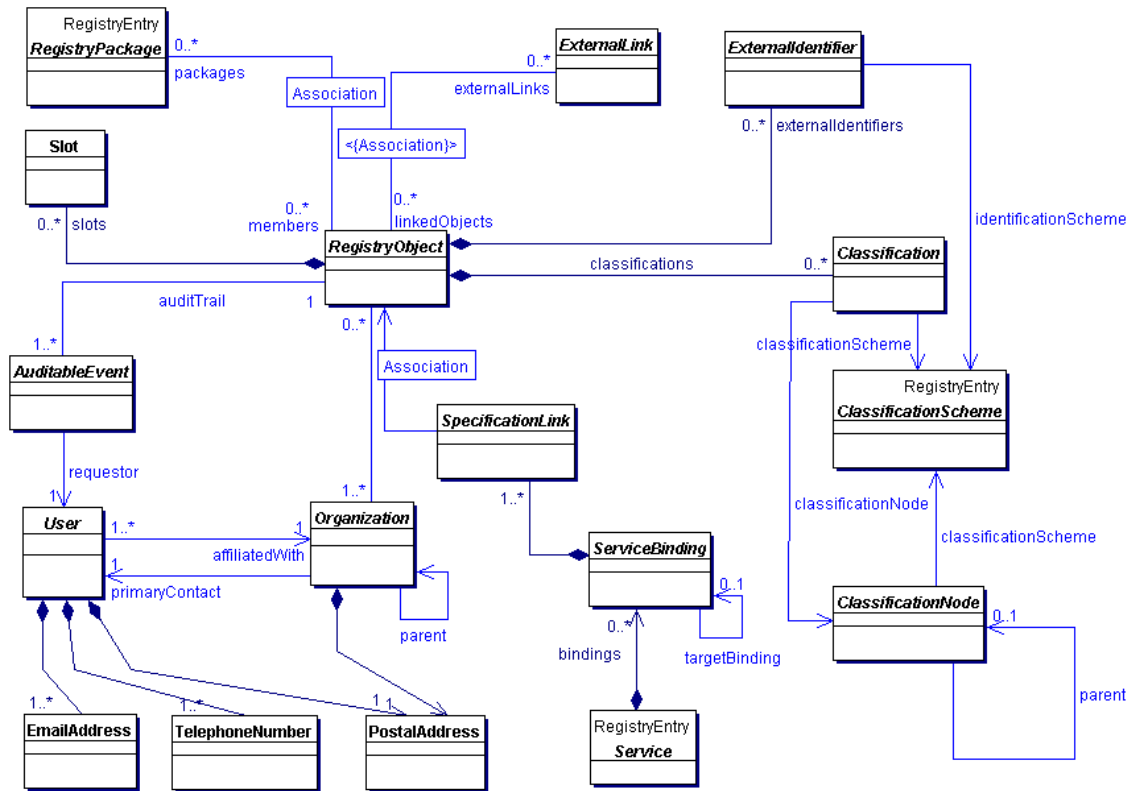
375 If an *Implementation* claims *Conformance* to this specification then it supports all  
376 required information model *Classes* and interfaces, their attributes and their  
377 semantic definitions that are visible through the ebXML *Registry Services*.

378 **6 Registry Information Model: High Level Public View**

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

381  
382 Figure 1 shows the high level public view of the objects in the *Registry* and their  
383 relationships as a *UML Class Diagram*. It does not show *Inheritance*, *Class*  
384 attributes or *Class* methods.

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



388

389

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

## 390 **6.1 RegistryObject**

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

## 395 **6.2 Slot**

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

## 402 **6.3 Association**

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

## 406 **6.4 ExternalIdentifier**

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

## 410 **6.5 ExternalLink**

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

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

## 422 **6.6 ClassificationScheme**

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

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

### 431 **6.7 ClassificationNode**

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

### 438 **6.8 Classification**

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

### 445 **6.9 RegistryPackage**

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

### 448 **6.10 AuditableEvent**

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

### 452 **6.11 User**

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

### 456 **6.12 PostalAddress**

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

### 459 **6.13 EmailAddress**

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

## 462 **6.14 Organization**

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

## 466 **6.15 Service**

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

## 469 **6.16 ServiceBinding**

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

## 474 **6.17 SpecificationLink**

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

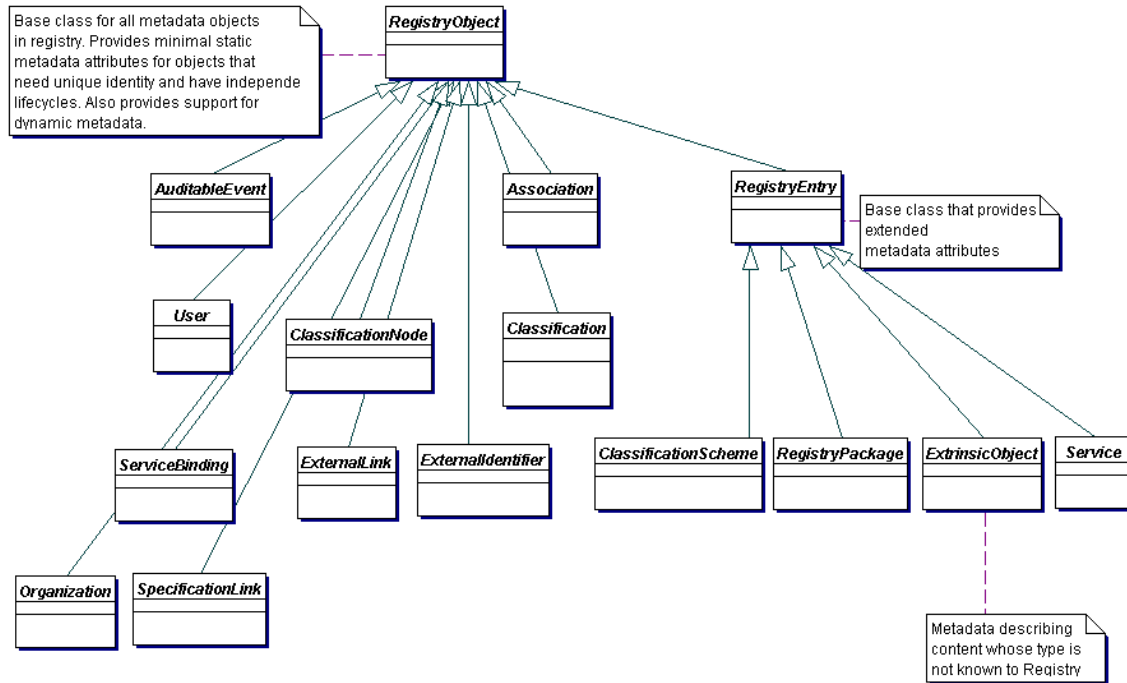
## 481 **7 Registry Information Model: Detail View**

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

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

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

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



499  
500  
501

Figure 2: Information Model *Inheritance View*

502 **7.1 Attribute and Methods of Information Model Classes**

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

507  
508 Information model classes may also have methods defined for them. These  
509 methods provide additional behavior for the class they are defined within.  
510 Methods are currently used in mapping to filter query and the SQL query  
511 capabilities defined in [ebRS].

512  
513 Since the model supports inheritance between classes, it is usually the case that  
514 a class in the model inherits attributes and methods from its base classes, in  
515 addition to defining its own specialized attributes and methods.

516

## 516 7.2 Data Types

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

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	

520

## 521 7.3 Internationalization (I18N) Support

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

525

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

### 529 7.3.1 Class InternationalString

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



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

#### 536 7.3.1.1 Attribute Summary

537

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

538

#### 539 7.3.1.2 Attribute localizedStrings

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

### 542 7.3.2 Class LocalizedString

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

#### 547 7.3.2.1 Attribute Summary

548

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

549

#### 550 7.3.2.2 Attribute lang

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

#### 553 7.3.2.3 Attribute charset

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

#### 556 7.3.2.4 Attribute value

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

## 559 7.4 Class RegistryObject

### 560 Direct Known Subclasses:

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

OASIS/ebXML Registry Information Model

Page 17

564  
565 RegistryObject provides a common base class for almost all objects in the  
566 information model. Information model *Classes* whose instances have a unique  
567 identity are descendants of the RegistryObject *Class*.

