



Creating A Single Global Electronic Market

1
2
3
4

OASIS/ebXML Registry Information Model v2.0

Approved Committee Specification

OASIS/ebXML Registry Technical Committee

18 December 2001

9

1 Status of this Document

Distribution of this document is unlimited.

This version:

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

Latest version:

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

20

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

21 This document, in its current form, is an approved Committee Specification of the
22 OASIS ebXML Registry Technical Committee. It builds upon version 1.0 which
23 was approved by the OASIS/ebXML Registry Technical Committee as a DRAFT
24 Specification of the TC.

25

26 At the time of that approval the following were members of the OASIS/ebXML
27 Registry Technical Committee:

28

29 Kathryn Breininger, Boeing
30 Lisa Carnahan, US NIST (TC Chair)
31 Joseph M. Chiusano, LMI
32 Suresh Damodaran, Sterling Commerce
33 Mike DeNicola Fujitsu
34 Anne Fischer, Drummond Group
35 Sally Fuger, AIAG
36 Jong Kim InnoDigital
37 Kyu-Chul Lee, Chungnam National University
38 Joel Munter, Intel
39 Farrukh Najmi, Sun Microsystems
40 Joel Neu, Vitria Technologies
41 Sanjay Patil, IONA
42 Neal Smith, ChevronTexaco
43 Nikola Stojanovic, Encoda Systems, Inc.
44 Prasad Yendluri, webMethods
45 Yutaka Yoshida, Sun Microsystems

46

47 **2.1 Contributors**

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

50

51 Len Gallagher, NIST
52 Sekhar Vajjhala, Sun Microsystems

53

54

54	Table of Contents	
55		
56	1 STATUS OF THIS DOCUMENT	1
57	2 OASIS/EBXML REGISTRY TECHNICAL COMMITTEE	2
58	2.1 CONTRIBUTORS	2
59	3 INTRODUCTION.....	8
60	3.1 SUMMARY OF CONTENTS OF DOCUMENT	8
61	3.2 GENERAL CONVENTIONS.....	8
62	3.2.1 <i>Naming Conventions</i>	8
63	3.3 AUDIENCE	9
64	3.4 RELATED DOCUMENTS	9
65	4 DESIGN OBJECTIVES	9
66	4.1 GOALS.....	9
67	5 SYSTEM OVERVIEW.....	10
68	5.1 ROLE OF EBXML <i>REGISTRY</i>	10
69	5.2 REGISTRY SERVICES	10
70	5.3 WHAT THE REGISTRY INFORMATION MODEL DOES	10
71	5.4 HOW THE REGISTRY INFORMATION MODEL WORKS	10
72	5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED	10
73	5.6 CONFORMANCE TO AN EBXML <i>REGISTRY</i>	11
74	6 REGISTRY INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW	11
75	6.1 REGISTRYOBJECT.....	12
76	6.2 SLOT	12
77	6.3 ASSOCIATION	12
78	6.4 EXTERNALIDENTIFIER	12
79	6.5 EXTERNALLINK.....	12
80	6.6 CLASSIFICATIONSCHEME	12
81	6.7 CLASSIFICATIONNODE	13
82	6.8 CLASSIFICATION.....	13
83	6.9 REGISTRYPACKAGE	13
84	6.10 AUDITABLEEVENT	13
85	6.11 USER	13
86	6.12 POSTALADDRESS	13
87	6.13 EMAILADDRESS	13
88	6.14 ORGANIZATION	14
89	6.15 SERVICE	14
90	6.16 SERVICEBINDING	14
91	6.17 SPECIFICATIONLINK.....	14

92	7	REGISTRY INFORMATION MODEL: DETAIL VIEW	14
93	7.1	ATTRIBUTE AND METHODS OF INFORMATION MODEL CLASSES	15
94	7.2	DATA TYPES	16
95	7.3	INTERNATIONALIZATION (I18N) SUPPORT	16
96	7.3.1	Class <i>InternationalString</i>	16
97	7.3.2	Class <i>LocalizedString</i>	17
98	7.4	CLASS REGISTRYOBJECT	17
99	7.4.1	Attribute <i>Summary</i>	17
100	7.4.2	Attribute <i>accessControlPolicy</i>	18
101	7.4.3	Attribute <i>description</i>	18
102	7.4.4	Attribute <i>id</i>	18
103	7.4.5	Attribute <i>name</i>	18
104	7.4.6	Attribute <i>objectType</i>	18
105	7.4.7	Method <i>Summary</i>	20
106	7.5	CLASS REGISTRYENTRY	20
107	7.5.1	Attribute <i>Summary</i>	21
108	7.5.2	Attribute <i>expiration</i>	21
109	7.5.3	Attribute <i>majorVersion</i>	21
110	7.5.4	Attribute <i>minorVersion</i>	21
111	7.5.5	Attribute <i>stability</i>	22
112	7.5.6	Attribute <i>status</i>	22
113	7.5.7	Attribute <i>userVersion</i>	23
114	7.5.8	Method <i>Summary</i>	23
115	7.6	CLASS SLOT	23
116	7.6.1	Attribute <i>Summary</i>	23
117	7.6.2	Attribute <i>name</i>	24
118	7.6.3	Attribute <i>slotType</i>	24
119	7.6.4	Attribute <i>values</i>	24
120	7.7	CLASS EXTRINSICOBJECT	24
121	7.7.1	Attribute <i>Summary</i>	24
122	7.7.2	Attribute <i>isOpaque</i>	25
123	7.7.3	Attribute <i>mimeType</i>	25
124	7.8	CLASS REGISTRYPACKAGE	25
125	7.8.1	Attribute <i>Summary</i>	25
126	7.8.2	Method <i>Summary</i>	25
127	7.9	CLASS EXTERNALIDENTIFIER	25
128	7.9.1	Attribute <i>Summary</i>	26
129	7.9.2	Attribute <i>identificationScheme</i>	26
130	7.9.3	Attribute <i>registryObject</i>	26
131	7.9.4	Attribute <i>value</i>	26
132	7.10	CLASS EXTERNALLINK	26
133	7.10.1	Attribute <i>Summary</i>	26
134	7.10.2	Attribute <i>externalURI</i>	27
135	7.10.3	Method <i>Summary</i>	27
136	8	REGISTRY AUDIT TRAIL	27

137	8.1	CLASS AUDITABLEEVENT	27
138	8.1.1	Attribute Summary.....	28
139	8.1.2	Attribute eventType	28
140	8.1.3	Attribute registryObject.....	28
141	8.1.4	Attribute timestamp	28
142	8.1.5	Attribute user.....	28
143	8.2	CLASS USER.....	29
144	8.2.1	Attribute Summary.....	29
145	8.2.2	Attribute address	29
146	8.2.3	Attribute emailAddresses.....	29
147	8.2.4	Attribute organization	29
148	8.2.5	Attribute personName	29
149	8.2.6	Attribute telephoneNumbers.....	30
150	8.2.7	Attribute url.....	30
151	8.3	CLASS ORGANIZATION	30
152	8.3.1	Attribute Summary.....	30
153	8.3.2	Attribute address	30
154	8.3.3	Attribute parent	30
155	8.3.4	Attribute primaryContact	30
156	8.3.5	Attribute telephoneNumbers.....	30
157	8.4	CLASS POSTALADDRESS	31
158	8.4.1	Attribute Summary.....	31
159	8.4.2	Attribute city.....	31
160	8.4.3	Attribute country	31
161	8.4.4	Attribute postalCode	31
162	8.4.5	Attribute state	31
163	8.4.6	Attribute street.....	31
164	8.4.7	Attribute streetNumber.....	31
165	8.4.8	Method Summary.....	32
166	8.5	CLASS TELEPHONENUMBER.....	32
167	8.5.1	Attribute Summary.....	32
168	8.5.2	Attribute areaCode.....	32
169	8.5.3	Attribute countryCode.....	32
170	8.5.4	Attribute extension.....	32
171	8.5.5	Attribute number	33
172	8.5.6	Attribute phoneType	33
173	8.6	CLASS EMAILADDRESS	33
174	8.6.1	Attribute Summary.....	33
175	8.6.2	Attribute address	33
176	8.6.3	Attribute type.....	33
177	8.7	CLASS PERSONNAME	33
178	8.7.1	Attribute Summary.....	33
179	8.7.2	Attribute firstName.....	33
180	8.7.3	Attribute lastName.....	34
181	8.7.4	Attribute middleName	34
182	8.8	CLASS SERVICE.....	34

183	8.8.1	<i>Attribute Summary</i>	34
184	8.8.2	<i>Method Summary</i>	34
185	8.9	CLASS SERVICEBINDING	34
186	8.9.1	<i>Attribute Summary</i>	35
187	8.9.2	<i>Attribute accessURL</i>	35
188	8.9.3	<i>Attribute targetBinding</i>	35
189	8.9.4	<i>Method Summary</i>	35
190	8.10	CLASS SPECIFICATIONLINK.....	35
191	8.10.1	<i>Attribute Summary</i>	36
192	8.10.2	<i>Attribute specificationObject</i>	36
193	8.10.3	<i>Attribute usageDescription</i>	36
194	8.10.4	<i>Attribute usageParameters</i>	36
195	9	ASSOCIATION OF REGISTRY OBJECTS	37
196	9.1	EXAMPLE OF AN ASSOCIATION.....	37
197	9.2	SOURCE AND TARGET OBJECTS.....	37
198	9.3	ASSOCIATION TYPES	37
199	9.4	INTRAMURAL ASSOCIATION	38
200	9.5	EXTRAMURAL ASSOCIATION.....	38
201	9.6	CONFIRMATION OF AN ASSOCIATION	39
202	9.6.1	<i>Confirmation of Intramural Associations</i>	39
203	9.6.2	<i>Confirmation of Extramural Associations</i>	40
204	9.7	VISIBILITY OF UNCONFIRMED ASSOCIATIONS	40
205	9.8	POSSIBLE CONFIRMATION STATES	40
206	9.9	CLASS ASSOCIATION	40
207	9.9.1	<i>Attribute Summary</i>	41
208	9.9.2	<i>Attribute associationType</i>	41
209	9.9.3	<i>Attribute sourceObject</i>	42
210	9.9.4	<i>Attribute targetObject</i>	42
211	10	CLASSIFICATION OF REGISTRYOBJECT	43
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>Method Summary</i>	47
221	10.2.5	<i>Canonical Path Syntax</i>	48
222	10.3	CLASS CLASSIFICATION	49
223	10.3.1	<i>Attribute Summary</i>	49
224	10.3.2	<i>Attribute classificationScheme</i>	50
225	10.3.3	<i>Attribute classificationNode</i>	50
226	10.3.4	<i>Attribute classifiedObject</i>	50

