



Creating A Single Global Electronic Market

1
2
3
4

5 **OASIS/ebXML Registry Information Model v2.0** 6 **DRAFT**

7 **OASIS/ebXML Registry Technical Committee**

8 **5 December 2001**

9

10 **1 Status of this Document**

11
12 Distribution of this document is unlimited.

13
14 ***This version:***

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

16
17 ***Latest version:***

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

19
20

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

21 This document, in its current form, is a draft working document of the OASIS
22 ebXML Registry Technical Committee. It builds upon version 1.0 which was
23 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 Breining, 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>contentType</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	7.10.1	26
135	7.10.2	Attribute <i>externalURI</i>	27
136	7.10.3	Method <i>Summary</i>	27

137	8	REGISTRY AUDIT TRAIL	27
138	8.1	CLASS AUDITABLEEVENT.....	27
139	8.1.1	Attribute Summary.....	28
140	8.1.2	Attribute eventType.....	28
141	8.1.3	Attribute registryObject.....	28
142	8.1.4	Attribute timestamp.....	28
143	8.1.5	Attribute user.....	29
144	8.2	CLASS USER.....	29
145	8.2.1	Attribute Summary.....	29
146	8.2.2	Attribute address.....	29
147	8.2.3	Attribute emailAddresses.....	29
148	8.2.4	Attribute organization.....	29
149	8.2.5	Attribute personName.....	30
150	8.2.6	Attribute telephoneNumbers.....	30
151	8.2.7	Attribute url.....	30
152	8.3	CLASS ORGANIZATION.....	30
153	8.3.1	Attribute Summary.....	30
154	8.3.2	Attribute address.....	30
155	8.3.3	Attribute parent.....	30
156	8.3.4	Attribute primaryContact.....	30
157	8.3.5	Attribute telephoneNumbers.....	31
158	8.4	CLASS POSTALADDRESS.....	31
159	8.4.1	Attribute Summary.....	31
160	8.4.2	Attribute city.....	31
161	8.4.3	Attribute country.....	31
162	8.4.4	Attribute postalCode.....	31
163	8.4.5	Attribute state.....	31
164	8.4.6	Attribute street.....	31
165	8.4.7	Attribute streetNumber.....	32
166	8.4.8	Method Summary.....	32
167	8.5	CLASS TELEPHONENUMBER.....	32
168	8.5.1	Attribute Summary.....	32
169	8.5.2	Attribute areaCode.....	32
170	8.5.3	Attribute countryCode.....	32
171	8.5.4	Attribute extension.....	32
172	8.5.5	Attribute number.....	33
173	8.5.6	Attribute phoneType.....	33
174	8.6	CLASS EMAILADDRESS.....	33
175	8.6.1	Attribute Summary.....	33
176	8.6.2	Attribute address.....	33
177	8.6.3	Attribute type.....	33
178	8.7	CLASS PERSONNAME.....	33
179	8.7.1	Attribute Summary.....	33
180	8.7.2	Attribute firstName.....	33
181	8.7.3	Attribute lastName.....	34
182	8.7.4	Attribute middleName.....	34

183	8.8	CLASS SERVICE	34
184	8.8.1	Attribute Summary	34
185	8.8.2	Method Summary	34
186	8.9	CLASS SERVICEBINDING	34
187	8.9.1	Attribute Summary	35
188	8.9.2	Attribute accessURI	35
189	8.9.3	Attribute targetBinding	35
190	8.9.4	Method Summary	35
191	8.10	CLASS SPECIFICATIONLINK	35
192	8.10.1	Attribute Summary	36
193	8.10.2	Attribute specificationObject	36
194	8.10.3	Attribute usageDescription	36
195	8.10.4	Attribute usageParameters	36
196	9	ASSOCIATION OF REGISTRY OBJECTS	37
197	9.1	EXAMPLE OF AN ASSOCIATION	37
198	9.2	SOURCE AND TARGET OBJECTS	37
199	9.3	ASSOCIATION TYPES	37
200	9.4	INTRAMURAL ASSOCIATION	38
201	9.5	EXTRAMURAL ASSOCIATION	38
202	9.6	CONFIRMATION OF AN ASSOCIATION	39
203	9.6.1	Confirmation of Intramural Associations	39
204	9.6.2	Confirmation of Extramural Associations	40
205	9.7	VISIBILITY OF UNCONFIRMED ASSOCIATIONS	40
206	9.8	POSSIBLE CONFIRMATION STATES	40
207	9.9	CLASS ASSOCIATION	40
208	9.9.1	Attribute Summary	41
209	9.9.2	Attribute associationType	41
210	9.9.3	Attribute sourceObject	42
211	9.9.4	Attribute targetObject	42
212	10	CLASSIFICATION OF REGISTRY OBJECT	43
213	10.1	CLASS CLASSIFICATIONSCHEME	46
214	10.1.1	Attribute Summary	46
215	10.1.2	Attribute isInternal	46
216	10.1.3	Attribute nodeType	46
217	10.2	CLASS CLASSIFICATIONNODE	47
218	10.2.1	Attribute Summary	47
219	10.2.2	Attribute parent	47
220	10.2.3	Attribute code	47
221	10.2.4	Method Summary	47
222	10.2.5	Canonical Path Syntax	48
223	10.3	CLASS CLASSIFICATION	49
224	10.3.1	Attribute Summary	49
225	10.3.2	Attribute classificationScheme	50
226	10.3.3	Attribute classificationNode	50

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

248 **Table of Figures**

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

259 **Table of Tables**

260	Table 1: Sample <i>Classification</i> Schemes.....	53
261		
262		

262 **3 Introduction**

263 **3.1 Summary of Contents of Document**

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

265

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

269 **3.2 General Conventions**

270 The following conventions are used throughout this document:

271

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

274

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

278

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

281

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

285

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

287

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

291

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

294 **3.2.1 Naming Conventions**

295

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

298 Capitalization styles are used in the following conventions:

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

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

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

310 **3.3 Audience**

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

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

315 **3.4 Related Documents**

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

- 318
319 a) ebXML Registry Services Specification [ebRS] - defines the actual
320 *Registry Services* based on this information model
321 b) ebXML Collaboration-Protocol Profile and Agreement Specification
322 [ebCPP] - defines how profiles can be defined for a *Party* and how two
323 *Parties'* profiles may be used to define a *Party* agreement
324