568  
569 Note that Slot, PostalAddress, and a few other classes are not descendants of  
570 the RegistryObject Class because their instances do not have an independent  
571 existence and unique identity. They are always a part of some other Class's  
572 Instance (e.g., Organization has a PostalAddress).

### 573 **7.4.1 Attribute Summary**

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

576

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

577

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

### 578 **7.4.2 Attribute accessControlPolicy**

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

### 583 **7.4.3 Attribute description**

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

#### 587 **7.4.4 Attribute id**

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

591  
592 Note that some classes in the information model do not have a need for a unique  
593 id. Such classes do not inherit from RegistryObject class. Examples include  
594 Entity classes such as TelephoneNumber, PostalAddress, EmailAddress and  
595 PersonName.

596  
597 All classes derived from RegistryObject have an id that is a Universally Unique ID  
598 as defined by [UUID]. Such UUID based id attributes may be specified by the  
599 client. If the UUID based id is not specified, then it must be generated by the  
600 registry when a new RegistryObject instance is first submitted to the registry.

#### 601 **7.4.5 Attribute name**

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

#### 605 **7.4.6 Attribute objectType**

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

##### 611 **7.4.6.1 Pre-defined Object Types**

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

616  
617  
618 These pre-defined object types are defined as a *ClassificationScheme*. While the  
619 scheme may easily be extended a *Registry* MUST support the object types listed  
620 below.

621

<b>Name</b>	<b>description</b>
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	An ExtrinsicObject of this type catalogues an 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 Schema</i> , <i>RELAX</i> 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

622

623 **7.4.7 Method Summary**

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

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.

628

629

## 630 7.5 Class RegistryEntry

### 631 Super Classes:

632 [RegistryObject](#)

633

### 634 Direct Known Subclasses:

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

636

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

645

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

### 648 7.5.1 Attribute Summary

649

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

650

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

### 653 **7.5.2 Attribute expiration**

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

### 660 **7.5.3 Attribute majorVersion**

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

### 665 **7.5.4 Attribute minorVersion**

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

### 670 **7.5.5 Attribute stability**

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

#### 674 **7.5.5.1 Pre-defined RegistryEntry Stability Enumerations**

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

679

<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.

680

681 **7.5.6 Attribute status**

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

684 **7.5.6.1 Pre-defined RegistryObject Status Types**

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

687

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

688

689 **7.5.7 Attribute userVersion**

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

694 **7.6 Class Slot**

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

698

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

701 **7.6.1 Attribute Summary**

702

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

703

704 **7.6.2 Attribute name**

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

708 **7.6.3 Attribute slotType**

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

711 **7.6.4 Attribute values**

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

717 **7.7 Class ExtrinsicObject**718 **Super Classes:**

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

720

721

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

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

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

732 **7.7.1 Attribute Summary**

733

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

734

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



### 737 **7.7.2 Attribute isOpaque**

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

### 743 **7.7.3 Attribute mimeType**

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

## 748 **7.8 Class RegistryPackage**

### 749 **Super Classes:**

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

751 

---

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

### 755 **7.8.1 Attribute Summary**

756

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

### 760 **7.8.2 Method Summary**

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

763

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

764

## 765 **7.9 Class ExternalIdentifier**

### 766 **Super Classes:**

767 [RegistryObject](#)

768 

---

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

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

### 776 **7.9.1 Attribute Summary**

777

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

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

### 779 **7.9.2 Attribute identificationScheme**

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

### 784 **7.9.3 Attribute registryObject**

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

### 787 **7.9.4 Attribute value**

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

## 790 **7.10 Class ExternalLink**

### 791 **Super Classes:**

792 [RegistryObject](#)

793

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

### 798 **7.10.1 Attribute Summary**

799

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

800

### 801 **7.10.2 Attribute externalURI**

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

### 807 **7.10.3 Method Summary**

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

810

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.

811

## 812 **8 Registry Audit Trail**

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

817

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

### 824 **8.1 Class AuditableEvent**

825 **Super Classes:**

826 [RegistryObject](#)

827

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

832

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

835

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

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

### 843 8.1.1 Attribute Summary

844

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

845

### 846 8.1.2 Attribute eventType

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

#### 849 8.1.2.1 Pre-defined Auditable Event Types

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

853

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.

### 854 8.1.3 Attribute registryObject

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

### 857 8.1.4 Attribute timestamp

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

### 860 8.1.5 Attribute user

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

863  
864

## 865 **8.2 Class User**

866 **Super Classes:**  
867 [RegistryObject](#)

868

---

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

### 871 **8.2.1 Attribute Summary**

872

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

873

### 874 **8.2.2 Attribute address**

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

### 877 **8.2.3 Attribute emailAddresses**

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

### 881 **8.2.4 Attribute organization**

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

### 884 **8.2.5 Attribute personName**

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

### 887 **8.2.6 Attribute telephoneNumbers**

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

### 891 **8.2.7 Attribute url**

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

## 894 **8.3 Class Organization**

### 895 **Super Classes:**

896 [RegistryObject](#)

897 

---

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

### 901 **8.3.1 Attribute Summary**

902

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

903

### 904 **8.3.2 Attribute address**

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

### 907 **8.3.3 Attribute parent**

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

### 910 **8.3.4 Attribute primaryContact**

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

### 913 **8.3.5 Attribute telephoneNumbers**

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

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

## 918 **8.4 Class PostalAddress**

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

### 921 **8.4.1 Attribute Summary**

922

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

923

### 924 **8.4.2 Attribute city**

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

### 926 **8.4.3 Attribute country**

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

### 929 **8.4.4 Attribute postalCode**

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

### 932 **8.4.5 Attribute state**

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

### 935 **8.4.6 Attribute street**

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

### 938 **8.4.7 Attribute streetNumber**

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

941 **8.4.8 Method Summary**

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

944

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>

945

946 **8.5 Class TelephoneNumber**

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

948 **8.5.1 Attribute Summary**

949

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

950

951 **8.5.2 Attribute areaCode**

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

954 **8.5.3 Attribute countryCode**

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

957 **8.5.4 Attribute extension**

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



960 **8.5.5 Attribute number**

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

964 **8.5.6 Attribute phoneType**

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

968 **8.6 Class EmailAddress**

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

970 **8.6.1 Attribute Summary**

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

971 **8.6.2 Attribute address**

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

974 **8.6.3 Attribute type**

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

978 **8.7 Class PersonName**

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

980 **8.7.1 Attribute Summary**

981

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

982 **8.7.2 Attribute firstName**

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

985 **8.7.3 Attribute lastName**

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

988 **8.7.4 Attribute middleName**

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

991 **8.8 Class Service**

992 **Super Classes:**

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

994

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

996 **8.8.1 Attribute Summary**

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

999 **8.8.2 Method Summary**

1000 In addition to its attributes, the Service class also defines the following methods.  
1001

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

1002 **8.9 Class ServiceBinding**

1003 **Super Classes:**

1004 [RegistryObject](#)

1005

1006 ServiceBinding instances are RegistryObjects that represent technical  
1007 information on a specific way to access a specific interface offered by a Service  
1008 instance. A Service has a Collection of ServiceBindings.

1009 The description attribute of ServiceBinding provides details about the relationship  
1010 between several specification links comprising the Service Binding. This  
1011 description can be useful for human understanding such that the runtime system  
1012 can be appropriately configured by the human being. There is possibility of  
1013 enforcing a structure on this description for enabling machine processing of the  
1014 Service Binding, which is however not addressed by the current document.

1015

1016

1017 **8.9.1 Attribute Summary**

1018

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

1019

1020 **8.9.2 Attribute accessURI**

1021 A ServiceBinding may have an accessURI attribute that defines the URI to  
 1022 access that ServiceBinding. This attribute is ignored if a targetBinding attribute is  
 1023 specified for the ServiceBinding. If the URI is a URL then a registry must validate  
 1024 the URL to be resolvable at the time of submission before accepting a  
 1025 ServiceBinding submission to the registry.