227	10.3.5	<i>Attribute nodeRepresentation</i>	50
228	10.3.6	<i>Context Sensitive Classification</i>	50
229	10.3.7	<i>Method Summary</i>	52
230	10.4	EXAMPLE OF <i>CLASSIFICATION</i> SCHEMES	53
231	11	INFORMATION MODEL: SECURITY VIEW	53
232	11.1	CLASS ACCESSCONTROLPOLICY.....	54
233	11.2	CLASS PERMISSION	55
234	11.3	CLASS PRIVILEGE.....	55
235	11.4	CLASS PRIVILEGEATTRIBUTE.....	56
236	11.5	CLASS ROLE.....	56
237	11.5.1	<i>A security Role PrivilegeAttribute</i>	56
238	11.6	CLASS GROUP	56
239	11.6.1	<i>A security Group PrivilegeAttribute</i>	56
240	11.7	CLASS IDENTITY.....	57
241	11.7.1	<i>A security Identity PrivilegeAttribute</i>	57
242	11.8	CLASS PRINCIPAL.....	57
243	12	REFERENCES	58
244	13	DISCLAIMER	58
245	14	CONTACT INFORMATION	59
246		COPYRIGHT STATEMENT	60
247		Table of Figures	
248		Figure 1: Information Model High Level Public View	11
249		Figure 2: Information Model <i>Inheritance</i> View	15
250		Figure 3: Example of RegistryObject Association	37
251		Figure 4: Example of Intramural Association.....	38
252		Figure 5: Example of Extramural Association.....	39
253		Figure 6: Example showing a <i>Classification</i> Tree.....	44
254		Figure 7: Information Model <i>Classification</i> View.....	45
255		Figure 8: Classification <i>Instance</i> Diagram.....	45
256		Figure 9: Context Sensitive <i>Classification</i>	51
257		Figure 10: Information Model: Security View.....	54
258		Table of Tables	
259		Table 1: Sample <i>Classification</i> Schemes	53
260			
261			

261 **3 Introduction**

262 **3.1 Summary of Contents of Document**

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

264

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

268 **3.2 General Conventions**

269 The following conventions are used throughout this document:

270

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

273

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

277

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

280

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

284

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

286

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

290

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

293 **3.2.1 Naming Conventions**

294

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

297 Capitalization styles are used in the following conventions:

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

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

307

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

309 **3.3 Audience**

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

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

314 **3.4 Related Documents**

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

317

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

323

324 **4 Design Objectives**

325 **4.1 Goals**

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

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

334

335 **5 System Overview**

336 **5.1 Role of ebXML Registry**

337

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

344 **5.2 Registry Services**

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

349 **5.3 What the Registry Information Model Does**

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

354 The Registry information model:

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

358 **5.4 How the Registry Information Model Works**

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

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

367

368 **5.5 Where the Registry Information Model May Be Implemented**

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

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

373 **5.6 Conformance to an ebXML Registry**

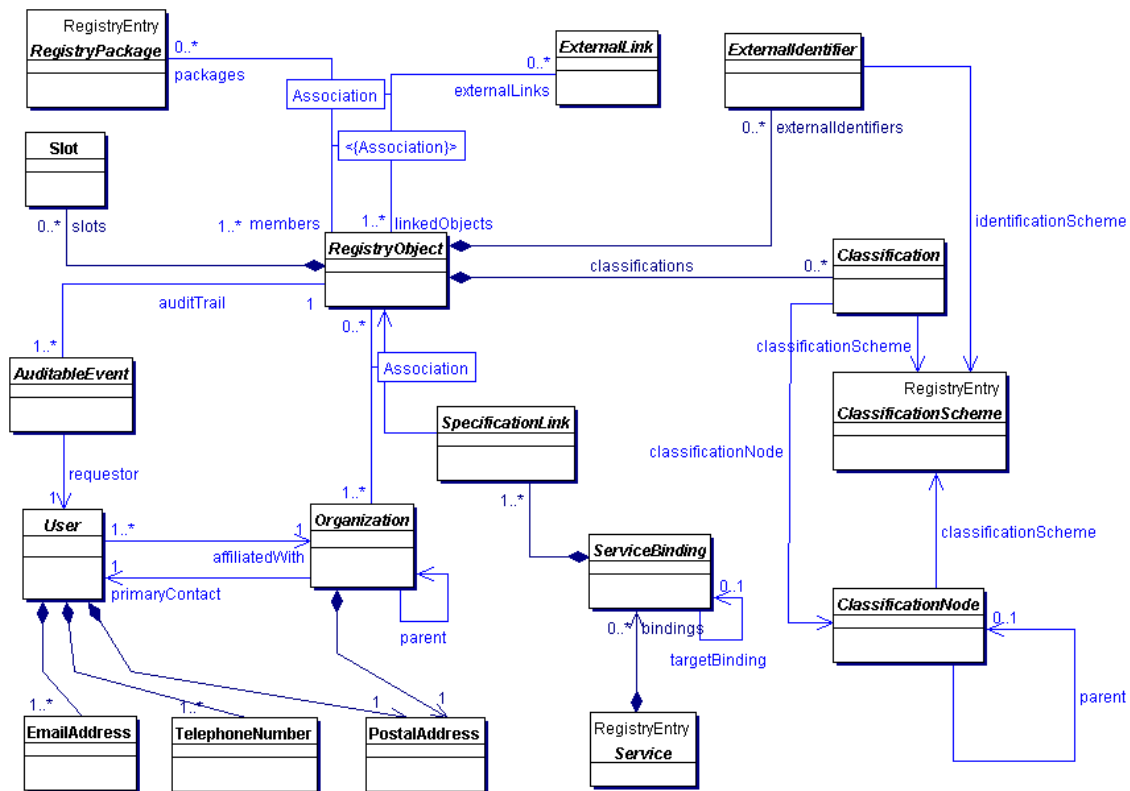
374 If an *Implementation* claims *Conformance* to this specification then it supports all
375 required information model *Classes* and interfaces, their attributes and their
376 semantic definitions that are visible through the ebXML *Registry Services*.

377 **6 Registry Information Model: High Level Public View**

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

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

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



387
388

Figure 1: Information Model High Level Public View

389 **6.1 RegistryObject**

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

394 **6.2 Slot**

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

401 **6.3 Association**

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

405 **6.4 ExternalIdentifier**

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

409 **6.5 ExternalLink**

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

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

421 **6.6 ClassificationScheme**

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

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

430 **6.7 ClassificationNode**

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

437 **6.8 Classification**

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

444 **6.9 RegistryPackage**

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

447 **6.10 AuditableEvent**

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

451 **6.11 User**

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

455 **6.12 PostalAddress**

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

458 **6.13 EmailAddress**

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

461 **6.14 Organization**

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

465 **6.15 Service**

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

468 **6.16 ServiceBinding**

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

473 **6.17 SpecificationLink**

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

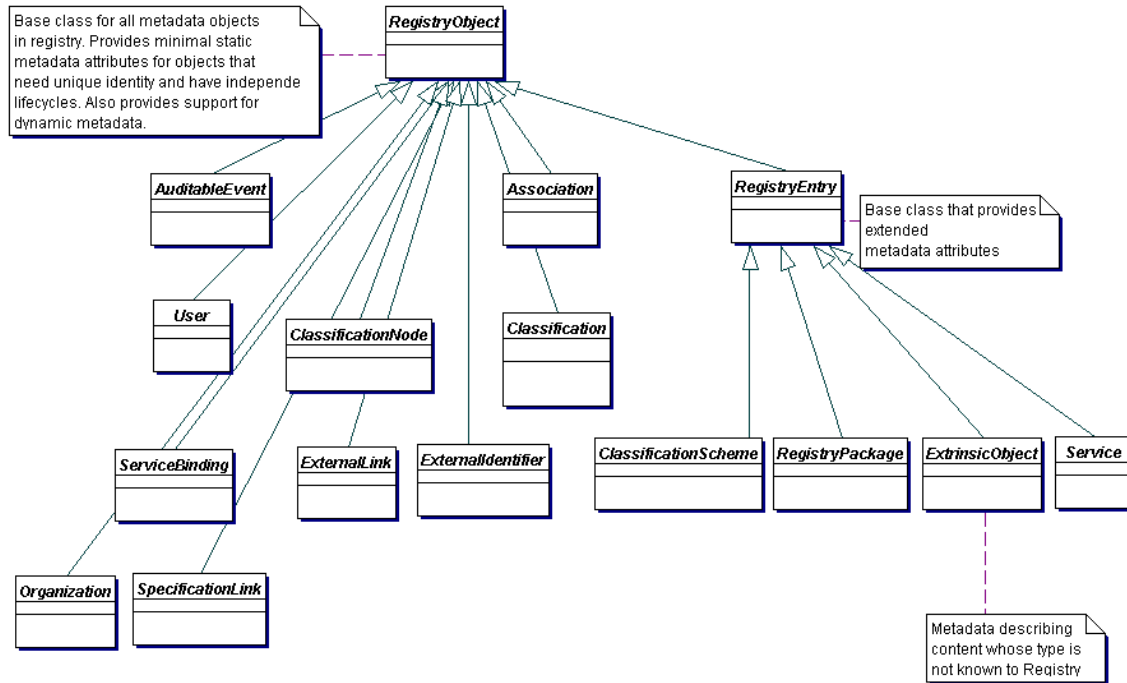
480 **7 Registry Information Model: Detail View**

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

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

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

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



498

499

500

Figure 2: Information Model *Inheritance View*

501

7.1 Attribute and Methods of Information Model Classes

502

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

506

507

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

511

512

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

513

514

515

515 7.2 Data Types

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

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
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	

519

520 7.3 Internationalization (I18N) Support

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

524

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

528 7.3.1 Class InternationalString

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

535 **7.3.2 Class LocalizedString**

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

540 **7.4 Class RegistryObject**

541 **Direct Known Subclasses:**

542 [Association](#), [AuditableEvent](#), [Classification](#), [ClassificationNode](#),
 543 [ExternalIdentifier](#), [ExternalLink](#), [Organization](#), [RegistryEntry](#), [User](#),
 544 [Service](#), [ServiceBinding](#), [SpecificationLink](#)

545 RegistryObject provides a common base class for almost all objects in the
 546 information model. Information model *Classes* whose instances have a unique
 547 identity are descendants of the RegistryObject *Class*.
 548

549 Note that Slot, PostalAddress, and a few other classes are not descendants of
 550 the RegistryObject Class because their instances do not have an independent
 551 existence and unique identity. They are always a part of some other Class's
 552 Instance (e.g., Organization has a PostalAddress).
 553

554 **7.4.1 Attribute Summary**

555 The following is the first of many tables that summarize the attributes of a class.
 556 The columns in the table are described as follows:
 557

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

558

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

559 **7.4.2 Attribute accessControlPolicy**

560 Each RegistryObject instance may have an accessControlPolicy instance
561 associated with it. An accessControlPolicy instance defines the *Security Model*
562 associated with the RegistryObject in terms of “who is permitted to do what” with
563 that RegistryObject.