325 **4 Design Objectives**

326 **4.1 Goals**

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

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

336 **5 System Overview**

337 **5.1 Role of ebXML Registry**

338

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

345 **5.2 Registry Services**

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

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

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

355 The Registry information model:

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

358

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

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

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

368

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

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

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

374 **5.6 Conformance to an ebXML Registry**

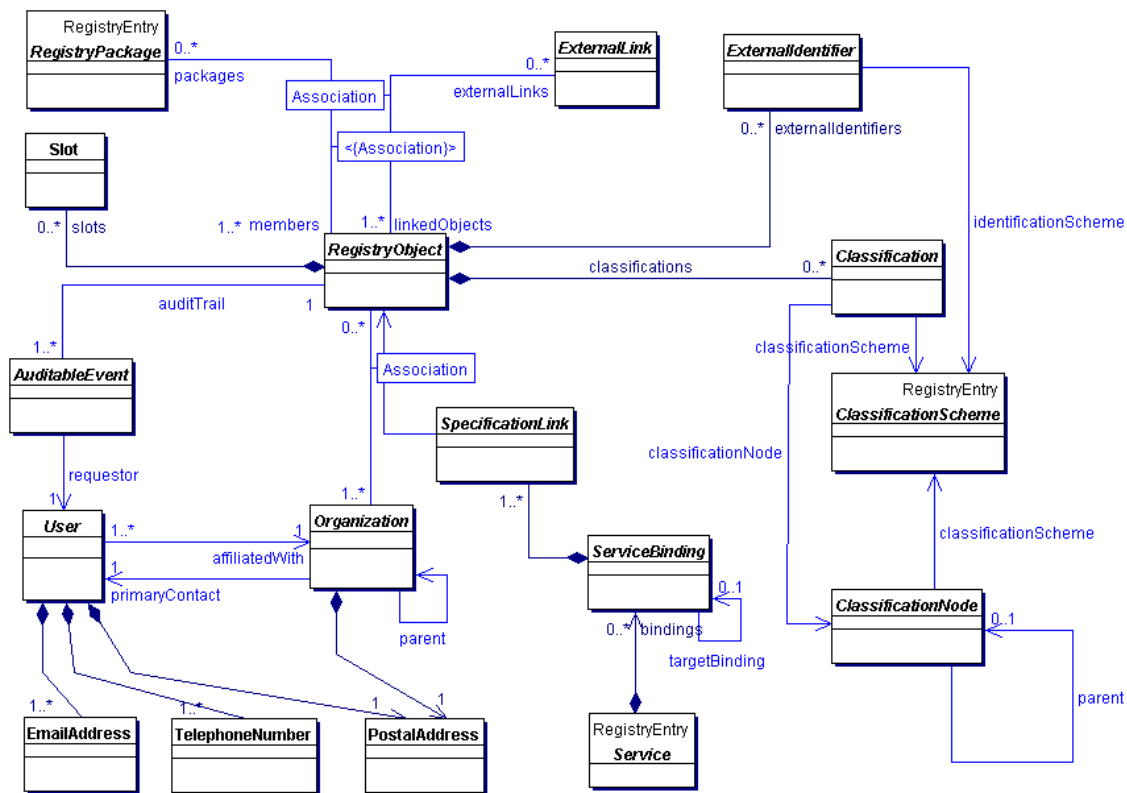
375 If an *Implementation* claims *Conformance* to this specification then it supports all
 376 required information model *Classes* and interfaces, their attributes and their
 377 semantic definitions that are visible through the ebXML *Registry Services*.

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

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

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

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



388

389

Figure 1: Information Model High Level Public View

390 **6.1 RegistryObject**

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

395 **6.2 Slot**

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

402 **6.3 Association**

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

406 **6.4 ExternalIdentifier**

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

410 **6.5 ExternalLink**

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

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

422 **6.6 ClassificationScheme**

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

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

431 **6.7 ClassificationNode**

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

438 **6.8 Classification**

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

445 **6.9 RegistryPackage**

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

448 **6.10 AuditableEvent**

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

452 **6.11 User**

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

456 **6.12 PostalAddress**

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

459 **6.13 EmailAddress**

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

462 **6.14 Organization**

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

466 **6.15 Service**

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

469 **6.16 ServiceBinding**

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

474 **6.17 SpecificationLink**

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

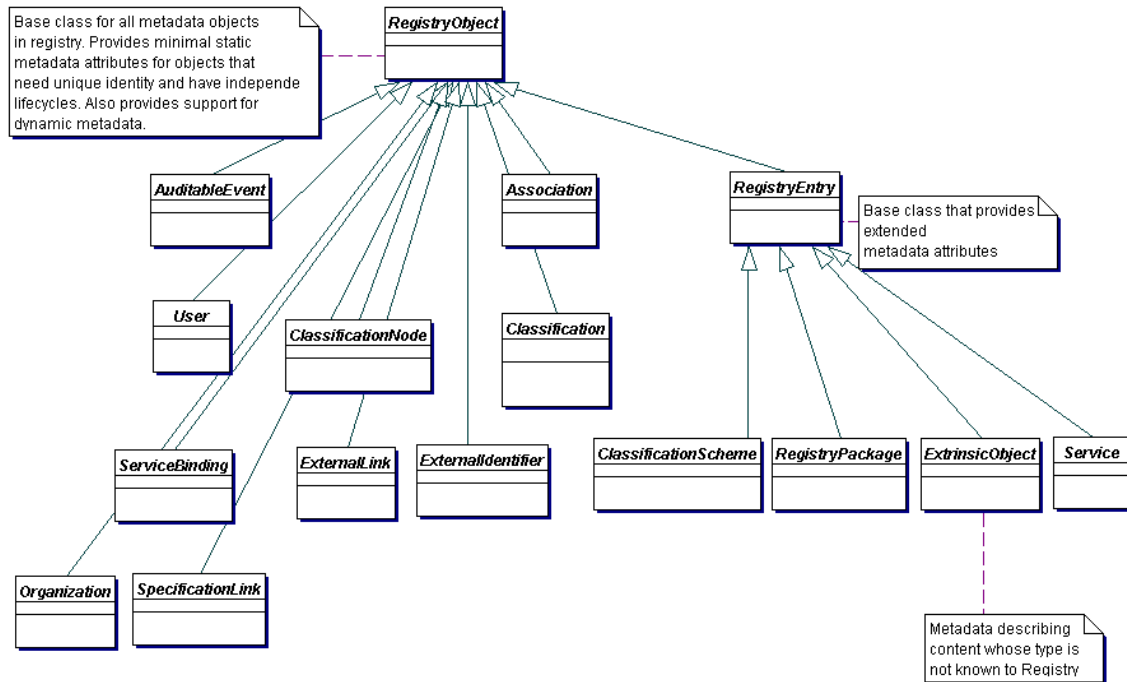
481 **7 Registry Information Model: Detail View**

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

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

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

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



499

500

501

Figure 2: Information Model *Inheritance View*

502

7.1 Attribute and Methods of Information Model Classes

503

Information model classes are defined primarily in terms of the attributes they 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.

504

505

506

507

508

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].

509

510

511

512

513

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.