1026 **8.9.3 Attribute targetBinding**

1027 A ServiceBinding may have a targetBinding attribute defined which references  
 1028 another ServiceBinding. A targetBinding may be specified when a service is  
 1029 being redirected to another service. This allows the rehosting of a service by  
 1030 another service provider.

1031 **8.9.4 Method Summary**

1032 In addition to its attributes, the ServiceBinding class also defines the following  
 1033 methods.

1034

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

1035

1036

1037

1038 **8.10 Class SpecificationLink**1039 **Super Classes:**1040 [RegistryObject](#)

1041

1042 A SpecificationLink provides the linkage between a ServiceBinding and one of its  
 1043 technical specifications that describes how to use the service using the  
 1044 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink  
 1045 instances that describe how to access the service using a technical specification  
 1046 in form of a WSDL document or a CORBA IDL document.

1047 **8.10.1 Attribute Summary**

1048

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

1049

1050 **8.10.2 Attribute specificationObject**

1051 A SpecificationLink instance must have a specificationObject attribute that  
 1052 provides a reference to a RegistryObject instance that provides a technical  
 1053 specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject  
 1054 instance representing the technical specification (e.g., a WSDL document).

1055 **8.10.3 Attribute usageDescription**

1056 A SpecificationLink instance may have a usageDescription attribute that provides  
 1057 a textual description of how to use the optional usageParameters attribute  
 1058 described next. The usageDescription is of type InternationalString, thus allowing  
 1059 the description to be in multiple languages.

1060 **8.10.4 Attribute usageParameters**

1061 A SpecificationLink instance may have a usageParameters attribute that provides  
 1062 a collection of Strings representing the instance specific parameters needed to  
 1063 use the technical specification (e.g., a WSDL document) specified by this  
 1064 SpecificationLink object.

1065

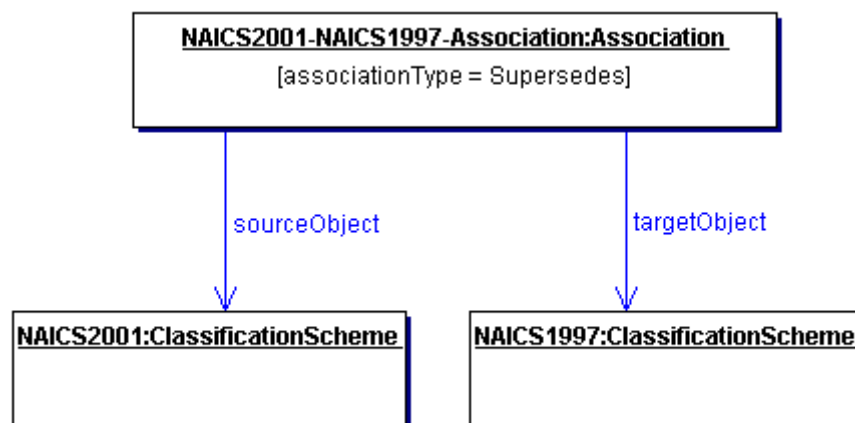
## 1065 9 Association of Registry Objects

1066 A RegistryObject instance may be *associated* with zero or more RegistryObject  
 1067 instances. The information model defines an Association class, an instance of  
 1068 which may be used to associate any two RegistryObject instances.

### 1069 9.1 Example of an Association

1070 One example of such an association is between two ClassificationScheme  
 1071 instances, where one ClassificationScheme supersedes the other  
 1072 ClassificationScheme as shown in Figure 3. This may be the case when a new  
 1073 version of a ClassificationScheme is submitted.

1074 In Figure 3, we see how an Association is defined between a new version of the  
 1075 NAICS ClassificationScheme and an older version of the NAICS  
 1076 ClassificationScheme.  
 1077



1078

1079

Figure 3: Example of RegistryObject Association

### 1080 9.2 Source and Target Objects

1081 An Association instance represents an association between a *source*  
 1082 RegistryObject and a *target* RegistryObject. These are referred to as  
 1083 *sourceObject* and *targetObject* for the Association instance. It is important which  
 1084 object is the sourceObject and which is the targetObject as it determines the  
 1085 directional semantics of an Association.

1086 In the example in Figure 3, it is important to make the newer version of NAICS  
 1087 ClassificationScheme be the sourceObject and the older version of NAICS be the  
 1088 targetObject because the associationType implies that the sourceObject  
 1089 supersedes the targetObject (and not the other way around).

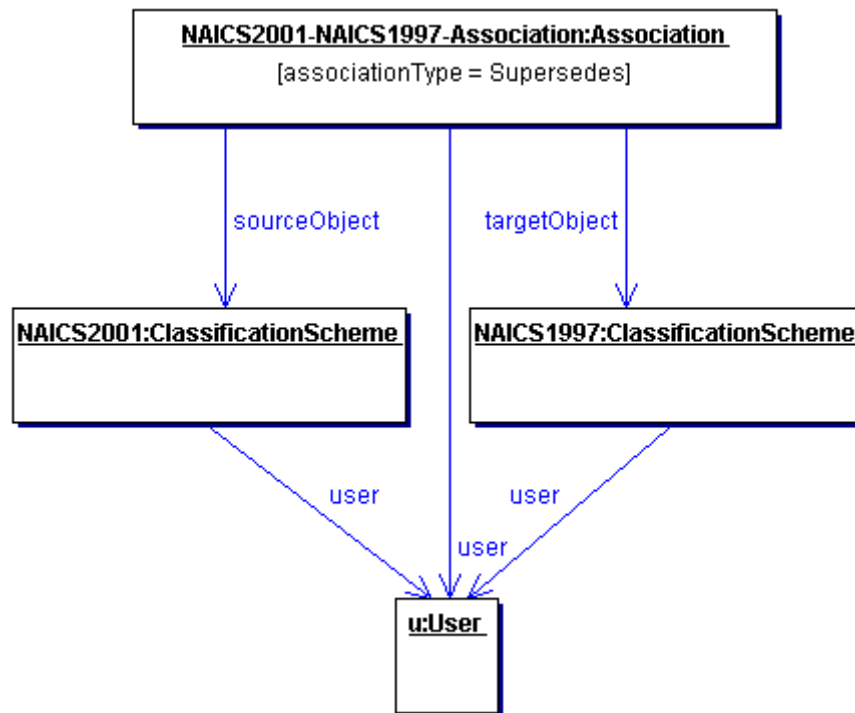
### 1090 9.3 Association Types

1091 Each Association must have an associationType attribute that identifies the type  
 1092 of that association.

## 1093 9.4 Intramural Association

1094 A common use case for the Association class is when a User “u” creates an  
 1095 Association “a” between two RegistryObjects “o1” and “o2” where association “a”  
 1096 and RegistryObjects “o1” and “o2” are objects that were created by the same  
 1097 User “u.” This is the simplest use case, where the association is between two  
 1098 objects that are owned by the same User that is defining the Association. Such  
 1099 associations are referred to as *intramural associations*.  
 1100 Figure 4 below, extends the previous example in Figure 3 for the intramural  
 1101 association case.