564 **7.4.3 Attribute description**

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

568 **7.4.4 Attribute id**

569 Each RegistryObject instance must have a universally unique ID. Registry
570 objects use the id of other RegistryObject instances for the purpose of
571 referencing those objects.

572
573 Note that some classes in the information model do not have a need for a unique
574 id. Such classes do not inherit from RegistryObject class. Examples include
575 Entity classes such as TelephoneNumber, PostalAddress, EmailAddress and
576 PersonName.

577
578 All classes derived from RegistryObject have an id that is a Universally Unique ID
579 as defined by [UUID]. Such UUID based id attributes may be specified by the
580 client. If the UUID based id is not specified, then it must be generated by the
581 registry when a new RegistryObject instance is first submitted to the registry.

582 **7.4.5 Attribute name**

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

586 **7.4.6 Attribute objectType**

587 Each RegistryObject instance has an objectType. The objectType for almost all
588 objects in the information model is the name of their class. For example the
589 objectType for a Classification is “Classification”. The only exception to this rule
590 is that the objectType for an ExtrinsicObject instance is user defined and
591 indicates the type of repository item associated with the ExtrinsicObject.

592 **7.4.6.1 Pre-defined Object Types**

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

597
598

599 These pre-defined object types are defined as a *ClassificationScheme*. While the
 600 scheme may easily be extended a *Registry* MUST support the object types listed
 601 below.
 602

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

603

604 **7.4.7 Method Summary**

605 In addition to its attributes, the RegistryObject class also defines the following
 606 methods. These methods are used to navigate relationship links from a
 607 RegistryObject instance to other objects.
 608

Method Summary for RegistryObject	
Collection	getAssociations () Gets all Associations where this object is the source of the Association.
Collection	getAuditTrail () Gets the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.
Collection	getClassifications () Gets the Classification that classify this object.
Collection	getExternalIdentifiers () Gets the collection of ExternalIdentifiers associated with this object.
Collection	getExternalLinks () Gets the ExternalLinks associated with this object.
Collection	getOrganizations (String type) Gets the Organizations associated with this object. If a non-null type is specified it is used as a filter to match only specified type of organizations as indicated by the associationType attribute in the Association instance linking the object to the Organization.
Collection	getRegistryPackages () Gets the RegistryPackages that this object is a member of.
Collection	getSlots () Gets the Slots associated with this object.

609

610

611 **7.5 Class RegistryEntry**612 **Super Classes:**613 [RegistryObject](#)

614

615 **Direct Known Subclasses:**616 [ClassificationScheme](#), [ExtrinsicObject](#), [RegistryPackage](#)

617

618 RegistryEntry is a common base Class for classes in the information model that
 619 require additional metadata beyond the minimal metadata provided by
 620 RegistryObject class. RegistryEntry is used as a base class for high level coarse
 621 grained objects in the registry. Their life cycle typically requires more
 622 management (e.g. may require approval, deprecation). They typically have

623 relatively fewer instances but serve as a root of a composition hierarchy
 624 consisting of numerous objects that are sub-classes of RegistryObject but not
 625 RegistryEntry.

626
 627 The additional metadata is described by the attributes of the RegistryEntry class
 628 below.

629 **7.5.1 Attribute Summary**

630

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

631

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

634 **7.5.2 Attribute expiration**

635 Each RegistryEntry instance may have an expirationDate. This attribute defines a
 636 time limit upon the stability indication provided by the stability attribute. Once the
 637 expirationDate has been reached the stability attribute in effect becomes
 638 STABILITY_DYNAMIC implying that the repository item can change at any time
 639 and in any manner. A null value implies that there is no expiration on stability
 640 attribute.

641 **7.5.3 Attribute majorVersion**

642 Each RegistryEntry instance must have a major revision number for the current
 643 version of the RegistryEntry instance. This number is assigned by the registry
 644 when the object is created. This number may be updated by the registry when an
 645 object is updated.

646 **7.5.4 Attribute minorVersion**

647 Each RegistryEntry instance must have a minor revision number for the current
 648 version of the RegistryEntry instance. This number is assigned by the registry
 649 when the object is created. This number may be updated by the registry when an
 650 object is updated.

651 **7.5.5 Attribute stability**

652 Each RegistryEntry instance may have a stability indicator. The stability indicator
 653 is provided by the submitter as an indication of the level of stability for the
 654 repository item.

655 **7.5.5.1 Pre-defined RegistryEntry Stability Enumerations**

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

660

Name	Description
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.

661

662 **7.5.6 Attribute status**

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

665 **7.5.6.1 Pre-defined RegistryObject Status Types**

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

668

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> .

669

670 **7.5.7 Attribute userVersion**

671 Each RegistryEntry instance may have a userVersion. The userVersion is similar
 672 to the majorVersion-minorVersion tuple. They both provide an indication of the
 673 version of the object. The majorVersion-minorVersion tuple is provided by the
 674 registry while userVersion provides a user specified version for the object.
 675

676 **7.5.8 Method Summary**

677 In addition to its attributes, the RegistryEntry class also defines the following
 678 methods.

Method Summary for RegistryEntry	
Organization	<p>getSubmittingOrganization() Gets the Organization instance of the organization that submitted the given RegistryEntry instance. This method returns a non-null result for every RegistryEntry. For privilege assignment, the organization returned by this method is regarded as the owner of the RegistryEntry instance.</p>
Organization	<p>getResponsibleOrganization() Gets the Organization instance of the organization responsible for definition, approval, and/or maintenance of the repository item referenced by the given RegistryEntry instance. This method may return a null result if the submitting organization of this RegistryEntry does not identify a responsible organization or if the registration authority does not assign a responsible organization.</p>

679

680 **7.6 Class Slot**

681 Slot instances provide a dynamic way to add arbitrary attributes to
 682 RegistryObject instances. This ability to add attributes dynamically to
 683 RegistryObject instances enables extensibility within the information model.
 684

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

687 **7.6.1 Attribute Summary**

688

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

689

690 **7.6.2 Attribute name**

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

694 **7.6.3 Attribute slotType**

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

697 **7.6.4 Attribute values**

698 A Slot instance must have a Collection of values. The collection of values may be
699 empty. Since a Slot represent an extensible attribute whose value may be a
700 collection, therefore a Slot is allowed to have a collection of values rather than a
701 single value.
702

703 **7.7 Class ExtrinsicObject**

704 **Super Classes:**

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

706

707

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

711

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

715

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

718 **7.7.1 Attribute Summary**

719

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

720

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

723 **7.7.2 Attribute isOpaque**

724 Each ExtrinsicObject instance may have an isOpaque attribute defined. This
 725 attribute determines whether the content catalogued by this ExtrinsicObject is
 726 opaque to (not readable by) the *Registry*. In some situations, a *Submitting*
 727 *Organization* may submit content that is encrypted and not even readable by the
 728 *Registry*.

729 **7.7.3 Attribute mimeType**

730 Each ExtrinsicObject instance may have a mimeType attribute defined. The
 731 mimeType provides information on the type of repository item catalogued by the
 732 ExtrinsicObject instance.
 733

734 **7.8 Class RegistryPackage**

735 **Super Classes:**

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

737

738 RegistryPackage instances allow for grouping of logically related RegistryObject
 739 instances even if individual member objects belong to different Submitting
 740 Organizations.

741 **7.8.1 Attribute Summary**

742

743 The RegistryPackage class defines no new attributes other than those that are
 744 inherited from RegistryEntry and RegistryObject base classes. The inherited
 745 attributes are not shown here.

746 **7.8.2 Method Summary**

747 In addition to its attributes, the RegistryPackage class also defines the following
 748 methods.

749

Method Summary of RegistryPackage	
Collection	getMemberObjects() Get the collection of RegistryObject instances that are members of this RegistryPackage.

750

751 **7.9 Class ExternalIdentifier**

752 **Super Classes:**

753 [RegistryObject](#)

754

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

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

762 **7.9.1 Attribute Summary**

763

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

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

765 **7.9.2 Attribute identificationScheme**

766 Each ExternalIdentifier instance must have an identificationScheme attribute that
 767 references a ClassificationScheme. This ClassificationScheme defines the
 768 namespace within which an identifier is defined using the value attribute for the
 769 RegistryObject referenced by the RegistryObject attribute.

770 **7.9.3 Attribute registryObject**

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

773 **7.9.4 Attribute value**

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

776 **7.10 Class ExternalLink**

777 **Super Classes:**

778 [RegistryObject](#)

779

780 ExternalLinks use URIs to associate content in the *Registry* with content that may
 781 reside outside the *Registry*. For example, an organization submitting a *DTD*
 782 could use an ExternalLink to associate the *DTD* with the organization's home
 783 page.

784 **7.10.1 Attribute Summary**

785

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

786

787 **7.10.2 Attribute externalURI**

788 Each ExternalLink instance must have an externalURI attribute defined. The
 789 externalURI attribute provides a URI to the external resource pointed to by this
 790 ExternalLink instance. If the URI is a URL then a registry must validate the URL
 791 to be resolvable at the time of submission before accepting an ExternalLink
 792 submission to the registry.

793 **7.10.3 Method Summary**

794 In addition to its attributes, the ExternalLink class also defines the following
 795 methods.

796

Method Summary of ExternalLink	
Collection	getLinkedObjects() Gets the collection of RegistryObjects that are linked by this ExternalLink to content outside the registry.

797

798 **8 Registry Audit Trail**

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

803

804 The getAuditTrail() method of a RegistryObject returns an ordered Collection of
 805 AuditableEvents. These AuditableEvents constitute the audit trail for the
 806 RegistryObject. AuditableEvents include a timestamp for the *Event*. Each
 807 AuditableEvent has a reference to a User identifying the specific user that
 808 performed an action that resulted in an AuditableEvent. Each User is affiliated
 809 with an Organization, which is usually the *Submitting Organization*.

810 **8.1 Class AuditableEvent**

811 **Super Classes:**

812 [RegistryObject](#)

813

814 AuditableEvent instances provide a long-term record of *Events* that effect a
 815 change in a RegistryObject. A RegistryObject is associated with an ordered
 816 Collection of AuditableEvent instances that provide a complete audit trail for that
 817 RegistryObject.