514

515

516

516 7.2 Data Types

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

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

520

521 7.3 Internationalization (I18N) Support

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

525

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

529 7.3.1 Class InternationalString

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

536 **7.3.2 Class LocalizedString**

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

541 **7.4 Class RegistryObject**

542 **Direct Known Subclasses:**

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

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

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

555 **7.4.1 Attribute Summary**

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

558

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

559

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

560 **7.4.2 Attribute accessControlPolicy**

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

565 **7.4.3 Attribute description**

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

569 **7.4.4 Attribute id**

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

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

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

583 **7.4.5 Attribute name**

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

587 **7.4.6 Attribute objectType**

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

593 **7.4.6.1 Pre-defined Object Types**

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

598
599

600 These pre-defined object types are defined as a *ClassificationScheme*. While the
 601 scheme may easily be extended a *Registry* MUST support the object types listed
 602 below.
 603

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

604

605 **7.4.7 Method Summary**

606 In addition to its attributes, the RegistryObject class also defines the following
 607 methods. These methods are used to navigate relationship links from a
 608 RegistryObject instance to other objects.
 609

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.

610

611

612 **7.5 Class RegistryEntry**613 **Super Classes:**614 [RegistryObject](#)

615

616 **Direct Known Subclasses:**617 [ClassificationScheme](#), [ExtrinsicObject](#), [RegistryPackage](#)

618

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

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

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

630 **7.5.1 Attribute Summary**

631

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

632

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

635 **7.5.2 Attribute expiration**

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

642 **7.5.3 Attribute majorVersion**

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

647 **7.5.4 Attribute minorVersion**

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

652 **7.5.5 Attribute stability**

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

656 **7.5.5.1 Pre-defined RegistryEntry Stability Enumerations**

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

661

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.

662

663 **7.5.6 Attribute status**

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

666 **7.5.6.1 Pre-defined RegistryObject Status Types**

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

669

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> .

670

671 **7.5.7 Attribute userVersion**

672 Each RegistryEntry instance may have a userVersion. The userVersion is similar
 673 to the majorVersion-minorVersion tuple. They both provide an indication of the
 674 version of the object. The majorVersion-minorVersion tuple is provided by the
 675 registry while userVersion provides a user specified version for the object.
 676

677 **7.5.8 Method Summary**

678 In addition to its attributes, the RegistryEntry class also defines the following
 679 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>

680

681 **7.6 Class Slot**

682 Slot instances provide a dynamic way to add arbitrary attributes to
 683 RegistryObject instances. This ability to add attributes dynamically to
 684 RegistryObject instances enables extensibility within the information model.
 685

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

688 **7.6.1 Attribute Summary**

689

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

690

691 **7.6.2 Attribute name**

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

695 **7.6.3 Attribute slotType**

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

698 **7.6.4 Attribute values**

699 A Slot instance must have a Collection of values. The collection of values may be
700 empty. Since a Slot represent an extensible attribute whose value may be a
701 collection, therefore a Slot is allowed to have a collection of values rather than a
702 single value.
703

704 **7.7 Class ExtrinsicObject**

705 **Super Classes:**

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

707

708

709 ExtrinsicObjects provide metadata that describes submitted content whose type
710 is not intrinsically known to the *Registry* and therefore **MUST** be described by
711 means of additional attributes (e.g., mime type).
712

713 Since the registry can contain arbitrary content without intrinsic knowledge about
714 that content, ExtrinsicObjects require special metadata attributes to provide some
715 knowledge about the object (e.g., mime type).
716

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

719 **7.7.1 Attribute Summary**

720

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

721

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

724 **7.7.2 Attribute isOpaque**

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

730 **7.7.3 Attribute mimeType**

731 Each ExtrinsicObject instance may have a mimeType attribute defined. The
 732 mimeType provides information on the type of repository item catalogued by the
 733 ExtrinsicObject instance.
 734

735 **7.8 Class RegistryPackage**

736 **Super Classes:**

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

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

742 **7.8.1 Attribute Summary**

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

747 **7.8.2 Method Summary**

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

750

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

751

752 **7.9 Class ExternalIdentifier**

753 **Super Classes:**

754 [RegistryObject](#)

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

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

763 **7.9.1 Attribute Summary**

764

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

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

766 **7.9.2 Attribute identificationScheme**

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

771 **7.9.3 Attribute registryObject**

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

774 **7.9.4 Attribute value**

775 Each ExternalIdentifier instance must have a value attribute that provides the
 776 identifier value for this ExternalIdentifier (e.g., the actual social security number).
 777

778 **7.10 Class ExternalLink**

779 **Super Classes:**

780 [RegistryObject](#)

781

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

786 **7.10.1 Attribute Summary**

787

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

788

789 **7.10.2 Attribute externalURI**

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

795 **7.10.3 Method Summary**

796 In addition to its attributes, the ExternalLink class also defines the following
 797 methods.

798

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

799

800 **8 Registry Audit Trail**

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

805

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

812 **8.1 Class AuditableEvent**813 **Super Classes:**814 [RegistryObject](#)

815

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

820

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

823

824 Often such *Events* effect a change in the life cycle of a RegistryObject. For
 825 example a client request could Create, Update, Deprecate or Delete a
 826 RegistryObject. An AuditableEvent is created if and only if a request creates or
 827 alters the content or ownership of a RegistryObject. Read-only requests do not
 828 generate an AuditableEvent. No AuditableEvent is generated for a
 829 RegistryObject when it is classified, assigned to a RegistryPackage or associated
 830 with another RegistryObject.

831 **8.1.1 Attribute Summary**

832

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

833

834 **8.1.2 Attribute eventType**

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

837 **8.1.2.1 Pre-defined Auditable Event Types**

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

841

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.

842 **8.1.3 Attribute registryObject**

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

845 **8.1.4 Attribute timestamp**

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

848 **8.1.5 Attribute user**

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

851
852

853 **8.2 Class User**

854 **Super Classes:**

855 [RegistryObject](#)

856

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

859 **8.2.1 Attribute Summary**

860

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

861

862 **8.2.2 Attribute address**

863 Each User instance must have an address attribute that provides the postal
864 address for that user.

865 **8.2.3 Attribute emailAddresses**

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

869 **8.2.4 Attribute organization**

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

872 **8.2.5 Attribute personName**

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

875 **8.2.6 Attribute telephoneNumbers**

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

879 **8.2.7 Attribute url**

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

882 **8.3 Class Organization**883 **Super Classes:**

884 [RegistryObject](#)

885

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

889 **8.3.1 Attribute Summary**

890

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

891

892 **8.3.2 Attribute address**

893 Each Organization instance must have an address attribute that provides the
894 postal address for that organization.

895 **8.3.3 Attribute parent**

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

898 **8.3.4 Attribute primaryContact**

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

901 **8.3.5 Attribute telephoneNumbers**

902 Each Organization instance must have a telephoneNumbers attribute that
 903 contains the Collection of TelephoneNumber instances for each telephone
 904 number defined for that organization. An Organization must have at least one
 905 telephone number.