1102



1103

1104

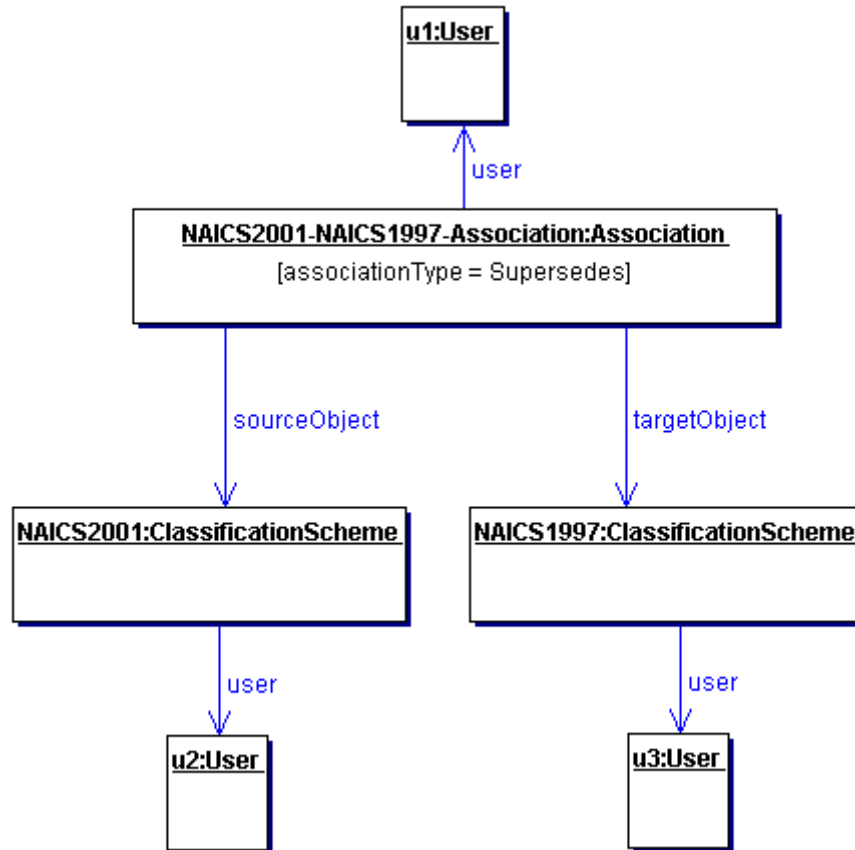
Figure 4: Example of Intramural Association

## 1105 9.5 Extramural Association

1106 The information model also allows more sophisticated use cases. For example, a  
 1107 User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”  
 1108 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2”  
 1109 are owned by User “u2” and User “u3” respectively.

1110 In this use case an Association is defined where either or both objects that are  
 1111 being associated are owned by a User different from the User defining the  
 1112 Association. Such associations are referred to as *extramural associations*. The  
 1113 Association class provides a convenience method called `isExtramural` that  
 1114 returns "true" if the Association instance is an extramural Association.

1115 Figure 5 below, extends the previous example in Figure 3 for the extramural  
 1116 association case. Note that it is possible for an extramural association to have  
 1117 two distinct Users rather than three distinct Users as shown in Figure 5. In such  
 1118 case, one of the two users owns two of the three objects involved (Association,  
 1119 sourceObject and targetObject).  
 1120



1121  
 1122

Figure 5: Example of Extramural Association

## 1123 9.6 Confirmation of an Association

1124 An association may need to be confirmed by the parties whose objects are  
 1125 involved in that Association as the sourceObject or targetObject. This section  
 1126 describes the semantics of confirmation of an association by the parties involved.

### 1127 9.6.1 Confirmation of Intramural Associations

1128 Intramural associations may be viewed as declarations of truth and do not  
 1129 require any explicit steps to confirm that Association as being true. In other  
 1130 words, intramural associations are implicitly considered confirmed.

### 1131 **9.6.2 Confirmation of Extramural Associations**

1132 An extramural association may be thought of as a unilateral assertion that may  
1133 not be viewed as truth until it has been confirmed by the other (extramural)  
1134 parties involved (Users “u2” and “u3” in the example in section 9.5).  
1135 To confirm an extramural association, each of the extramural parties (parties that  
1136 own the source or target object but do not own the Association) must submit an  
1137 identical Association (clone Association) as the Association they are intending to  
1138 confirm using a SubmitObjectsRequest. The clone Association must have the  
1139 same id as the original Association.

### 1140 **9.6.3 Deleting an Extramural Associations**

1141 An Extramural Association is deleted like any other type of RegistryObject, using  
1142 the RemoveObjectsRequest as defined in [ebRS]. However, in some cases  
1143 deleting an extramural Association may not actually delete it but instead only  
1144 revert a confirmed association to unconfirmed state.  
1145

1146 An Association must always be deleted when deleted by the owner of that  
1147 Association, irrespective of its confirmation state. An extramural Association must  
1148 become unconfirmed by the owner of its source/target object when deleted by  
1149 the owner of its source/target object when the requestor is not the owner of the  
1150 Association itself.

### 1151 **9.7 Visibility of Unconfirmed Associations**

1152 Extramural associations require each extramural party to confirm the assertion  
1153 being made by the extramural Association before the Association is visible to  
1154 third parties that are not involved in the Association. This ensures that  
1155 unconfirmed Associations are not visible to third party registry clients.

### 1156 **9.8 Possible Confirmation States**

1157 Assume the most general case where there are three distinct User instances as  
1158 shown in Figure 5 for an extramural Association. The extramural Association  
1159 needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3”  
1160 in example) in order to be fully confirmed. The methods  
1161 `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the  
1162 Association class provide access to the confirmation state for both the  
1163 sourceObject and targetObject. A third convenience method called  
1164 `isConfirmed` provides a way to determine whether the Association is fully  
1165 confirmed or not. So there are the following four possibilities related to the  
1166 confirmation state of an extramural Association:

- 1167 ○ The Association is confirmed neither by the owner of the sourceObject nor  
1168 by the owner of the targetObject.
- 1169 ○ The Association is confirmed by the owner of the sourceObject but it is not  
1170 confirmed by the owner of the targetObject.
- 1171 ○ The Association is not confirmed by the owner of the sourceObject but it is  
1172 confirmed by the owner of the targetObject.



- 1173 ○ The Association is confirmed by both the owner of the sourceObject and
- 1174 the owner of the targetObject. This is the only state where the Association
- 1175 is fully confirmed.
- 1176

1177 **9.9 Class Association**

1178 **Super Classes:**  
 1179 [RegistryObject](#)

1180 

---

1181

1182 Association instances are used to define many-to-many associations among

1183 RegistryObjects in the information model.

1184

1185 An *Instance* of the Association *Class* represents an association between two

1186 RegistryObjects.

1187 **9.9.1 Attribute Summary**

1188

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

1189

1190 **9.9.2 Attribute associationType**

1191 Each Association must have an associationType attribute that identifies the type

1192 of that association.

1193 **9.9.2.1 Pre-defined Association Types**

1194 The following table lists pre-defined association types. These pre-defined

1195 association types are defined as a *Classification* scheme. While the scheme may

1196 easily be extended a *Registry* MUST support the association types listed below.

1197

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.

1198

### 1199 **9.9.3 Attribute sourceObject**

1200 Each Association must have a sourceObject attribute that references the  
1201 RegistryObject instance that is the source of that association.

### 1202 **9.9.4 Attribute targetObject**

1203 Each Association must have a targetObject attribute that references the  
1204 RegistryObject instance that is the target of that association.

### 1205 **9.9.5 Attribute isConfirmedBySourceOwner**

1206 Each Association may have an isConfirmedBySourceOwner attribute that is set  
1207 by the registry to be true if the association has been confirmed by the owner of

1208 the sourceObject. For intramural Associations this attribute is always true. This  
 1209 attribute must be present when the object is retrieved from the registry. This  
 1210 attribute must be ignored if specified by the client when the object is submitted to  
 1211 the registry.

### 1212 **9.9.6 Attribute isConfirmedByTargetOwner**

1213 Each Association may have an isConfirmedByTargetOwner attribute that is set  
 1214 by the registry to be true if the association has been confirmed by the owner of  
 1215 the targetObject. For intramural Associations this attribute is always true. This  
 1216 attribute must be present when the object is retrieved from the registry. This  
 1217 attribute must be ignored if specified by the client when the object is submitted to  
 1218 the registry.  
 1219

Method Summary of Association	
Boolean	<p><a href="#"><u>isConfirmed()</u></a>            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="#"><u>isExtramural()</u></a>            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>