818

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

821

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

824 RegistryObject. An AuditableEvent is created if and only if a request creates or
 825 alters the content or ownership of a RegistryObject. Read-only requests do not
 826 generate an AuditableEvent. No AuditableEvent is generated for a
 827 RegistryObject when it is classified, assigned to a RegistryPackage or associated
 828 with another RegistryObject.

829 **8.1.1 Attribute Summary**

830

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

831

832 **8.1.2 Attribute eventType**

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

835 **8.1.2.1 Pre-defined Auditable Event Types**

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

839

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.

840 **8.1.3 Attribute registryObject**

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

843 **8.1.4 Attribute timestamp**

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

846 **8.1.5 Attribute user**

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

849
850

851 **8.2 Class User**

852 **Super Classes:**

853 [RegistryObject](#)

854

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

857 **8.2.1 Attribute Summary**

858

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

859

860 **8.2.2 Attribute address**

861 Each User instance must have an address attribute that provides the postal
862 address for that user.

863 **8.2.3 Attribute emailAddresses**

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

867 **8.2.4 Attribute organization**

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

870 **8.2.5 Attribute personName**

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

873 **8.2.6 Attribute telephoneNumbers**

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

877 **8.2.7 Attribute url**

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

880 **8.3 Class Organization**

881 **Super Classes:**

882 [RegistryObject](#)

883

884 Organization instances provide information on organizations such as a
885 *Submitting Organization*. Each Organization *Instance* may have a reference to a
886 parent Organization.

887 **8.3.1 Attribute Summary**

888

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

889

890 **8.3.2 Attribute address**

891 Each Organization instance must have an address attribute that provides the
892 postal address for that organization.

893 **8.3.3 Attribute parent**

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

896 **8.3.4 Attribute primaryContact**

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

899 **8.3.5 Attribute telephoneNumbers**

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

902 number defined for that organization. An Organization must have at least one
903 telephone number.

904 **8.4 Class PostalAddress**

905 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal
906 address.

907 **8.4.1 Attribute Summary**

908

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

909

910 **8.4.2 Attribute city**

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

912 **8.4.3 Attribute country**

913 Each PostalAddress may have a country attribute identifying the country for that
914 address.

915 **8.4.4 Attribute postalCode**

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

918 **8.4.5 Attribute state**

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

921 **8.4.6 Attribute street**

922 Each PostalAddress may have a street attribute identifying the street name for
923 that address.

924 **8.4.7 Attribute streetNumber**

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

927 **8.4.8 Method Summary**

928 In addition to its attributes, the PostalAddress class also defines the following
929 methods.

930

Method Summary of ExternalLink	
Collection	<p>getSlots()</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>

931

932 **8.5 Class TelephoneNumber**

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

934 **8.5.1 Attribute Summary**

935

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

936

937 **8.5.2 Attribute areaCode**

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

940 **8.5.3 Attribute countryCode**

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

943 **8.5.4 Attribute extension**

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

946 **8.5.5 Attribute number**

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

950 **8.5.6 Attribute phoneType**

951 Each TelephoneNumber instance may have phoneType attribute that provides
 952 the type for the TelephoneNumber. Some examples of phoneType are “home”,
 953 “office”.

954 **8.6 Class EmailAddress**

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

956 **8.6.1 Attribute Summary**

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

957 **8.6.2 Attribute address**

958 Each EmailAddress instance must have an address attribute that provides the
 959 actual email address.

960 **8.6.3 Attribute type**

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

964 **8.7 Class PersonName**

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

966 **8.7.1 Attribute Summary**

967

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

968 **8.7.2 Attribute firstName**

969 Each PersonName may have a firstName attribute that is the first name of the
 970 person.

971 **8.7.3 Attribute lastName**

972 Each PersonName may have a lastName attribute that is the last name of the
973 person.

974 **8.7.4 Attribute middleName**

975 Each PersonName may have a middleName attribute that is the middle name of the
976 person.

977 **8.8 Class Service**

978 **Super Classes:**

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

980

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

982 **8.8.1 Attribute Summary**

983 The Service class does not define any specialized attributes other than its
984 inherited attributes.

985 **8.8.2 Method Summary**

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

987

Method Summary of Service	
Collection	getServiceBindings() Gets the collection of ServiceBinding instances defined for this Service.

988 **8.9 Class ServiceBinding**

989 **Super Classes:**

990 [RegistryObject](#)

991

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

995 The description attribute of ServiceBinding provides details about the relationship
996 between several specification links comprising the Service Binding. This
997 description can be useful for human understanding such that the runtime system
998 can be appropriately configured by the human being. There is possibility of
999 enforcing a structure on this description for enabling machine processing of the
1000 Service Binding, which is however not addressed by the current document.

1001

1002

1003 **8.9.1 Attribute Summary**

1004

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

1005

1006 **8.9.2 Attribute accessURI**

1007 A ServiceBinding may have an accessURI attribute that defines the URI to
 1008 access that ServiceBinding. This attribute is ignored if a targetBinding attribute is
 1009 specified for the ServiceBinding. If the URI is a URL then a registry must validate
 1010 the URL to be resolvable at the time of submission before accepting a
 1011 ServiceBinding submission to the registry.

1012 **8.9.3 Attribute targetBinding**

1013 A ServiceBinding may have a targetBinding attribute defined which references
 1014 another ServiceBinding. A targetBinding may be specified when a service is
 1015 being redirected to another service. This allows the rehosting of a service by
 1016 another service provider.

1017 **8.9.4 Method Summary**

1018 In addition to its attributes, the ServiceBinding class also defines the following
 1019 methods.

1020

Method Summary of ServiceBinding	
Collection	getSpecificationLinks () Get the collection of SpecificationLink instances defined for this ServiceBinding.

1021

1022

1023

1024 **8.10 Class SpecificationLink**

1025 **Super Classes:**

1026 [RegistryObject](#)

1027

1028 A SpecificationLink provides the linkage between a ServiceBinding and one of its
 1029 technical specifications that describes how to use the service using the
 1030 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink
 1031 instances that describe how to access the service using a technical specification
 1032 in form of a WSDL document or a CORBA IDL document.

1033 **8.10.1 Attribute Summary**

1034

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

1035

1036 **8.10.2 Attribute specificationObject**

1037 A SpecificationLink instance must have a specificationObject attribute that
 1038 provides a reference to a RegistryObject instance that provides a technical
 1039 specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject
 1040 instance representing the technical specification (e.g., a WSDL document).

1041 **8.10.3 Attribute usageDescription**

1042 A SpecificationLink instance may have a usageDescription attribute that provides
 1043 a textual description of how to use the optional usageParameters attribute
 1044 described next. The usageDescription is of type InternationalString, thus allowing
 1045 the description to be in multiple languages.

1046 **8.10.4 Attribute usageParameters**

1047 A SpecificationLink instance may have a usageParameters attribute that provides
 1048 a collection of Strings representing the instance specific parameters needed to
 1049 use the technical specification (e.g., a WSDL document) specified by this
 1050 SpecificationLink object.

1051

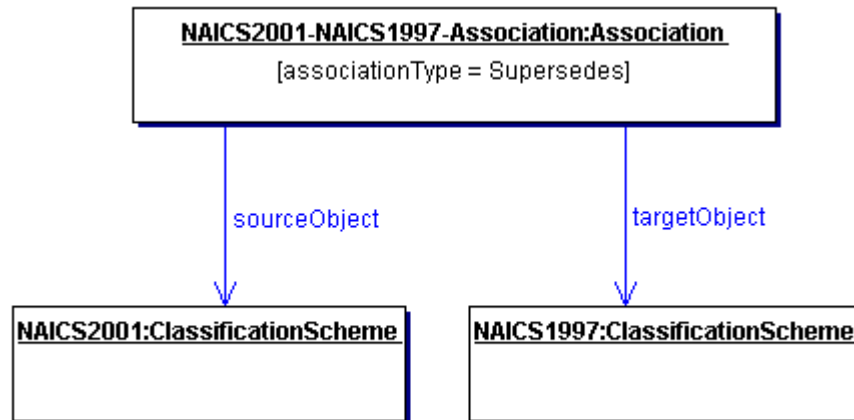
1051 9 Association of Registry Objects

1052 A RegistryObject instance may be *associated* with zero or more RegistryObject
 1053 instances. The information model defines an Association class, an instance of
 1054 which may be used to associate any two RegistryObject instances.

1055 9.1 Example of an Association

1056 One example of such an association is between two ClassificationScheme
 1057 instances, where one ClassificationScheme supersedes the other
 1058 ClassificationScheme as shown in Figure 3. This may be the case when a new
 1059 version of a ClassificationScheme is submitted.

1060 In Figure 3, we see how an Association is defined between a new version of the
 1061 NAICS ClassificationScheme and an older version of the NAICS
 1062 ClassificationScheme.
 1063



1064

1065

Figure 3: Example of RegistryObject Association

1066 9.2 Source and Target Objects

1067 An Association instance represents an association between a *source*
 1068 RegistryObject and a *target* RegistryObject. These are referred to as
 1069 *sourceObject* and *targetObject* for the Association instance. It is important which
 1070 object is the sourceObject and which is the targetObject as it determines the
 1071 directional semantics of an Association.

1072 In the example in Figure 3, it is important to make the newer version of NAICS
 1073 ClassificationScheme be the sourceObject and the older version of NAICS be the
 1074 targetObject because the associationType implies that the sourceObject
 1075 supersedes the targetObject (and not the other way around).

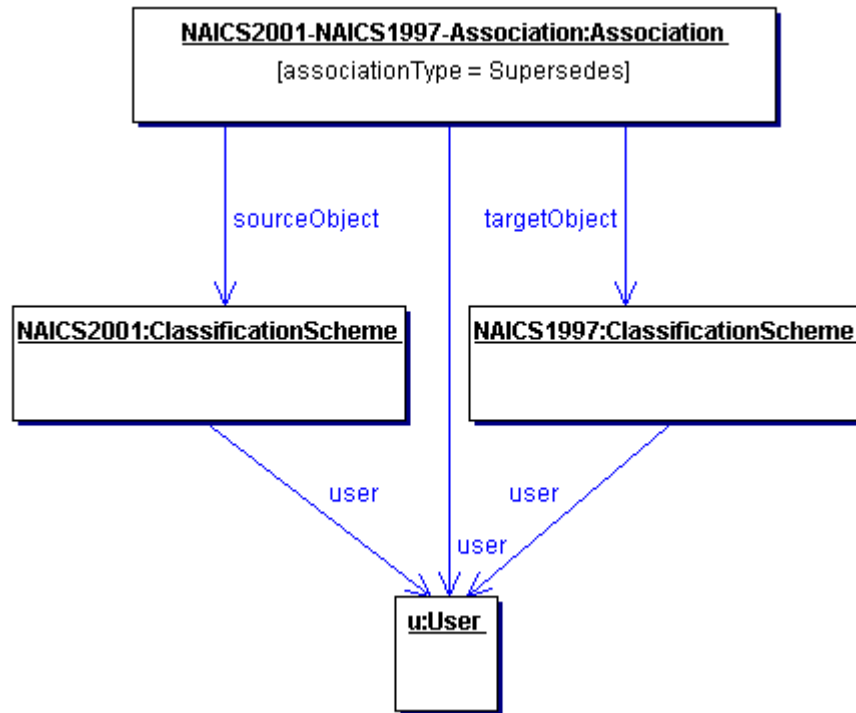
1076 9.3 Association Types

1077 Each Association must have an associationType attribute that identifies the type
 1078 of that association.

1079 **9.4 Intramural Association**

1080 A common use case for the Association class is when a User “u” creates an
 1081 Association “a” between two RegistryObjects “o1” and “o2” where association “a”
 1082 and RegistryObjects “o1” and “o2” are objects that were created by the same
 1083 User “u.” This is the simplest use case, where the association is between two
 1084 objects that are owned by the same User that is defining the Association. Such
 1085 associations are referred to as *intramural associations*.
 1086 Figure 4 below, extends the previous example in Figure 3 for the intramural
 1087 association case.