906 **8.4 Class PostalAddress**

907 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal
 908 address.

909 **8.4.1 Attribute Summary**

910

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

911

912 **8.4.2 Attribute city**

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

914 **8.4.3 Attribute country**

915 Each PostalAddress may have a country attribute identifying the country for that
 916 address.

917 **8.4.4 Attribute postalCode**

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

920 **8.4.5 Attribute state**

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

923 **8.4.6 Attribute street**

924 Each PostalAddress may have a street attribute identifying the street name for
 925 that address.

926 **8.4.7 Attribute streetNumber**

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

929 **8.4.8 Method Summary**

930 In addition to its attributes, the PostalAddress class also defines the following
931 methods.

932

Method Summary of ExternalLink	
Collection	getSlots() 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.

933

934 **8.5 Class TelephoneNumber**

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

936 **8.5.1 Attribute Summary**

937

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

938

939 **8.5.2 Attribute areaCode**

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

942 **8.5.3 Attribute countryCode**

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

945 **8.5.4 Attribute extension**

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

948 **8.5.5 Attribute number**

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

952 **8.5.6 Attribute phoneType**

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

956 **8.6 Class EmailAddress**

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

958 **8.6.1 Attribute Summary**

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

959 **8.6.2 Attribute address**

960 Each EmailAddress instance must have an address attribute that provides the
961 actual email address.

962 **8.6.3 Attribute type**

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

966 **8.7 Class PersonName**

967 A simple *Entity Class* for a person's name.

968 **8.7.1 Attribute Summary**

969

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

970 **8.7.2 Attribute firstName**

971 Each PersonName may have a firstName attribute that is the first name of the
972 person.

973 **8.7.3 Attribute lastName**

974 Each PersonName may have a lastName attribute that is the last name of the
975 person.

976 **8.7.4 Attribute middleName**

977 Each PersonName may have a middleName attribute that is the middle name of the
978 person.

979 **8.8 Class Service**

980 **Super Classes:**

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

982

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

984 **8.8.1 Attribute Summary**

985 The Service class does not define any specialized attributes other than its
986 inherited attributes.

987 **8.8.2 Method Summary**

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

989

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

990 **8.9 Class ServiceBinding**

991 **Super Classes:**

992 [RegistryObject](#)

993

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

997 The description attribute of ServiceBinding provides details about the relationship
998 between several specification links comprising the Service Binding. This
999 description can be useful for human understanding such that the runtime system
1000 can be appropriately configured by the human being. There is possibility of
1001 enforcing a structure on this description for enabling machine processing of the
1002 Service Binding, which is however not addressed by the current document.

1003

1004

1005 **8.9.1 Attribute Summary**

1006

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

1007

1008 **8.9.2 Attribute accessURI**

1009 A ServiceBinding may have an accessURI attribute that defines the URI to
 1010 access that ServiceBinding. This attribute is ignored if a targetBinding attribute is
 1011 specified for the ServiceBinding. If the URI is a URL then a registry must validate
 1012 the URL to be resolvable at the time of submission before accepting a
 1013 ServiceBinding submission to the registry.

1014 **8.9.3 Attribute targetBinding**

1015 A ServiceBinding may have a targetBinding attribute defined which references
 1016 another ServiceBinding. A targetBinding may be specified when a service is
 1017 being redirected to another service. This allows the rehosting of a service by
 1018 another service provider.

1019 **8.9.4 Method Summary**

1020 In addition to its attributes, the ServiceBinding class also defines the following
 1021 methods.

1022

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

1023

1024

1025

1026 **8.10 Class SpecificationLink**1027 **Super Classes:**1028 [RegistryObject](#)

1029

1030 A SpecificationLink provides the linkage between a ServiceBinding and one of its
 1031 technical specifications that describes how to use the service using the
 1032 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink
 1033 instances that describe how to access the service using a technical specification
 1034 in form of a WSDL document or a CORBA IDL document.

1035 **8.10.1 Attribute Summary**

1036

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

1037

1038 **8.10.2 Attribute specificationObject**

1039 A SpecificationLink instance must have a specificationObject attribute that
 1040 provides a reference to a RegistryObject instance that provides a technical
 1041 specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject
 1042 instance representing the technical specification (e.g., a WSDL document).

1043 **8.10.3 Attribute usageDescription**

1044 A SpecificationLink instance may have a usageDescription attribute that provides
 1045 a textual description of how to use the optional usageParameters attribute
 1046 described next. The usageDescription is of type InternationalString, thus allowing
 1047 the description to be in multiple languages.

1048 **8.10.4 Attribute usageParameters**

1049 A SpecificationLink instance may have a usageParameters attribute that provides
 1050 a collection of Strings representing the instance specific parameters needed to
 1051 use the technical specification (e.g., a WSDL document) specified by this
 1052 SpecificationLink object.

1053

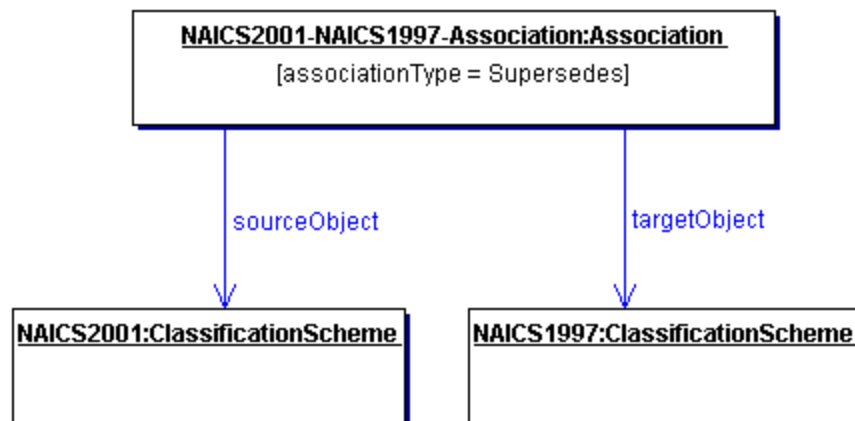
1053 9 Association of Registry Objects

1054 A RegistryObject instance may be *associated* with zero or more RegistryObject
 1055 instances. The information model defines an Association class, an instance of
 1056 which may be used to associate any two RegistryObject instances.

1057 9.1 Example of an Association

1058 One example of such an association is between two ClassificationScheme
 1059 instances, where one ClassificationScheme supersedes the other
 1060 ClassificationScheme as shown in Figure 3. This may be the case when a new
 1061 version of a ClassificationScheme is submitted.