1220

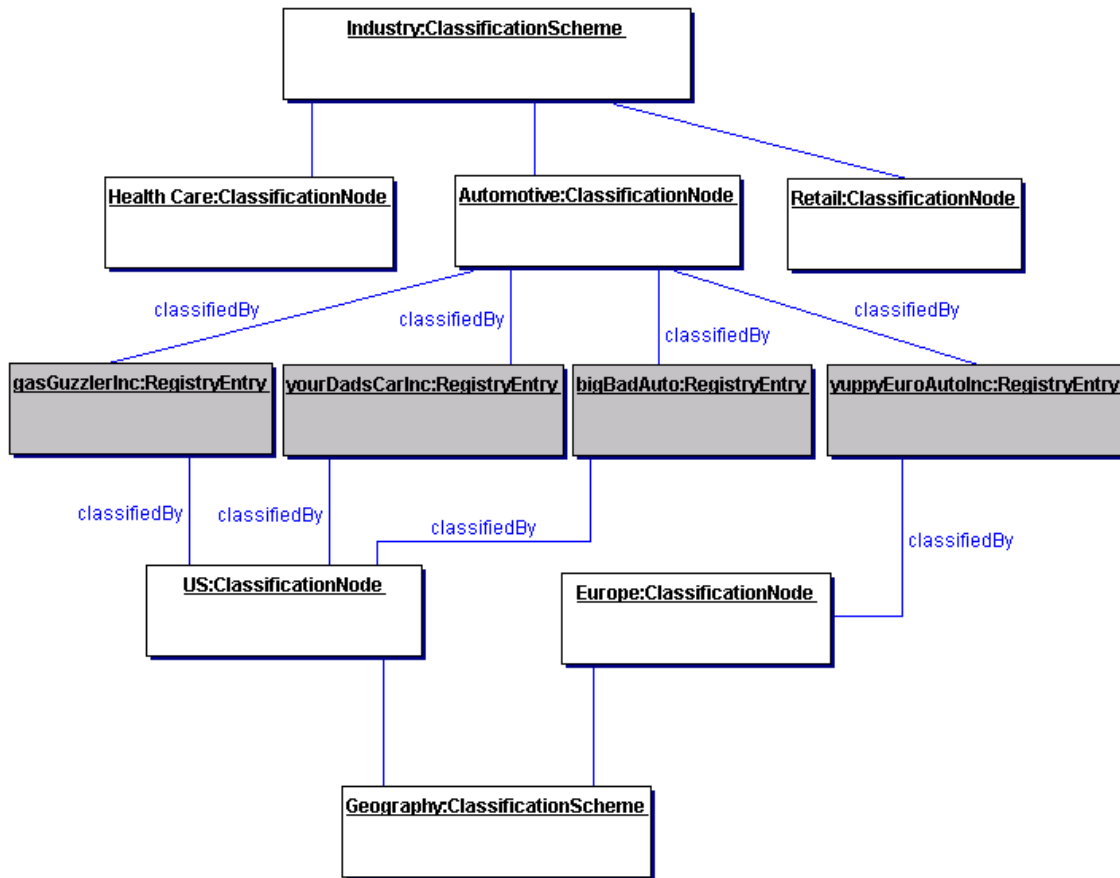
## 1221 **10 Classification of RegistryObject**

1222 This section describes the how the information model supports *Classification of*  
 1223 *RegistryObject*. It is a simplified version of the *OASIS* classification model [OAS].  
 1224

1225 A RegistryObject may be classified in many ways. For example the  
 1226 RegistryObject for the same *Collaboration Protocol Profile (CPP)* may be  
 1227 classified by its industry, by the products it sells and by its geographical location.  
 1228

1229 A general *ClassificationScheme* can be viewed as a *Classification* tree. In the  
 1230 example shown in Figure 6, RegistryObject instances representing *Collaboration*  
 1231 *Protocol Profiles* are shown as shaded boxes. Each *Collaboration Protocol*  
 1232 *Profile* represents an automobile manufacturer. Each *Collaboration Protocol*  
 1233 *Profile* is classified by the ClassificationNode named "Automotive" under the  
 1234 ClassificationScheme instance with name "Industry." Furthermore, the US  
 1235 Automobile manufacturers are classified by the US ClassificationNode under the  
 1236 ClassificationScheme with name "Geography." Similarly, a European automobile  
 1237 manufacturer is classified by the "Europe" ClassificationNode under the  
 1238 ClassificationScheme with name "Geography."  
 1239

1240 The example shows how a RegistryObject may be classified by multiple  
 1241 ClassificationNode instances under multiple ClassificationScheme instances  
 1242 (e.g., Industry, Geography).  
 1243



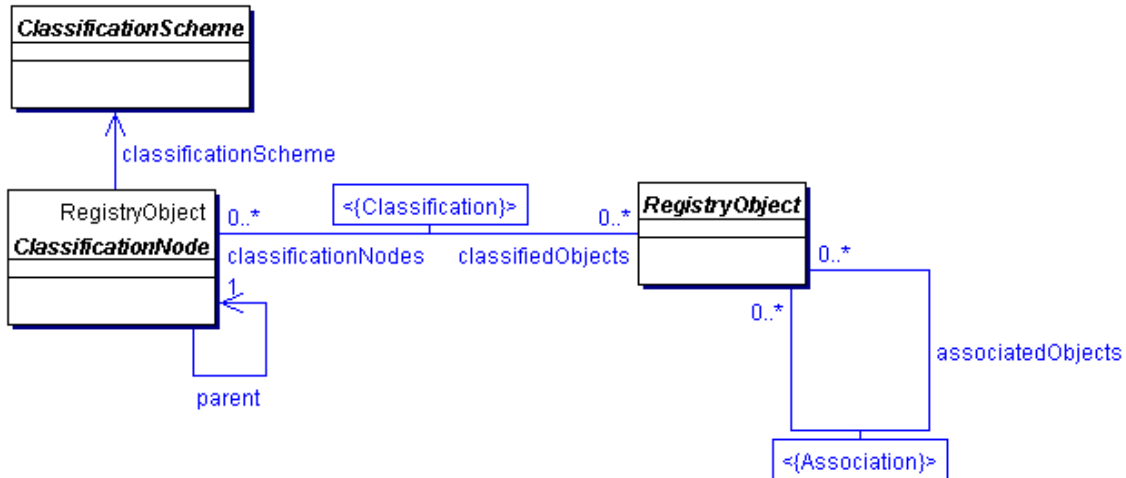
1244  
 1245

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

1246 [Note]It is important to point out that the dark  
 1247 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are  
 1248 not part of the *Classification tree*. The leaf  
 1249 nodes of the *Classification tree* are Health  
 1250 Care, Automotive, Retail, US and Europe. The  
 1251 dark nodes are associated with the  
 1252 *Classification tree* via a *Classification*  
 1253 *Instance* that is not shown in the picture  
 1254

1255  
 1256  
 1257

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.



1258

1259

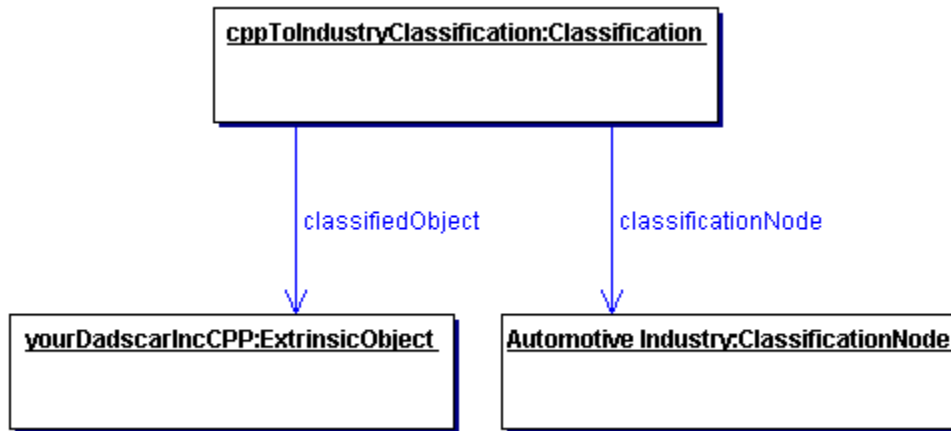
Figure 7: Information Model *Classification View*

1260

1261

1262 A Classification is somewhat like a specialized form of an Association. Figure 8  
 1263 shows an example of an ExtrinsicObject *Instance* for a *Collaboration Protocol*  
 1264 *Profile (CPP)* object that is classified by a ClassificationNode representing the  
 1265 Industry that it belongs to.