1088



1089

1090

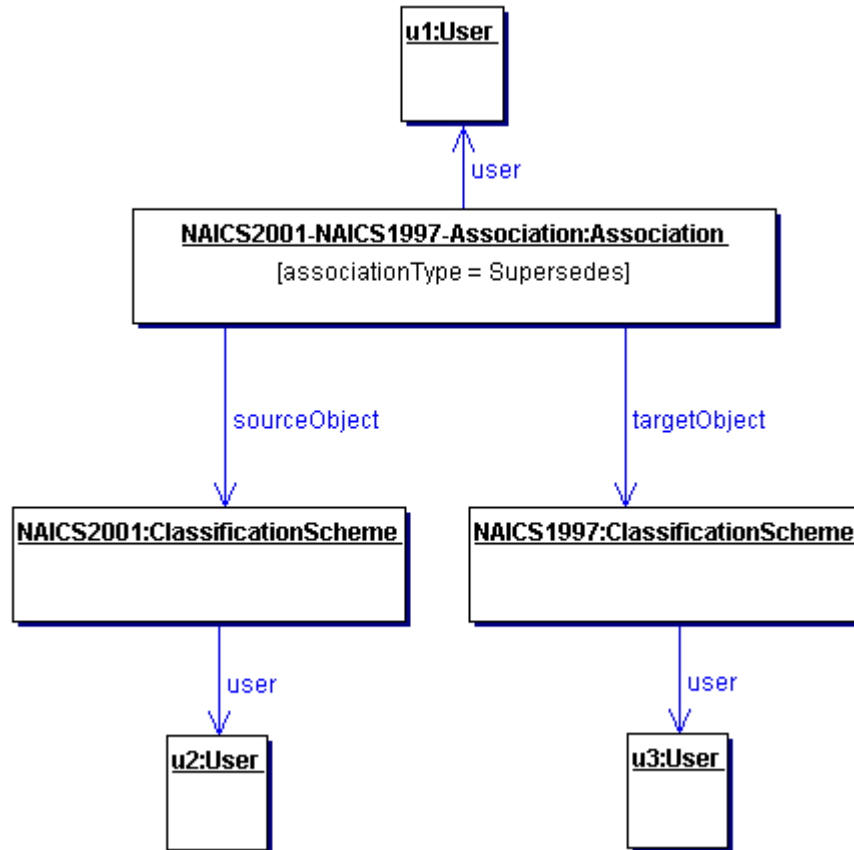
Figure 4: Example of Intramural Association

1091 **9.5 Extramural Association**

1092 The information model also allows more sophisticated use cases. For example, a
 1093 User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”
 1094 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2”
 1095 are owned by User “u2” and User “u3” respectively.

1096 In this use case an Association is defined where either or both objects that are
 1097 being associated are owned by a User different from the User defining the
 1098 Association. Such associations are referred to as *extramural associations*. The
 1099 Association class provides a convenience method called `isExtramural` that
 1100 returns "true" if the Association instance is an extramural Association.

1101 Figure 5 below, extends the previous example in Figure 3 for the extramural
 1102 association case. Note that it is possible for an extramural association to have
 1103 two distinct Users rather than three distinct Users as shown in Figure 5. In such
 1104 case, one of the two users owns two of the three objects involved (Association,
 1105 sourceObject and targetObject).
 1106



1107
 1108

Figure 5: Example of Extramural Association

1109 9.6 Confirmation of an Association

1110 An association may need to be confirmed by the parties whose objects are
 1111 involved in that Association as the sourceObject or targetObject. This section
 1112 describes the semantics of confirmation of an association by the parties involved.

1113 9.6.1 Confirmation of Intramural Associations

1114 Intramural associations may be viewed as declarations of truth and do not
 1115 require any explicit steps to confirm that Association as being true. In other
 1116 words, intramural associations are implicitly considered confirmed.

1117 **9.6.2 Confirmation of Extramural Associations**

1118 Extramural associations may be thought of as a unilateral assertion that may not
1119 be viewed as truth until it has been confirmed by the other (extramural) parties
1120 involved (Users “u2” and “u3” in the example in section 9.5).

1121 To confirm an extramural association, each of the extramural parties (parties that
1122 own the source or target object but do not own the Association) must submit an
1123 identical Association (clone Association) as the Association they are intending to
1124 confirm using a SubmitObjectsRequest. The clone Association must have the
1125 same id as the original Association.

1126 **9.7 Visibility of Unconfirmed Associations**

1127 Extramural associations require each extramural party to confirm the assertion
1128 being made by the extramural Association before the Association is visible to
1129 third parties that are not involved in the Association. This ensures that
1130 unconfirmed Associations are not visible to third party registry clients.

1131 **9.8 Possible Confirmation States**

1132 Assume the most general case where there are three distinct User instances as
1133 shown in Figure 5 for an extramural Association. The extramural Association
1134 needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3”
1135 in example) in order to be fully confirmed. The methods
1136 `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the
1137 Association class provide access to the confirmation state for both the
1138 `sourceObject` and `targetObject`. A third convenience method called
1139 `isConfirmed` provides a way to determine whether the Association is fully
1140 confirmed or not. So there are the following four possibilities related to the
1141 confirmation state of an extramural Association:

- 1142 ○ The Association is confirmed neither by the owner of the `sourceObject` nor
1143 by the owner of the `targetObject`.
- 1144 ○ The Association is confirmed by the owner of the `sourceObject` but it is not
1145 confirmed by the owner of the `targetObject`.
- 1146 ○ The Association is not confirmed by the owner of the `sourceObject` but it is
1147 confirmed by the owner of the `targetObject`.
- 1148 ○ The Association is confirmed by both the owner of the `sourceObject` and
1149 the owner of the `targetObject`. This is the only state where the Association
1150 is fully confirmed.

1151

1152 **9.9 Class Association**

1153 **Super Classes:**

1154 [RegistryObject](#)

1155

1156

1157 Association instances are used to define many-to-many associations among
1158 RegistryObjects in the information model.

1159

1160 An *Instance* of the Association Class represents an association between two
1161 RegistryObjects.

1162 9.9.1 Attribute Summary

1163

Attribute	Data Type	Required	Default Value	Specified By	Mutable
associationType	LongName	Yes		Client	No
sourceObject	UUID	Yes		Client	No
targetObject	UUID	Yes		Client	No

1164

1165 9.9.2 Attribute associationType

1166 Each Association must have an associationType attribute that identifies the type
1167 of that association.

1168 9.9.2.1 Pre-defined Association Types

1169 The following table lists pre-defined association types. These pre-defined
1170 association types are defined as a *Classification* scheme. While the scheme may
1171 easily be extended a *Registry* MUST support the association types listed below.

1172

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.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryObject object. Reserved for use in associating ExternalLinks with RegistryEntries.
Contains	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.
EquivalentTo	Defines that source RegistryObject is equivalent to the target RegistryObject.
Extends	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
Implements	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
InstanceOf	Defines that source RegistryObject is an <i>Instance</i> of

	target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.
SubmitterOf	Defines that the source Organization is the submitter of the target RegistryObject.
ResponsibleFor	Defines that the source Organization is responsible for the ongoing maintenance of the target RegistryObject.

1173

1174 **9.9.3 Attribute sourceObject**

1175 Each Association must have a sourceObject attribute that references the
 1176 RegistryObject instance that is the source of that association.

1177 **9.9.4 Attribute targetObject**

1178 Each Association must have a targetObject attribute that references the
 1179 RegistryObject instance that is the target of that association.

1180
 1181

Method Summary of Association	
boolean	<p><u>isConfirmed()</u> Returns true if isConfirmedBySourceOwner and isConfirmedByTargetOwner both return 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><u>isConfirmedBySourceOwner()</u> Returns true if the association has been confirmed by the owner of the sourceObject. For intramural Associations always return true.</p>
boolean	<p><u>isConfirmedByTargetOwner()</u> Returns true if the association has been confirmed by the owner of the targetObject. For intramural Associations always return true.</p>
boolean	<p><u>isExtramural()</u> 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>

1182

1183 **10 Classification of RegistryObject**

1184 This section describes the how the information model supports *Classification* of
1185 RegistryObject. It is a simplified version of the *OASIS* classification model [OAS].

1186

1187 A RegistryObject may be classified in many ways. For example the
1188 RegistryObject for the same *Collaboration Protocol Profile (CPP)* may be
1189 classified by its industry, by the products it sells and by its geographical location.