1062 In Figure 3, we see how an Association is defined between a new version of the
 1063 NAICS ClassificationScheme and an older version of the NAICS
 1064 ClassificationScheme.
 1065



1066

1067

Figure 3: Example of RegistryObject Association

1068 9.2 Source and Target Objects

1069 An Association instance represents an association between a *source*
 1070 RegistryObject and a *target* RegistryObject. These are referred to as
 1071 *sourceObject* and *targetObject* for the Association instance. It is important which
 1072 object is the sourceObject and which is the targetObject as it determines the
 1073 directional semantics of an Association.

1074 In the example in Figure 3, it is important to make the newer version of NAICS
 1075 ClassificationScheme be the sourceObject and the older version of NAICS be the
 1076 targetObject because the associationType implies that the sourceObject
 1077 supersedes the targetObject (and not the other way around).

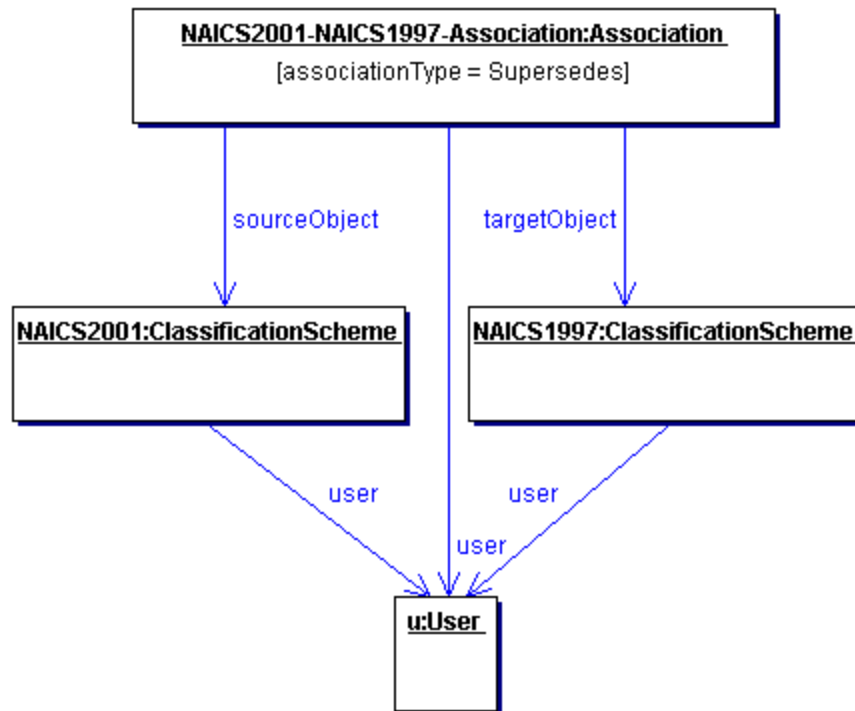
1078 9.3 Association Types

1079 Each Association must have an associationType attribute that identifies the type
 1080 of that association.

1081 9.4 Intramural Association

1082 A common use case for the Association class is when a User “u” creates an
 1083 Association “a” between two RegistryObjects “o1” and “o2” where association “a”
 1084 and RegistryObjects “o1” and “o2” are objects that were created by the same
 1085 User “u.” This is the simplest use case, where the association is between two
 1086 objects that are owned by the same User that is defining the Association. Such
 1087 associations are referred to as *intramural associations*.
 1088 Figure 4 below, extends the previous example in Figure 3 for the intramural
 1089 association case.

1090



1091

1092

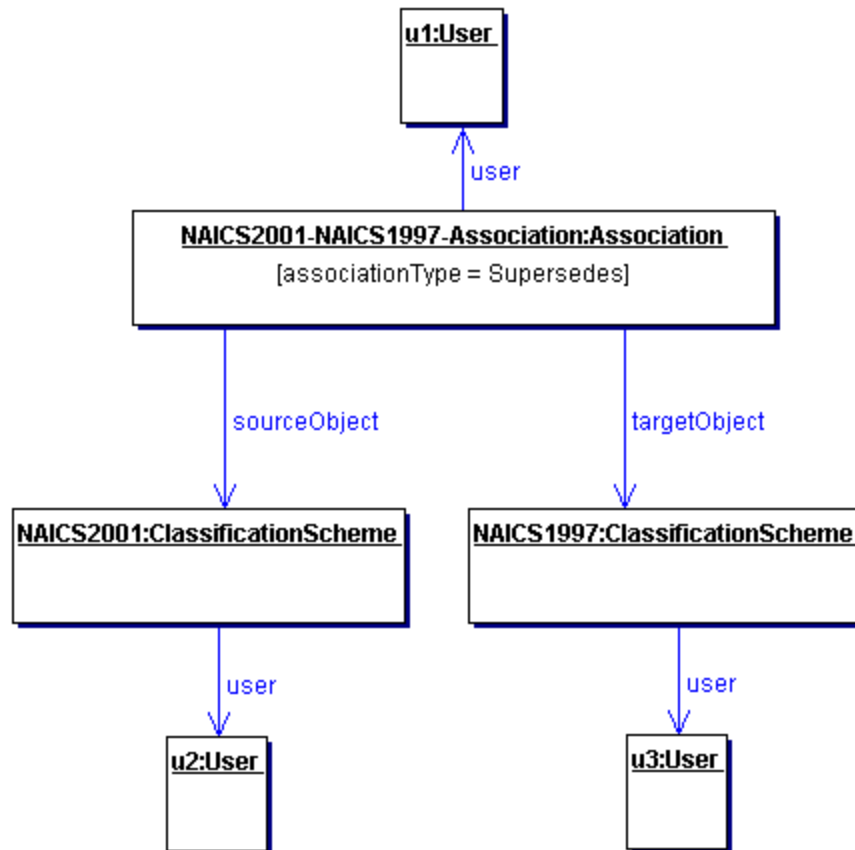
Figure 4: Example of Intramural Association

1093 9.5 Extramural Association

1094 The information model also allows more sophisticated use cases. For example, a
 1095 User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”
 1096 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2”
 1097 are owned by User “u2” and User “u3” respectively.

1098 In this use case an Association is defined where either or both objects that are
 1099 being associated are owned by a User different from the User defining the
 1100 Association. Such associations are referred to as *extramural associations*. The
 1101 Association class provides a convenience method called `isExtramural` that
 1102 returns "true" if the Association instance is an extramural Association.

1103 Figure 5 below, extends the previous example in Figure 3 for the extramural
 1104 association case. Note that it is possible for an extramural association to have
 1105 two distinct Users rather than three distinct Users as shown in Figure 5. In such
 1106 case, one of the two users owns two of the three objects involved (Association,
 1107 sourceObject and targetObject).
 1108



1109
 1110

Figure 5: Example of Extramural Association

1111 9.6 Confirmation of an Association

1112 An association may need to be confirmed by the parties whose objects are
 1113 involved in that Association as the sourceObject or targetObject. This section
 1114 describes the semantics of confirmation of an association by the parties involved.