1266



1267

1268

Figure 8: Classification *Instance Diagram*

1269

1270

1271

1272

1273

1274

## 1275 **10.1 Class ClassificationScheme**

### 1276 **Base classes:**

1277 [RegistryEntry](#), [RegistryObject](#)

---

1278  
1279 A ClassificationScheme instance is metadata that describes a registered  
1280 taxonomy. The taxonomy hierarchy may be defined internally to the  
1281 Registry by instances of ClassificationNode or it may be defined externally  
1282 to the Registry, in which case the structure and values of the taxonomy  
1283 elements are not known to the Registry.  
1284 In the first case the classification scheme is defined to be *internal* and in  
1285 the second case the classification scheme is defined to be *external*.  
1286 The ClassificationScheme class inherits attributes and methods from the  
1287 RegistryObject and RegistryEntry classes.  
1288

### 1289 **10.1.1 Attribute Summary**

1290

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

1291 Note that attributes inherited by ClassificationScheme class from the  
1292 RegistryEntry class are not shown.

1293

### 1294 **10.1.2 Attribute isInternal**

1295 When submitting a ClassificationScheme instance the Submitting Organization  
1296 needs to declare whether the ClassificationScheme instance represents an  
1297 internal or an external taxonomy. This allows the registry to validate the  
1298 subsequent submissions of ClassificationNode and Classification instances in  
1299 order to maintain the type of ClassificationScheme consistent throughout its  
1300 lifecycle.

1301

### 1302 **10.1.3 Attribute nodeType**

1303 When submitting a ClassificationScheme instance the Submitting Organization  
1304 needs to declare what is the structure of taxonomy nodes that this  
1305 ClassificationScheme instance will represent. This attribute is an enumeration  
1306 with the following values:

- 1307 - UniqueCode. This value says that each node of the taxonomy has  
1308 a unique code assigned to it.
- 1309 - EmbeddedPath. This value says that a unique code assigned to  
1310 each node of the taxonomy at the same time encodes its path. This  
1311 is the case in the NAICS taxonomy.

1312 - NonUniqueCode. In some cases nodes are not unique, and it is  
 1313 necessary to nominate the full path in order to identify the node. For  
 1314 example, in a geography taxonomy Moscow could be under both  
 1315 Russia and the USA, where there are five cities of that name in  
 1316 different states.

1317 This enumeration might expand in the future with some new values. An example  
 1318 for possible future values for this enumeration might be NamedPathElements for  
 1319 support of Named-Level taxonomies such as Genus/Species.  
 1320

## 1321 10.2 Class ClassificationNode

1322 **Base classes:**

1323 [RegistryObject](#)

---

1324 ClassificationNode instances are used to define tree structures where  
 1325 each node in the tree is a ClassificationNode. Such *Classification* trees  
 1326 are constructed with ClassificationNode instances under a  
 1327 ClassificationScheme instance, and are used to define *Classification*  
 1328 schemes or ontologies.  
 1329  
 1330

### 1331 10.2.1 Attribute Summary

1332

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

1333

### 1334 10.2.2 Attribute parent

1335 Each ClassificationNode may have a parent attribute. The parent attribute either  
 1336 references a parent ClassificationNode or a ClassificationScheme instance in  
 1337 case of first level ClassificationNode instances.  
 1338

### 1339 10.2.3 Attribute code

1340 Each ClassificationNode may have a code attribute. The code attribute contains  
 1341 a code within a standard coding scheme.

### 1342 10.2.4 Attribute path

1343 Each ClassificationNode may have a path attribute. The path attribute must be  
 1344 present when a ClassificationNode is retrieved from the registry. The path  
 1345 attribute must be ignored when the path is specified by the client when the object

1346 is submitted to the registry. The path attribute contains the canonical path from  
 1347 the ClassificationScheme of this ClassificationNode. The path syntax is defined  
 1348 in 10.2.6.

### 1349 **10.2.5 Method Summary**

1350 In addition to its attributes, the ClassificationNode class also defines the following  
 1351 methods.

1352

<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.

1353

1354 In Figure 6, several instances of ClassificationNode are defined (all light colored  
 1355 boxes). A ClassificationNode has zero or one parent and zero or more  
 1356 ClassificationNodes for its immediate children. The parent of a  
 1357 ClassificationNode may be another ClassificationNode or a ClassificationScheme  
 1358 in case of first level ClassificationNodes.  
 1359

### 1360 **10.2.6 Canonical Path Syntax**

1361 The path attribute of the ClassificationNode class contains an absolute path in a  
 1362 canonical representation that uniquely identifies the path leading from the  
 1363 ClassificationScheme to that ClassificationNode.

1364 The canonical path representation is defined by the following BNF grammar:

1365

```

1366 canonicalPath ::= '/' schemeld nodePath
1367 nodePath     ::= '/' nodeCode
1368              | '/' nodeCode ( nodePath )?
  
```

1369

1370 In the above grammar, schemeld is the id attribute of the ClassificationScheme  
 1371 instance, and nodeCode is defined by NCName production as defined by  
 1372 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1373



### 1374 10.2.6.1 Example of Canonical Path Representation

1375 The following canonical path represents what the path attribute would contain for  
 1376 the ClassificationNode with code 'United States' in the sample Geography  
 1377 scheme in section 10.2.6.2.

```
1378 /Geography-id/NorthAmerica/UnitedStates
```