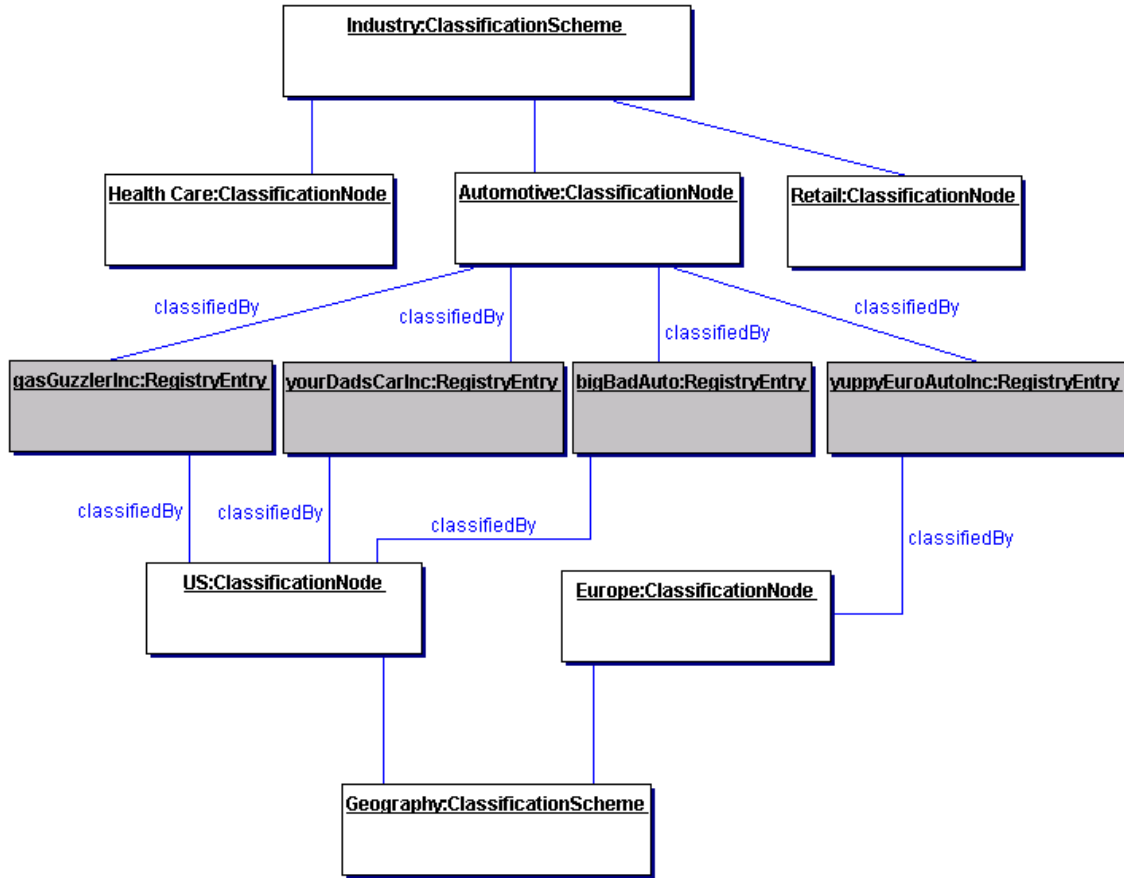
1190

1191 A general *ClassificationScheme* can be viewed as a *Classification* tree. In the
1192 example shown in Figure 6, RegistryObject instances representing *Collaboration*
1193 *Protocol Profiles* are shown as shaded boxes. Each *Collaboration Protocol*
1194 *Profile* represents an automobile manufacturer. Each *Collaboration Protocol*
1195 *Profile* is classified by the ClassificationNode named "Automotive" under the
1196 ClassificationScheme instance with name "Industry." Furthermore, the US
1197 Automobile manufacturers are classified by the US ClassificationNode under the
1198 ClassificationScheme with name "Geography." Similarly, a European automobile
1199 manufacturer is classified by the "Europe" ClassificationNode under the
1200 ClassificationScheme with name "Geography."

1201

1202 The example shows how a RegistryObject may be classified by multiple
1203 ClassificationNode instances under multiple ClassificationScheme instances
1204 (e.g., Industry, Geography).

1205



1206
1207

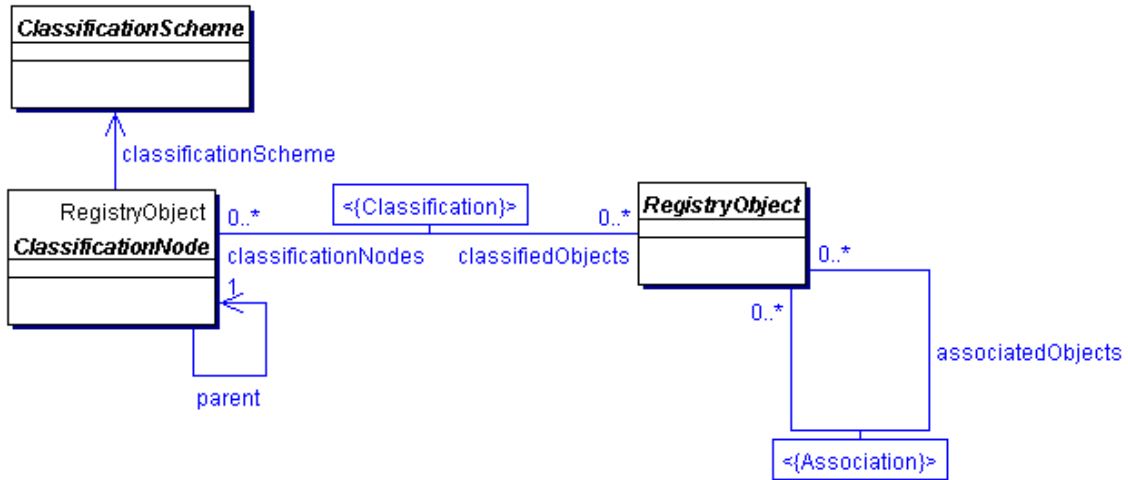
Figure 6: Example showing a *Classification Tree*

1208
1209
1210
1211
1212
1213
1214
1215
1216

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

1217
1218
1219

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.



1220

1221

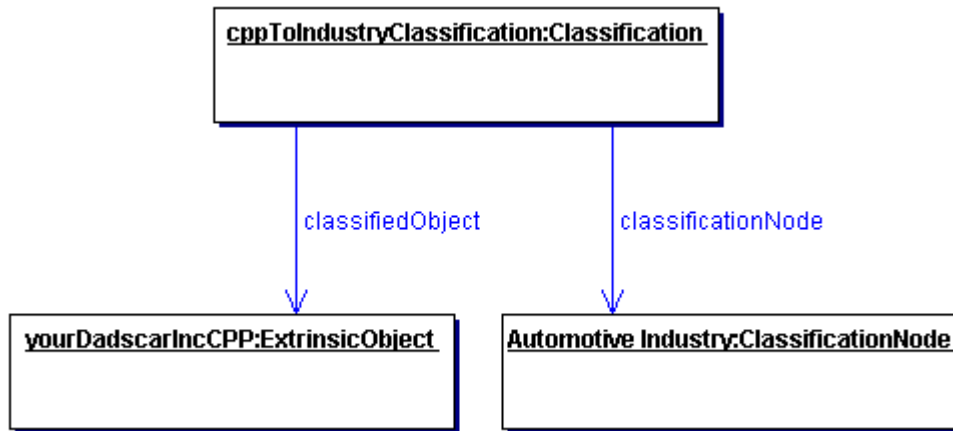
Figure 7: Information Model *Classification* View

1222

1223

1224 A Classification is somewhat like a specialized form of an Association. Figure 8
 1225 shows an example of an ExtrinsicObject *Instance* for a *Collaboration Protocol*
 1226 *Profile (CPP)* object that is classified by a *ClassificationNode* representing the
 1227 Industry that it belongs to.

1228



1229

1230

Figure 8: Classification *Instance* Diagram

1231

1232

1233

1234

1235

1236

1237 **10.1 Class ClassificationScheme**

1238 **Base classes:**

1239 [RegistryEntry](#), [RegistryObject](#)

1240

1241 A ClassificationScheme instance is metadata that describes a registered
1242 taxonomy. The taxonomy hierarchy may be defined internally to the
1243 Registry by instances of ClassificationNode or it may be defined externally
1244 to the Registry, in which case the structure and values of the taxonomy
1245 elements are not known to the Registry.

1246 In the first case the classification scheme is defined to be *internal* and in
1247 the second case the classification scheme is defined to be *external*.

1248 The ClassificationScheme class inherits attributes and methods from the
1249 RegistryObject and RegistryEntry classes.

1250

1251 **10.1.1 Attribute Summary**

1252

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

1253 Note that attributes inherited by ClassificationScheme class from the
1254 RegistryEntry class are not shown.

1255

1256 **10.1.2 Attribute isInternal**

1257 When submitting a ClassificationScheme instance the Submitting Organization
1258 needs to declare whether the ClassificationScheme instance represents an
1259 internal or an external taxonomy. This allows the registry to validate the
1260 subsequent submissions of ClassificationNode and Classification instances in
1261 order to maintain the type of ClassificationScheme consistent throughout its
1262 lifecycle.

1263

1264 **10.1.3 Attribute nodeType**

1265 When submitting a ClassificationScheme instance the Submitting Organization
1266 needs to declare what is the structure of taxonomy nodes that this
1267 ClassificationScheme instance will represent. This attribute is an enumeration
1268 with the following values:

- 1269 - UniqueCode. This value says that each node of the taxonomy has
1270 a unique code assigned to it.
- 1271 - EmbeddedPath. This value says that a unique code assigned to
1272 each node of the taxonomy at the same time encodes its path. This
1273 is the case in the NAICS taxonomy.

1274 - NonUniqueCode. In some cases nodes are not unique, and it is
 1275 necessary to nominate the full path in order to identify the node. For
 1276 example, in a geography taxonomy Moscow could be under both
 1277 Russia and the USA, where there are five cities of that name in
 1278 different states.

1279 This enumeration might expand in the future with some new values. An example
 1280 for possible future values for this enumeration might be NamedPathElements for
 1281 support of Named-Level taxonomies such as Genus/Species.
 1282

1283 10.2 Class ClassificationNode

1284 **Base classes:**

1285 [RegistryObject](#)

1286 ClassificationNode instances are used to define tree structures where
 1287 each node in the tree is a ClassificationNode. Such *Classification* trees
 1288 are constructed with ClassificationNode instances under a
 1289 ClassificationScheme instance, and are used to define *Classification*
 1290 schemes or ontologies.
 1291
 1292

1293 10.2.1 Attribute Summary

1294

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

1295

1296 10.2.2 Attribute parent

1297 Each ClassificationNode may have a parent attribute. The parent attribute either
 1298 references a parent ClassificationNode or a ClassificationScheme instance in
 1299 case of first level ClassificationNode instances.
 1300

1301 10.2.3 Attribute code

1302 Each ClassificationNode may have a code attribute. The code attribute contains
 1303 a code within a standard coding scheme.
 1304

1305 10.2.4 Method Summary

1306 In addition to its attributes, the ClassificationNode class also defines the following
 1307 methods.
 1308

Method Summary of ClassificationNode	
ClassificationScheme	getClassificationScheme() Get the ClassificationScheme that this ClassificationNode belongs to.
Collection	getClassifiedObjects() Get the collection of RegistryObjects classified by this ClassificationNode.
String	getPath() Gets the canonical path from the ClassificationScheme of this ClassificationNode. The path syntax is defined in 10.2.5.
Integer	getLevelNumber() 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.