1115 9.6.1 Confirmation of Intramural Associations

1116 Intramural associations may be viewed as declarations of truth and do not
 1117 require any explicit steps to confirm that Association as being true. In other
 1118 words, intramural associations are implicitly considered confirmed.

1119 **9.6.2 Confirmation of Extramural Associations**

1120 Extramural associations may be thought of as a unilateral assertion that may not
1121 be viewed as truth until it has been confirmed by the other (extramural) parties
1122 involved (Users “u2” and “u3” in the example in section 9.5).

1123 To confirm an extramural association, each of the extramural parties (parties that
1124 own the source or target object but do not own the Association) must submit an
1125 identical Association (clone Association) as the Association they are intending to
1126 confirm using a SubmitObjectsRequest. The clone Association must have the
1127 same id as the original Association.

1128 **9.7 Visibility of Unconfirmed Associations**

1129 Extramural associations require each extramural party to confirm the assertion
1130 being made by the extramural Association before the Association is visible to
1131 third parties that are not involved in the Association. This ensures that
1132 unconfirmed Associations are not visible to third party registry clients.

1133 **9.8 Possible Confirmation States**

1134 Assume the most general case where there are three distinct User instances as
1135 shown in Figure 5 for an extramural Association. The extramural Association
1136 needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3”
1137 in example) in order to be fully confirmed. The methods
1138 `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the
1139 Association class provide access to the confirmation state for both the
1140 `sourceObject` and `targetObject`. A third convenience method called
1141 `isConfirmed` provides a way to determine whether the Association is fully
1142 confirmed or not. So there are the following four possibilities related to the
1143 confirmation state of an extramural Association:

- 1144 ○ The Association is confirmed neither by the owner of the `sourceObject` nor
1145 by the owner of the `targetObject`.
- 1146 ○ The Association is confirmed by the owner of the `sourceObject` but it is not
1147 confirmed by the owner of the `targetObject`.
- 1148 ○ The Association is not confirmed by the owner of the `sourceObject` but it is
1149 confirmed by the owner of the `targetObject`.
- 1150 ○ The Association is confirmed by both the owner of the `sourceObject` and
1151 the owner of the `targetObject`. This is the only state where the Association
1152 is fully confirmed.

1153

1154 **9.9 Class Association**

1155 **Super Classes:**

1156 [RegistryObject](#)

1157

1158

1159 Association instances are used to define many-to-many associations among
1160 RegistryObjects in the information model.

1161
 1162 An *Instance* of the Association Class represents an association between two
 1163 RegistryObjects.

1164 9.9.1 Attribute Summary

1165

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

1166

1167 9.9.2 Attribute associationType

1168 Each Association must have an associationType attribute that identifies the type
 1169 of that association.

1170 9.9.2.1 Pre-defined Association Types

1171 The following table lists pre-defined association types. These pre-defined
 1172 association types are defined as a *Classification* scheme. While the scheme may
 1173 easily be extended a *Registry* MUST support the association types listed below.

1174

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.

1175

1176 **9.9.3 Attribute sourceObject**

1177 Each Association must have a sourceObject attribute that references the
1178 RegistryObject instance that is the source of that association.

1179 **9.9.4 Attribute targetObject**

1180 Each Association must have a targetObject attribute that references the
1181 RegistryObject instance that is the target of that association.

1182

1183

Method Summary of Association	
boolean	<p>isConfirmed()</p> <p>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>isConfirmedBySourceOwner()</p> <p>Returns true if the association has been confirmed by the owner of the sourceObject. For intramural Associations always return true.</p>
boolean	<p>isConfirmedByTargetOwner()</p> <p>Returns true if the association has been confirmed by the owner of the targetObject. For intramural Associations always return true.</p>
boolean	<p>isExtramural()</p> <p>Returns true if the sourceObject and/or the targetObject are owned by a User that is different from the User that created the Association.</p>

1184

1185 **10 Classification of RegistryObject**

1186 This section describes the how the information model supports *Classification of*
1187 *RegistryObject*. It is a simplified version of the *OASIS* classification model [OAS].

1188

1189 A *RegistryObject* may be classified in many ways. For example the
1190 *RegistryObject* for the same *Collaboration Protocol Profile (CPP)* may be
1191 classified by its industry, by the products it sells and by its geographical location.