### 1380 10.2.6.2 Sample Geography Scheme

1381 Note that in the following examples, the ID attributes have been chosen for ease  
 1382 of readability and are therefore not valid URN or UUID values.

```
1383 <ClassificationScheme id='Geography-id' name="Geography"/>
1384 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
1385 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
1386 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
1387 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
1388 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

1392

## 1393 10.3 Class Classification

### 1394 Base Classes:

1395 [RegistryObject](#)

1396

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

1402

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

1407

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

### 1411 10.3.1 Attribute Summary

1412

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

1413 Note that attributes inherited from the base classes of this class are not shown.  
1414

### 1415 **10.3.2 Attribute classificationScheme**

1416 If the Classification instance represents an external classification, then the  
1417 classificationScheme attribute is required. The classificationScheme value must  
1418 reference a ClassificationScheme instance.  
1419

### 1420 **10.3.3 Attribute classificationNode**

1421 If the Classification instance represents an internal classification, then the  
1422 classificationNode attribute is required. The classificationNode value must  
1423 reference a ClassificationNode instance.

### 1424 **10.3.4 Attribute classifiedObject**

1425 For both internal and external classifications, the ClassifiedObject attribute is  
1426 required and it references the RegistryObject instance that is classified by this  
1427 Classification.  
1428

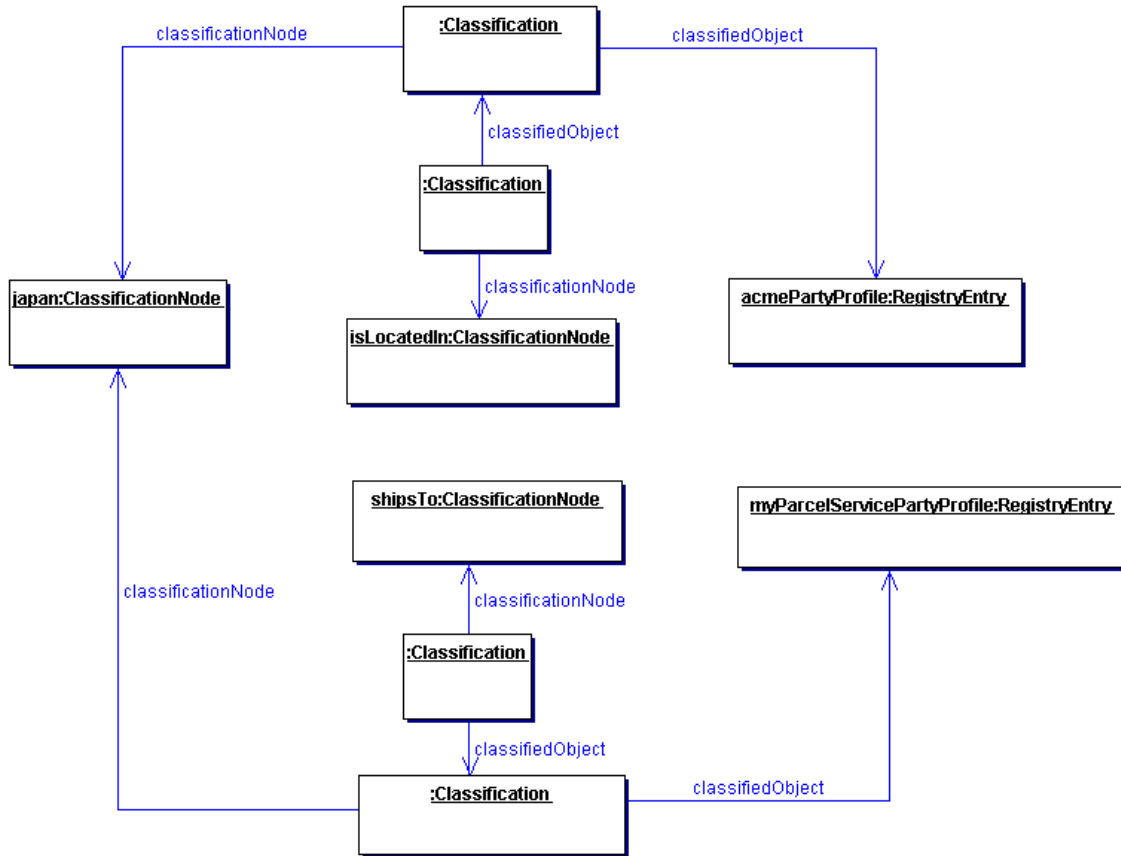
### 1429 **10.3.5 Attribute nodeRepresentation**

1430 If the Classification instance represents an external classification, then the  
1431 nodeRepresentation attribute is required. It is a representation of a taxonomy  
1432 element from a classification scheme. It is the responsibility of the registry to  
1433 distinguish between different types of nodeRepresentation, like between the  
1434 classification scheme node code and the classification scheme node canonical  
1435 path. This allows client to transparently use different syntaxes for  
1436 nodeRepresentation.

### 1437 **10.3.6 Context Sensitive Classification**

1438 Consider the case depicted in Figure 9 where a *Collaboration Protocol Profile* for  
1439 ACME Inc. is classified by the Japan ClassificationNode under the Geography  
1440 *Classification* scheme. In the absence of the context for this *Classification* its  
1441 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it  
1442 mean that ACME ships products to Japan, or does it have some other meaning?  
1443 To address this ambiguity a Classification may optionally be associated with  
1444 another ClassificationNode (in this example named isLocatedIn) that provides the  
1445 missing context for the Classification. Another *Collaboration Protocol Profile* for  
1446 MyParcelService may be classified by the Japan ClassificationNode where this

1447 Classification is associated with a different ClassificationNode (e.g., named  
 1448 shipsTo) to indicate a different context than the one used by ACME Inc.



1449

1450

**Figure 9: Context Sensitive Classification**

1451

1452

1453

1454 Thus, in order to support the possibility of Classification within multiple contexts,  
 1455 a Classification is itself classified by any number of Classifications that bind the  
 1456 first Classification to ClassificationNodes that provide the missing contexts.

1457

1458 In summary, the generalized support for *Classification* schemes in the  
 1459 information model allows:

1460

1461

- A RegistryObject to be classified by defining an internal Classification that associates it with a ClassificationNode in a *ClassificationScheme*.

1462

1463

- A RegistryObject to be classified by defining an external Classification that associates it with a value in an external *ClassificationScheme*.

1464

1465

- A RegistryObject to be classified along multiple facets by having multiple *Classifications* that associate it with multiple ClassificationNodes or value within a *ClassificationScheme*.

1466

1467

- A *Classification* defined for a RegistryObject to be qualified by the contexts in which it is being classified.

1468

1469  
1470

### 1471 **10.3.7 Method Summary**

1472 In addition to its attributes, the Classification class also defines the following  
1473 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>

1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485

### 1486 **10.4 Example of Classification Schemes**

1487 The following table lists some examples of possible *Classification* schemes  
1488 enabled by the information model. These schemes are based on a subset of  
1489 contextual concepts identified by the ebXML Business Process and Core  
1490 Components Project Teams. This list is meant to be illustrative not prescriptive.  
1491

1492

<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"	

1493

Table 1: Sample *Classification Schemes*

1494

1495

1496

1497

1498

1499

1500

## 1501 **11 Information Model: Security View**

1502 This section describes the aspects of the information model that relate to the  
 1503 security features of the *Registry*.

1504

1505 Figure 10 shows the view of the objects in the *Registry* from a security  
 1506 perspective. It shows object relationships as a *UML Class* diagram. It does not  
 1507 show *Class* attributes or *Class* methods that will be described in subsequent  
 1508 sections. It is meant to be illustrative not prescriptive.

1509

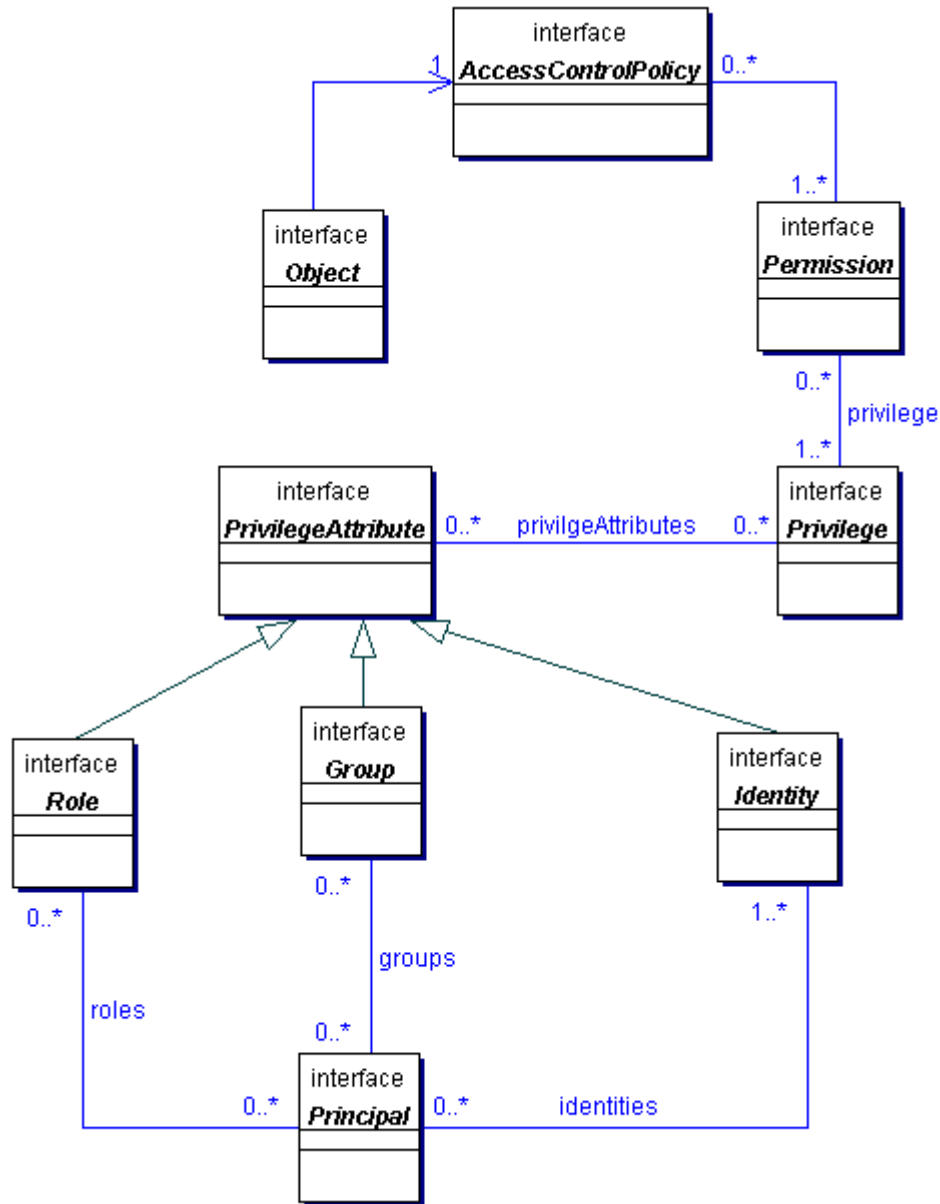


Figure 10: Information Model: Security View

1510  
 1511  
 1512  
 1513  
 1514  
 1515  
 1516  
 1517  
 1518  
 1519  
 1520

### 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.

1521

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

1522

## 1523 11.2 Class Permission

1524

1525 The Permission object is used for authorization and access control to  
 1526 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are  
 1527 defined in an AccessControlPolicy object.

1528

1529 A Permission object authorizes access to a method in a RegistryObject if the  
 1530 requesting Principal has any of the Privileges defined in the Permission.

1531

**See Also:**

1532

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

1533

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> .

1534

## 1535 11.3 Class Privilege

1536

1537 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute  
 1538 can be a Group, a Role, or an Identity.

1539

1540 A requesting Principal MUST have all of the PrivilegeAttributes specified in a  
 1541 Privilege in order to gain access to a method in a protected RegistryObject.  
 1542 Permissions defined in the RegistryObject's AccessControlPolicy define the  
 1543 Privileges that can authorize access to specific methods.

1544

1545 This mechanism enables the flexibility to have object access control policies that  
 1546 are based on any combination of Roles, Identities or Groups.

1547

**See Also:**

1548

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

1549

1550

1551

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

1552

## 1553 11.4 Class PrivilegeAttribute

1554 **All Known Subclasses:**