1309

1310 In Figure 6, several instances of ClassificationNode are defined (all light colored
1311 boxes). A ClassificationNode has zero or one parent and zero or more
1312 ClassificationNodes for its immediate children. The parent of a
1313 ClassificationNode may be another ClassificationNode or a ClassificationScheme
1314 in case of first level ClassificationNodes.
1315

1316 10.2.5 Canonical Path Syntax

1317 The getPath method of the ClassificationNode class returns an absolute path in a
1318 canonical representation that uniquely identifies the path leading from the
1319 ClassificationScheme to that ClassificationNode.

1320 The canonical path representation is defined by the following BNF grammar:

1321

```

1322 canonicalPath ::= '/' schemeld nodePath
1323 nodePath     ::= '/' nodeCode
1324              | '/' nodeCode ( nodePath )?

```

1325

1326 In the above grammar, schemeld is the id attribute of the ClassificationScheme
1327 instance, and nodeCode is defined by NCName production as defined by
1328 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.
1329

1330 10.2.5.1 Example of Canonical Path Representation

1331 The following canonical path represents what the getPath method would return
1332 for the ClassificationNode with code 'United States' in the sample Geography
1333 scheme in section 10.2.5.2.

1334

```
1335 /Geography-id/NorthAmerica/UnitedStates
```


1336 **10.2.5.2 Sample Geography Scheme**

1337 Note that in the following examples, the ID attributes have been chosen for ease
 1338 of readability and are therefore not valid URN or UUID values.

```

1339
1340 <ClassificationScheme id='Geography-id' name="Geography"/>
1341
1342 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
1343 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
1344
1345 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
1346 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
1347 <ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
1348
    
```

1349 **10.3 Class Classification**

1350 **Base Classes:**
 1351 [RegistryObject](#)

1352 A Classification instance classifies a RegistryObject instance by referencing a
 1353 node defined within a particular classification scheme. An internal classification
 1354 will always reference the node directly, by its id, while an external classification
 1355 will reference the node indirectly by specifying a representation of its value that is
 1356 unique within the external classification scheme.
 1357

1358
 1359 The attributes and methods for the Classification class are intended to allow for
 1360 representation of both internal and external classifications in order to minimize
 1361 the need for a submission or a query to distinguish between internal and external
 1362 classifications.

1363
 1364 In Figure 6, Classification instances are not explicitly shown but are implied as
 1365 associations between the RegistryObject instances (shaded leaf node) and the
 1366 associated ClassificationNode.

1367 **10.3.1 Attribute Summary**

1368

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

1369 Note that attributes inherited from the base classes of this class are not shown.
 1370

1371 10.3.2 Attribute classificationScheme

1372 If the Classification instance represents an external classification, then the
1373 classificationScheme attribute is required. The classificationScheme value must
1374 reference a ClassificationScheme instance.
1375

1376 10.3.3 Attribute classificationNode

1377 If the Classification instance represents an internal classification, then the
1378 classificationNode attribute is required. The classificationNode value must
1379 reference a ClassificationNode instance.

1380 10.3.4 Attribute classifiedObject

1381 For both internal and external classifications, the ClassifiedObject attribute is
1382 required and it references the RegistryObject instance that is classified by this
1383 Classification.
1384

1385 10.3.5 Attribute nodeRepresentation

1386 If the Classification instance represents an external classification, then the
1387 nodeRepresentation attribute is required. It is a representation of a taxonomy
1388 element from a classification scheme. It is the responsibility of the registry to
1389 distinguish between different types of nodeRepresentation, like between the
1390 classification scheme node code and the classification scheme node canonical
1391 path. This allows client to transparently use different syntaxes for
1392 nodeRepresentation.

1393 10.3.6 Context Sensitive Classification

1394 Consider the case depicted in Figure 9 where a *Collaboration Protocol Profile* for
1395 ACME Inc. is classified by the Japan ClassificationNode under the Geography
1396 *Classification* scheme. In the absence of the context for this *Classification* its
1397 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it
1398 mean that ACME ships products to Japan, or does it have some other meaning?
1399 To address this ambiguity a Classification may optionally be associated with
1400 another ClassificationNode (in this example named isLocatedIn) that provides the
1401 missing context for the Classification. Another *Collaboration Protocol Profile* for
1402 MyParcelService may be classified by the Japan ClassificationNode where this
1403 Classification is associated with a different ClassificationNode (e.g., named
1404 shipsTo) to indicate a different context than the one used by ACME Inc.

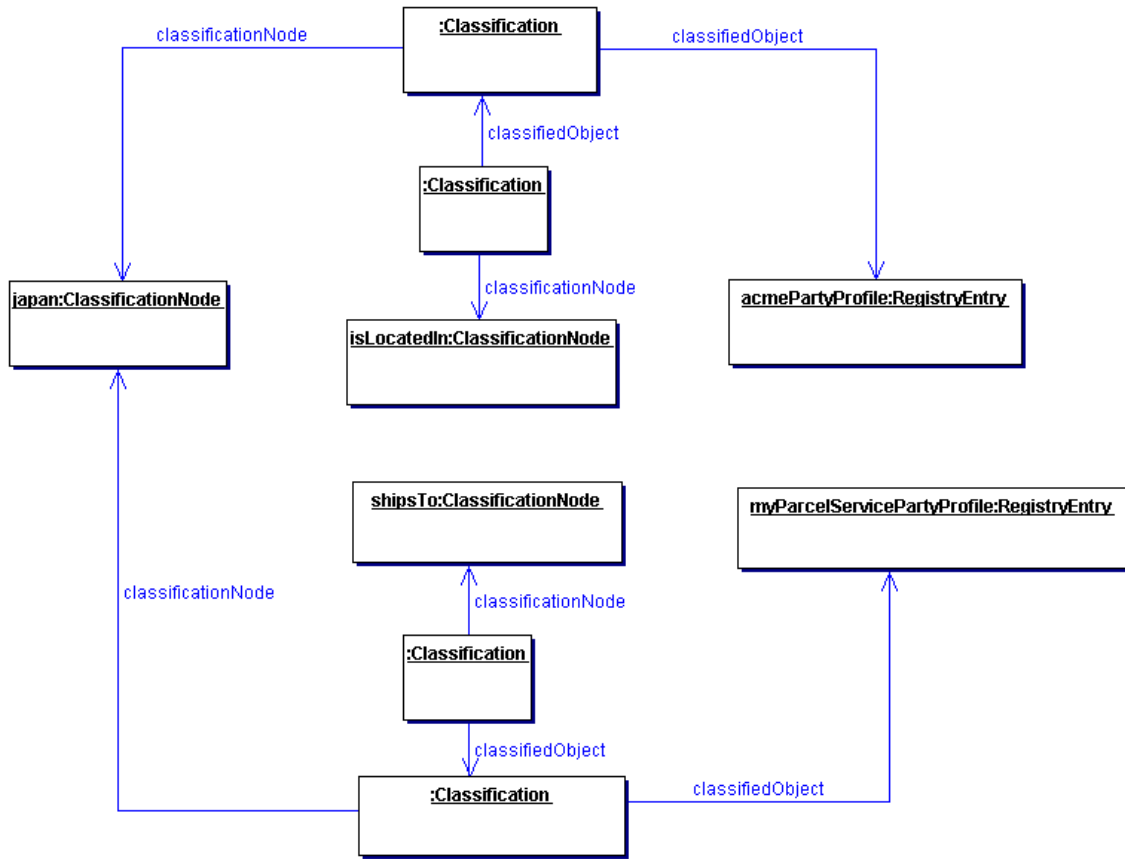


Figure 9: Context Sensitive Classification

1405

1406

1407

1408

1409

1410 Thus, in order to support the possibility of Classification within multiple contexts,
 1411 a Classification is itself classified by any number of Classifications that bind the
 1412 first Classification to ClassificationNodes that provide the missing contexts.

1413

1414 In summary, the generalized support for Classification schemes in the
 1415 information model allows:

- 1416 ○ A RegistryObject to be classified by defining an internal Classification that
 1417 associates it with a ClassificationNode in a ClassificationScheme.
- 1418 ○ A RegistryObject to be classified by defining an external Classification that
 1419 associates it with a value in an external ClassificationScheme.
- 1420 ○ A RegistryObject to be classified along multiple facets by having multiple
 1421 Classifications that associate it with multiple ClassificationNodes or value
 1422 within a ClassificationScheme.
- 1423 ○ A Classification defined for a RegistryObject to be qualified by the
 1424 contexts in which it is being classified.

1425

1426

1427 **10.3.7 Method Summary**

1428 In addition to its attributes, the Classification class also defines the following
 1429 methods:

Return Type	Method
UUID	<p>getClassificationScheme()</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>getPath()</p> <p>For an external classification returns a string that conforms to the string structure specified for the result of the getPath() method in the ClassificationNode class. For an internal classification, returns the same value as does the getPath() method applied to the ClassificationNode instance identified by the classificationNode attribute.</p>
ShortName	<p>getCode()</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>
Organization	<p>getSubmittingOrganization()</p> <p>Gets the Organization instance of the organization that submitted the given RegistryEntry instance. This method returns a non-null result for every RegistryEntry. For privilege assignment, the organization returned by this method is regarded as the owner of the Classification instance.</p>

1430
 1431
 1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441

1442 **10.4 Example of Classification Schemes**

1443 The following table lists some examples of possible *Classification* schemes
 1444 enabled by the information model. These schemes are based on a subset of
 1445 contextual concepts identified by the ebXML Business Process and Core
 1446 Components Project Teams. This list is meant to be illustrative not prescriptive.

1447
 1448

Classification Scheme	Usage Example	Standard Classification Schemes
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"	

1449

Table 1: Sample Classification Schemes

1450

1451

1452

1453

1454

1455

1456

1457 **11 Information Model: Security View**

1458 This section describes the aspects of the information model that relate to the
 1459 security features of the *Registry*.

1460

1461 Figure 10 shows the view of the objects in the *Registry* from a security
 1462 perspective. It shows object relationships as a *UML Class* diagram. It does not
 1463 show *Class* attributes or *Class* methods that will be described in subsequent
 1464 sections. It is meant to be illustrative not prescriptive.