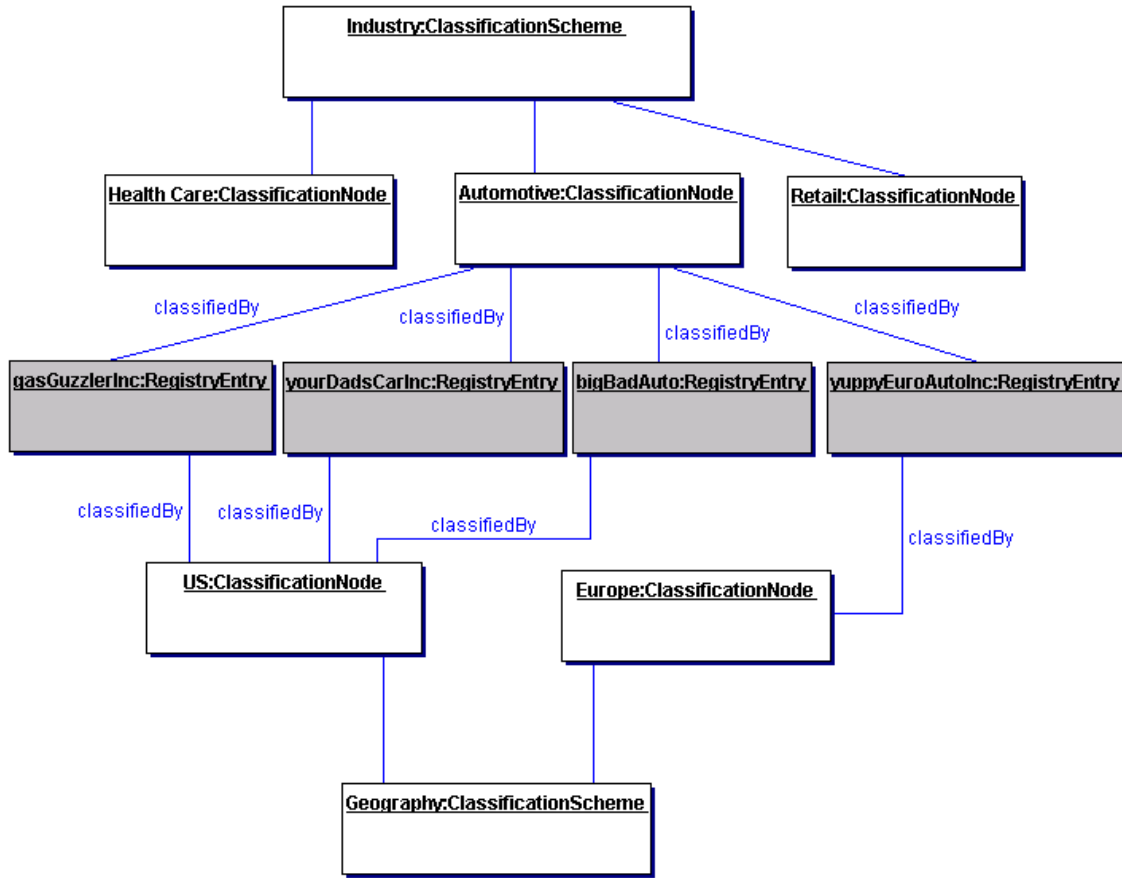
1192

1193 A general *ClassificationScheme* can be viewed as a *Classification* tree. In the
1194 example shown in Figure 6, *RegistryObject* instances representing *Collaboration*
1195 *Protocol Profiles* are shown as shaded boxes. Each *Collaboration Protocol*
1196 *Profile* represents an automobile manufacturer. Each *Collaboration Protocol*
1197 *Profile* is classified by the *ClassificationNode* named "Automotive" under the
1198 *ClassificationScheme* instance with name "Industry." Furthermore, the US
1199 Automobile manufacturers are classified by the US *ClassificationNode* under the
1200 *ClassificationScheme* with name "Geography." Similarly, a European automobile
1201 manufacturer is classified by the "Europe" *ClassificationNode* under the
1202 *ClassificationScheme* with name "Geography."

1203

1204 The example shows how a *RegistryObject* may be classified by multiple
1205 *ClassificationNode* instances under multiple *ClassificationScheme* instances
1206 (e.g., Industry, Geography).

1207



1208
1209

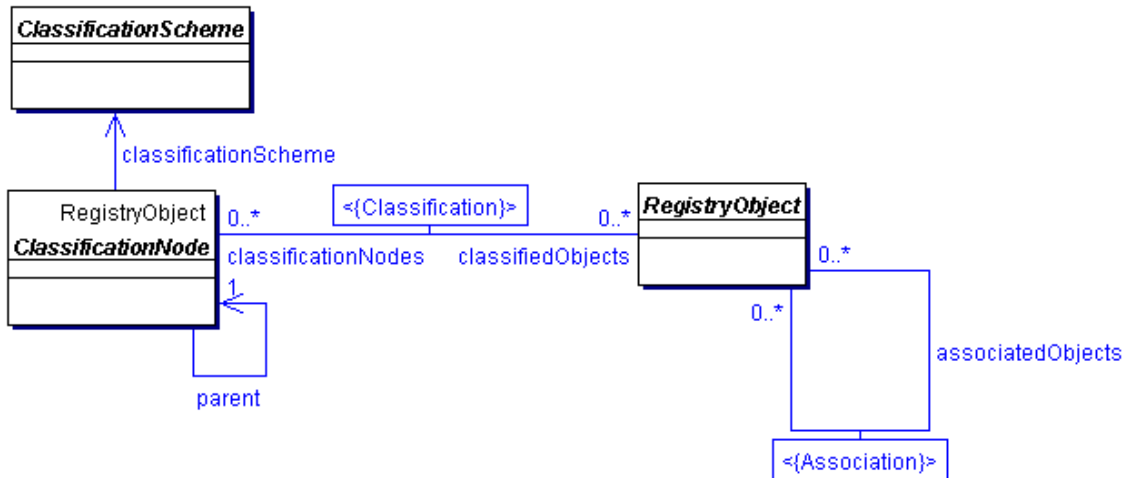
Figure 6: Example showing a *Classification Tree*

1210
1211
1212
1213
1214
1215
1216
1217
1218

[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

1219
1220
1221

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.



1222

1223

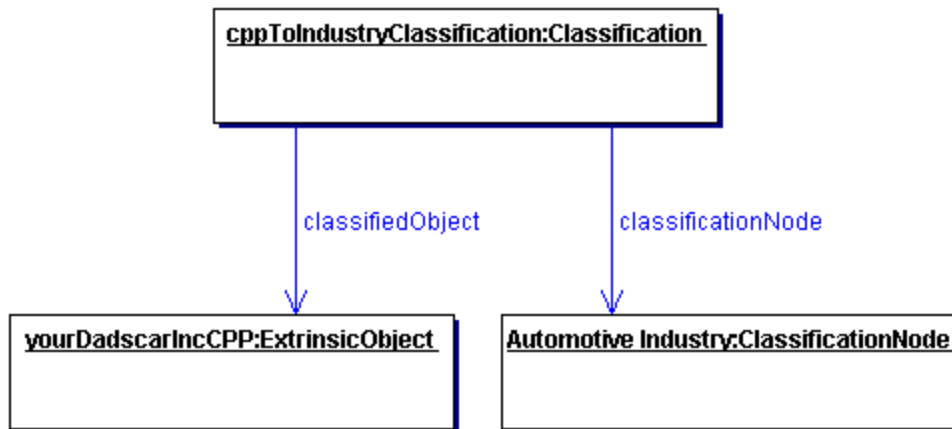
Figure 7: Information Model *Classification* View

1224

1225

1226 A Classification is somewhat like a specialized form of an Association. Figure 8
 1227 shows an example of an ExtrinsicObject *Instance* for a *Collaboration Protocol*
 1228 *Profile (CPP)* object that is classified by a ClassificationNode representing the
 1229 Industry that it belongs to.

1230



1231

1232

Figure 8: Classification *Instance* Diagram

1233

1234

1235

1236

1237

1238

1239 **10.1 Class ClassificationScheme**

1240 **Base classes:**

1241 [RegistryEntry](#), [RegistryObject](#)

1242

1243 A ClassificationScheme instance is metadata that describes a registered
1244 taxonomy. The taxonomy hierarchy may be defined internally to the
1245 Registry by instances of ClassificationNode or it may be defined externally
1246 to the Registry, in which case the structure and values of the taxonomy
1247 elements are not known to the Registry.

1248 In the first case the classification scheme is defined to be *internal* and in
1249 the second case the classification scheme is defined to be *external*.

1250 The ClassificationScheme class inherits attributes and methods from the
1251 RegistryObject and RegistryEntry classes.

1252

1253 **10.1.1 Attribute Summary**

1254

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

1255 Note that attributes inherited by ClassificationScheme class from the
1256 RegistryEntry class are not shown.

1257

1258 **10.1.2 Attribute isInternal**

1259 When submitting a ClassificationScheme instance the Submitting Organization
1260 needs to declare whether the ClassificationScheme instance represents an
1261 internal or an external taxonomy. This allows the registry to validate the
1262 subsequent submissions of ClassificationNode and Classification instances in
1263 order to maintain the type of ClassificationScheme consistent throughout its
1264 lifecycle.