1555 [Group](#), [Identity](#), [Role](#)

1556

1557 PrivilegeAttribute is a common base *Class* for all types of security attributes that  
1558 are used to grant specific access control privileges to a Principal. A Principal may  
1559 have several different types of PrivilegeAttributes. Specific combination of  
1560 PrivilegeAttributes may be defined as a Privilege object.

1561 **See Also:**

1562 [Principal](#), [Privilege](#)

## 1563 11.5 Class Role

1564 **All Superclasses:**

1565 [PrivilegeAttribute](#)

1566

### 1567 11.5.1 A security Role PrivilegeAttribute

1568 For example a hospital may have *Roles* such as Nurse, Doctor, Administrator  
1569 etc. Roles are used to grant Privileges to Principals. For example a Doctor *Role*  
1570 may be allowed to write a prescription but a Nurse *Role* may not.

## 1571 11.6 Class Group

1572 **All Superclasses:**

1573 [PrivilegeAttribute](#)

1574

### 1575 11.6.1 A security Group PrivilegeAttribute

1576 A Group is an aggregation of users that may have different Roles. For example  
1577 a hospital may have a Group defined for Nurses and Doctors that are  
1578 participating in a specific clinical trial (e.g., AspirinTrial group). Groups are used  
1579 to grant Privileges to Principals. For example the members of the AspirinTrial  
1580 group may be allowed to write a prescription for Aspirin (even though Nurse Role  
1581 as a rule may not be allowed to write prescriptions).

1582

1583



1584 **11.7 Class Identity**1585 **All Superclasses:**1586 [PrivilegeAttribute](#)

1587

1588 **11.7.1 A security Identity PrivilegeAttribute**

1589 This is typically used to identify a person, an organization, or software service.  
 1590 Identity attribute may be in the form of a digital certificate.

1591 **11.8 Class Principal**

1592

1593 Principal is a generic term used by the security community to include both people  
 1594 and software systems. The Principal object is an entity that has a set of  
 1595 PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and  
 1596 optionally a set of role memberships, group memberships or security clearances.  
 1597 A principal is used to authenticate a requestor and to authorize the requested  
 1598 action based on the PrivilegeAttributes associated with the Principal.

1599 **See Also:**1600 PrivilegeAttributes, [Privilege](#), [Permission](#)

1601

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> .

1602

1603

## 1603 **12 References**

- 1604 [ebGLOSS] ebXML Glossary,  
1605 [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)
- 1606 [OAS] OASIS Information Model  
1607 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>
- 1608 [ISO] ISO 11179 Information Model  
1609 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 1611 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use  
1612 in RFCs to Indicate Requirement Levels  
1613 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>
- 1614 [ebRS] ebXML Registry Services Specification  
1615 <http://www.oasisopen.org/committees/regrep/documents/2.1/specs/ebRS.pdf>  
1616 [pdf](http://www.oasisopen.org/committees/regrep/documents/2.1/specs/ebRS.pdf)
- 1617 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification  
1618 <http://www.ebxml.org/specrafts/>  
1619
- 1620 [UUID] DCE 128 bit Universal Unique Identifier  
1621 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
1622 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>  
1623 [TR/REC-xml](http://www.w3.org/TR/REC-xml)  
1624
- 1625 [XPath] XML Path Language (XPath) Version 1.0  
1626 <http://www.w3.org/TR/xpath>  
1627
- 1628 [NCName] Namespaces in XML 19990114  
1629 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

## 1630 **13 Disclaimer**

- 1631 The views and specification expressed in this document are those of the authors  
1632 and are not necessarily those of their employers. The authors and their  
1633 employers specifically disclaim responsibility for any problems arising from  
1634 correct or incorrect implementation or use of this design.  
1635

**1635 14 Contact Information**

1636

1637 Team Leader

1638 Name: Lisa Carnahan

1639 Company: NIST

1640 Street: 100 Bureau Drive STOP 8970

1641 City, State, Postal Code: Gaithersburg, MD 20899-8970

1642 Country: USA

1643 Phone: (301) 975-3362

1644 Email: lisa.carnahan@nist.gov

1645

1646 Editor

1647 Name: Sally Fuger

1648 Company: Automotive Industry Action Group

1649 Street: 26200 Lahser Road, Suite 200

1650 City, State, Postal Code: Southfield, MI 48034

1651 Country: USA

1652 Phone: (248) 358-9744

1653 Email: sfuger@aiag.org

1654

1655 Technical Editor

1656 Name: Farrukh S. Najmi

1657 Company: Sun Microsystems

1658 Street: 1 Network Dr., MS BUR02-302

1659 City, State, Postal Code: Burlington, MA, 01803-0902

1660 Country: USA

1661 Phone: (781) 442-0703

1662 Email: najmi@east.sun.com

1663

1664

**1664 Copyright Statement**

1665 OASIS takes no position regarding the validity or scope of any intellectual  
1666 property or other rights that might be claimed to pertain to the implementation or  
1667 use of the technology described in this document or the extent to which any  
1668 license under such rights might or might not be available; neither does it  
1669 represent that it has made any effort to identify any such rights. Information on  
1670 OASIS's procedures with respect to rights in OASIS specifications can be found  
1671 at the OASIS website. Copies of claims of rights made available for publication  
1672 and any assurances of licenses to be made available, or the result of an attempt  
1673 made to obtain a general license or permission for the use of such proprietary  
1674 rights by implementors or users of this specification, can be obtained from the  
1675 OASIS Executive Director.

1676  
1677 OASIS invites any interested party to bring to its attention any copyrights, patents  
1678 or patent applications, or other proprietary rights which may cover technology  
1679 that may be required to implement this specification. Please address the  
1680 information to the OASIS Executive Director.

1681  
1682 Copyright ©The Organization for the Advancement of Structured Information  
1683 Standards [OASIS] 2002. All Rights Reserved.

1684 This document and translations of it may be copied and furnished to others, and  
1685 derivative works that comment on or otherwise explain it or assist in its  
1686 implementation may be prepared, copied, published and distributed, in whole or  
1687 in part, without restriction of any kind, provided that the above copyright notice  
1688 and this paragraph are included on all such copies and derivative works.

1689 However, this document itself may not be modified in any way, such as by  
1690 removing the copyright notice or references to OASIS, except as needed for the  
1691 purpose of developing OASIS specifications, in which case the procedures for  
1692 copyrights defined in the OASIS Intellectual Property Rights document must be  
1693 followed, or as required to translate it into languages other than English.

1694 The limited permissions granted above are perpetual and will not be revoked by  
1695 OASIS or its successors or assigns.

1696 This document and the information contained herein is provided on an "AS IS"  
1697 basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
1698 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
1699 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
1700 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
1701 PURPOSE."