1465

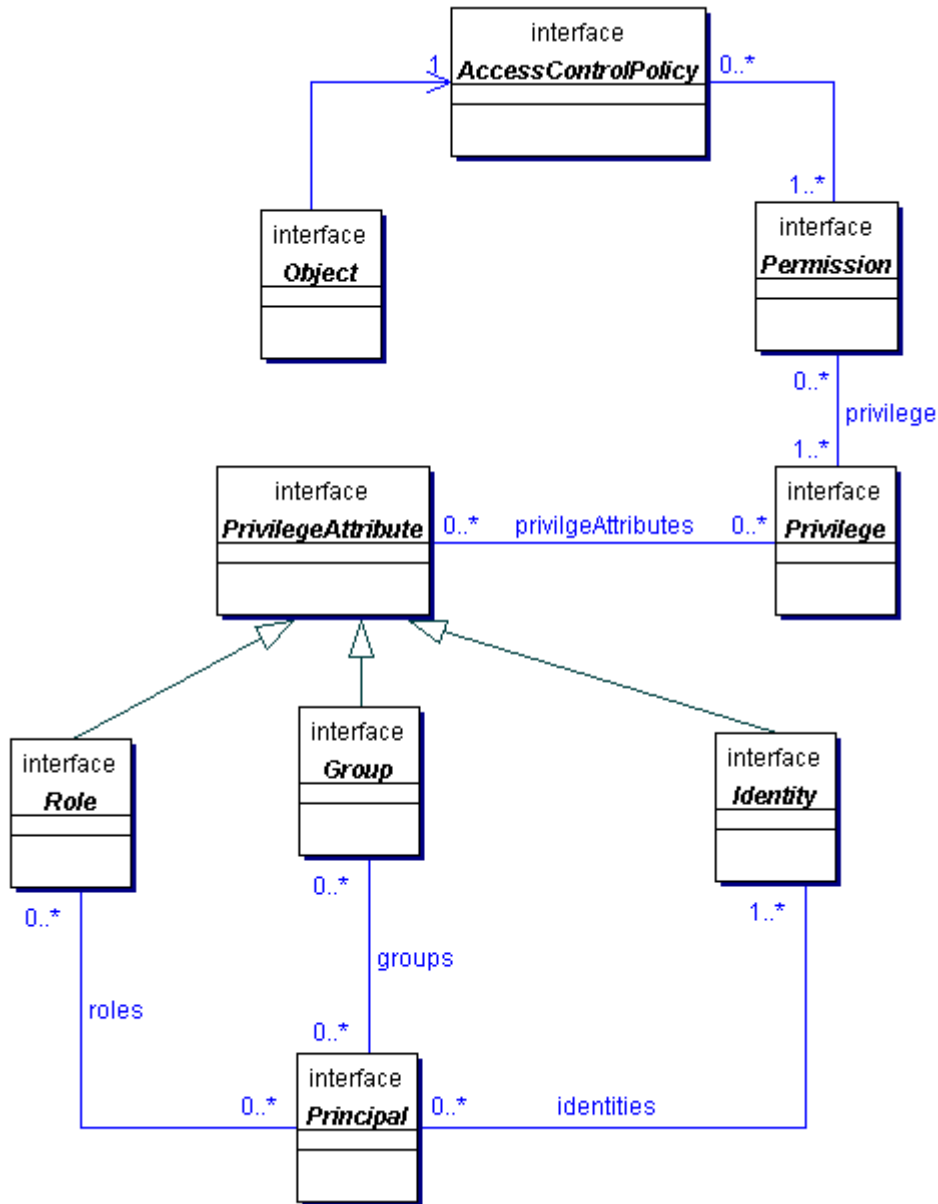


Figure 10: Information Model: Security View

1466
 1467
 1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476

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.

1477

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

1478

1479 11.2 Class Permission

1480

1481 The Permission object is used for authorization and access control to
 1482 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are
 1483 defined in an AccessControlPolicy object.

1484

1485 A Permission object authorizes access to a method in a RegistryObject if the
 1486 requesting Principal has any of the Privileges defined in the Permission.

1487 **See Also:**

1488 [Privilege](#), [AccessControlPolicy](#)

1489

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

1490

1491 11.3 Class Privilege

1492

1493 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute
 1494 can be a Group, a Role, or an Identity.

1495

1496 A requesting Principal MUST have all of the PrivilegeAttributes specified in a
 1497 Privilege in order to gain access to a method in a protected RegistryObject.
 1498 Permissions defined in the RegistryObject's AccessControlPolicy define the
 1499 Privileges that can authorize access to specific methods.

1500

1501 This mechanism enables the flexibility to have object access control policies that
 1502 are based on any combination of Roles, Identities or Groups.

1503 **See Also:**

1504 [PrivilegeAttribute](#), [Permission](#)

1505

1506

1507

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

1508

1509 11.4 Class PrivilegeAttribute

1510 **All Known Subclasses:**

1511 [Group](#), [Identity](#), [Role](#)

1512

1513 PrivilegeAttribute is a common base *Class* for all types of security attributes that
1514 are used to grant specific access control privileges to a Principal. A Principal may
1515 have several different types of PrivilegeAttributes. Specific combination of
1516 PrivilegeAttributes may be defined as a Privilege object.

1517 **See Also:**

1518 [Principal](#), [Privilege](#)

1519 11.5 Class Role

1520 **All Superclasses:**

1521 [PrivilegeAttribute](#)

1522

1523 11.5.1 A security Role PrivilegeAttribute

1524 For example a hospital may have *Roles* such as Nurse, Doctor, Administrator
1525 etc. Roles are used to grant Privileges to Principals. For example a Doctor *Role*
1526 may be allowed to write a prescription but a Nurse *Role* may not.

1527 11.6 Class Group

1528 **All Superclasses:**

1529 [PrivilegeAttribute](#)

1530

1531 11.6.1 A security Group PrivilegeAttribute

1532 A Group is an aggregation of users that may have different Roles. For example
1533 a hospital may have a Group defined for Nurses and Doctors that are
1534 participating in a specific clinical trial (e.g., AspirinTrial group). Groups are used
1535 to grant Privileges to Principals. For example the members of the AspirinTrial
1536 group may be allowed to write a prescription for Aspirin (even though Nurse Role
1537 as a rule may not be allowed to write prescriptions).

1538

1539

1540 **11.7 Class Identity**

1541 **All Superclasses:**
 1542 [PrivilegeAttribute](#)

1543

1544 **11.7.1 A security Identity PrivilegeAttribute**

1545 This is typically used to identify a person, an organization, or software service.
 1546 Identity attribute may be in the form of a digital certificate.

1547 **11.8 Class Principal**

1548 Principal is a generic term used by the security community to include both people
 1549 and software systems. The Principal object is an entity that has a set of
 1550 PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and
 1551 optionally a set of role memberships, group memberships or security clearances.
 1552 A principal is used to authenticate a requestor and to authorize the requested
 1553 action based on the PrivilegeAttributes associated with the Principal.

1554 **See Also:**
 1555 PrivilegeAttributes, [Privilege](#), [Permission](#)

1557

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

1558

1559

1559 **12 References**

- 1560 [ebGLOSS] ebXML Glossary,
1561 http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 1562 [OAS] OASIS Information Model
1563 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>
- 1564 [ISO] ISO 11179 Information Model
1565 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 1567 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use
1568 in RFCs to Indicate Requirement Levels
1569 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>
- 1570 [ebRS] ebXML Registry Services Specification
1571 <http://www.oasisopen.org/committees/regrep/documents/2.0/specs/ebRS.pdf>
1572 [pdf](http://www.oasisopen.org/committees/regrep/documents/2.0/specs/ebRS.pdf)
- 1573 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
1574 <http://www.ebxml.org/specrafts/>
1575
- 1576 [UUID] DCE 128 bit Universal Unique Identifier
1577 http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
1578 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
1579 [TR/REC-xml](http://www.w3.org/TR/REC-xml)
- 1580
- 1581 [XPath] XML Path Language (XPath) Version 1.0
1582 <http://www.w3.org/TR/xpath>
1583
- 1584 [NCName] Namespaces in XML 19990114
1585 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1586 **13 Disclaimer**

- 1587 The views and specification expressed in this document are those of the authors
1588 and are not necessarily those of their employers. The authors and their
1589 employers specifically disclaim responsibility for any problems arising from
1590 correct or incorrect implementation or use of this design.
1591

1591 **14 Contact Information**

1592

1593 Team Leader

1594 Name: Lisa Carnahan
1595 Company: NIST
1596 Street: 100 Bureau Drive STOP 8970
1597 City, State, Postal Code: Gaithersburg, MD 20899-8970
1598 Country: USA
1599 Phone: (301) 975-3362
1600 Email: lisa.carnahan@nist.gov

1601

1602 Editor

1603 Name: Sally Fuger
1604 Company: Automotive Industry Action Group
1605 Street: 26200 Lahser Road, Suite 200
1606 City, State, Postal Code: Southfield, MI 48034
1607 Country: USA
1608 Phone: (248) 358-9744
1609 Email: sfuger@aiag.org

1610

1611 Technical Editor

1612 Name: Farrukh S. Najmi
1613 Company: Sun Microsystems
1614 Street: 1 Network Dr., MS BUR02-302
1615 City, State, Postal Code: Burlington, MA, 01803-0902
1616 Country: USA
1617 Phone: (781) 442-0703
1618 Email: najmi@east.sun.com

1619

1620

1620 **Copyright Statement**

1621 Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

1622

1623 OASIS takes no position regarding the validity or scope of any intellectual
1624 property or other rights that might be claimed to pertain to the implementation or
1625 use of the technology described in this document or the extent to which any
1626 license under such rights might or might not be available; neither does it
1627 represent that it has made any effort to identify any such rights. Information on
1628 OASIS's procedures with respect to rights in OASIS specifications can be found
1629 at the OASIS website. Copies of claims of rights made available for publication
1630 and any assurances of licenses to be made available, or the result of an attempt
1631 made to obtain a general license or permission for the use of such proprietary
1632 rights by implementors or users of this specification, can be obtained from the
1633 OASIS Executive Director.

1634

1635 OASIS invites any interested party to bring to its attention any copyrights, patents
1636 or patent applications, or other proprietary rights which may cover technology
1637 that may be required to implement this specification. Please address the
1638 information to the OASIS Executive Director.

1639

1640 Copyright ©The Organization for the Advancement of Structured Information
1641 Standards [OASIS] 2001. All Rights Reserved.

1642 This document and translations of it may be copied and furnished to others, and
1643 derivative works that comment on or otherwise explain it or assist in its
1644 implementation may be prepared, copied, published and distributed, in whole or
1645 in part, without restriction of any kind, provided that the above copyright notice
1646 and this paragraph are included on all such copies and derivative works.

1647 However, this document itself may not be modified in any way, such as by
1648 removing the copyright notice or references to OASIS, except as needed for the
1649 purpose of developing OASIS specifications, in which case the procedures for
1650 copyrights defined in the OASIS Intellectual Property Rights document must be
1651 followed, or as required to translate it into languages other than English.

1652 The limited permissions granted above are perpetual and will not be revoked by
1653 OASIS or its successors or assigns.

1654 This document and the information contained herein is provided on an "AS IS"
1655 basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
1656 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
1657 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
1658 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
1659 PURPOSE."