1265

1266 **10.1.3 Attribute nodeType**

1267 When submitting a ClassificationScheme instance the Submitting Organization
1268 needs to declare what is the structure of taxonomy nodes that this
1269 ClassificationScheme instance will represent. This attribute is an enumeration
1270 with the following values:

- 1271 - UniqueCode. This value says that each node of the taxonomy has
1272 a unique code assigned to it.
- 1273 - EmbeddedPath. This value says that a unique code assigned to
1274 each node of the taxonomy at the same time encodes its path. This
1275 is the case in the NAICS taxonomy.

1276 - NonUniqueCode. In some cases nodes are not unique, and it is
 1277 necessary to nominate the full path in order to identify the node. For
 1278 example, in a geography taxonomy Moscow could be under both
 1279 Russia and the USA, where there are five cities of that name in
 1280 different states.

1281 This enumeration might expand in the future with some new values. An example
 1282 for possible future values for this enumeration might be NamedPathElements for
 1283 support of Named-Level taxonomies such as Genus/Species.
 1284

1285 **10.2 Class ClassificationNode**

1286 **Base classes:**

1287 [RegistryObject](#)

1288 ClassificationNode instances are used to define tree structures where
 1289 each node in the tree is a ClassificationNode. Such *Classification* trees
 1290 are constructed with ClassificationNode instances under a
 1291 ClassificationScheme instance, and are used to define *Classification*
 1292 schemes or ontologies.
 1293
 1294

1295 **10.2.1 Attribute Summary**

1296

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

1297

1298 **10.2.2 Attribute parent**

1299 Each ClassificationNode may have a parent attribute. The parent attribute either
 1300 references a parent ClassificationNode or a ClassificationScheme instance in
 1301 case of first level ClassificationNode instances.
 1302

1303 **10.2.3 Attribute code**

1304 Each ClassificationNode may have a code attribute. The code attribute contains
 1305 a code within a standard coding scheme.
 1306

1307 **10.2.4 Method Summary**

1308 In addition to its attributes, the ClassificationNode class also defines the following
 1309 methods.
 1310

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.

1311

1312 In Figure 6, several instances of ClassificationNode are defined (all light colored
1313 boxes). A ClassificationNode has zero or one parent and zero or more
1314 ClassificationNodes for its immediate children. The parent of a
1315 ClassificationNode may be another ClassificationNode or a ClassificationScheme
1316 in case of first level ClassificationNodes.

1317

1318 10.2.5 Canonical Path Syntax

1319 The getPath method of the ClassificationNode class returns an absolute path in a
1320 canonical representation that uniquely identifies the path leading from the
1321 ClassificationScheme to that ClassificationNode.

1322 The canonical path representation is defined by the following BNF grammar:

1323

```
1324 canonicalPath ::= '/' schemeld nodePath
1325 nodePath    ::= '/' nodeCode
1326             | '/' nodeCode ( nodePath )?
```

1327

1328 In the above grammar, schemeld is the id attribute of the ClassificationScheme
1329 instance, and nodeCode is defined by NCName production as defined by
1330 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1331

1332 10.2.5.1 Example of Canonical Path Representation

1333 The following canonical path represents what the getPath method would return
1334 for the ClassificationNode with code 'United States' in the sample Geography
1335 scheme in section 10.2.5.2.

1336

```
1337 /Geography-id/NorthAmerica/UnitedStates
```


1338 10.2.5.2 Sample Geography Scheme

1339 Note that in the following examples, the ID attributes have been chosen for ease
1340 of readability and are therefore not valid URN or UUID values.

1341

```
1342 <ClassificationScheme id='Geography-id' name='Geography' />
```

1343

```
1344 <ClassificationNode id="NorthAmerica-id" parent="Geography-id" code="NorthAmerica" />
```

1345

```
1346 <ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
```

1347

```
1348 <ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
```

1349

```
1350 <ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
```

1351 10.3 Class Classification

1352 Base Classes:

1353 [RegistryObject](#)

1354

1355 A Classification instance classifies a RegistryObject instance by referencing a
1356 node defined within a particular classification scheme. An internal classification
1357 will always reference the node directly, by its id, while an external classification
1358 will reference the node indirectly by specifying a representation of its value that is
1359 unique within the external classification scheme.

1360

1361 The attributes and methods for the Classification class are intended to allow for
1362 representation of both internal and external classifications in order to minimize
1363 the need for a submission or a query to distinguish between internal and external
1364 classifications.

1365

1366 In Figure 6, Classification instances are not explicitly shown but are implied as
1367 associations between the RegistryObject instances (shaded leaf node) and the
1368 associated ClassificationNode.

1369 10.3.1 Attribute Summary

1370

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

1371 Note that attributes inherited from the base classes of this class are not shown.

1372

1373 10.3.2 Attribute classificationScheme

1374 If the Classification instance represents an external classification, then the
1375 classificationScheme attribute is required. The classificationScheme value must
1376 reference a ClassificationScheme instance.
1377

1378 10.3.3 Attribute classificationNode

1379 If the Classification instance represents an internal classification, then the
1380 classificationNode attribute is required. The classificationNode value must
1381 reference a ClassificationNode instance.

1382 10.3.4 Attribute classifiedObject

1383 For both internal and external classifications, the ClassifiedObject attribute is
1384 required and it references the RegistryObject instance that is classified by this
1385 Classification.
1386

1387 10.3.5 Attribute nodeRepresentation

1388 If the Classification instance represents an external classification, then the
1389 nodeRepresentation attribute is required. It is a representation of a taxonomy
1390 element from a classification scheme. It is the responsibility of the registry to
1391 distinguish between different types of nodeRepresentation, like between the
1392 classification scheme node code and the classification scheme node canonical
1393 path. This allows client to transparently use different syntaxes for
1394 nodeRepresentation.

1395 10.3.6 Context Sensitive Classification

1396 Consider the case depicted in Figure 9 where a *Collaboration Protocol Profile* for
1397 ACME Inc. is classified by the Japan ClassificationNode under the Geography
1398 *Classification* scheme. In the absence of the context for this *Classification* its
1399 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it
1400 mean that ACME ships products to Japan, or does it have some other meaning?
1401 To address this ambiguity a Classification may optionally be associated with
1402 another ClassificationNode (in this example named isLocatedIn) that provides the
1403 missing context for the Classification. Another *Collaboration Protocol Profile* for
1404 MyParcelService may be classified by the Japan ClassificationNode where this
1405 Classification is associated with a different ClassificationNode (e.g., named
1406 shipsTo) to indicate a different context than the one used by ACME Inc.

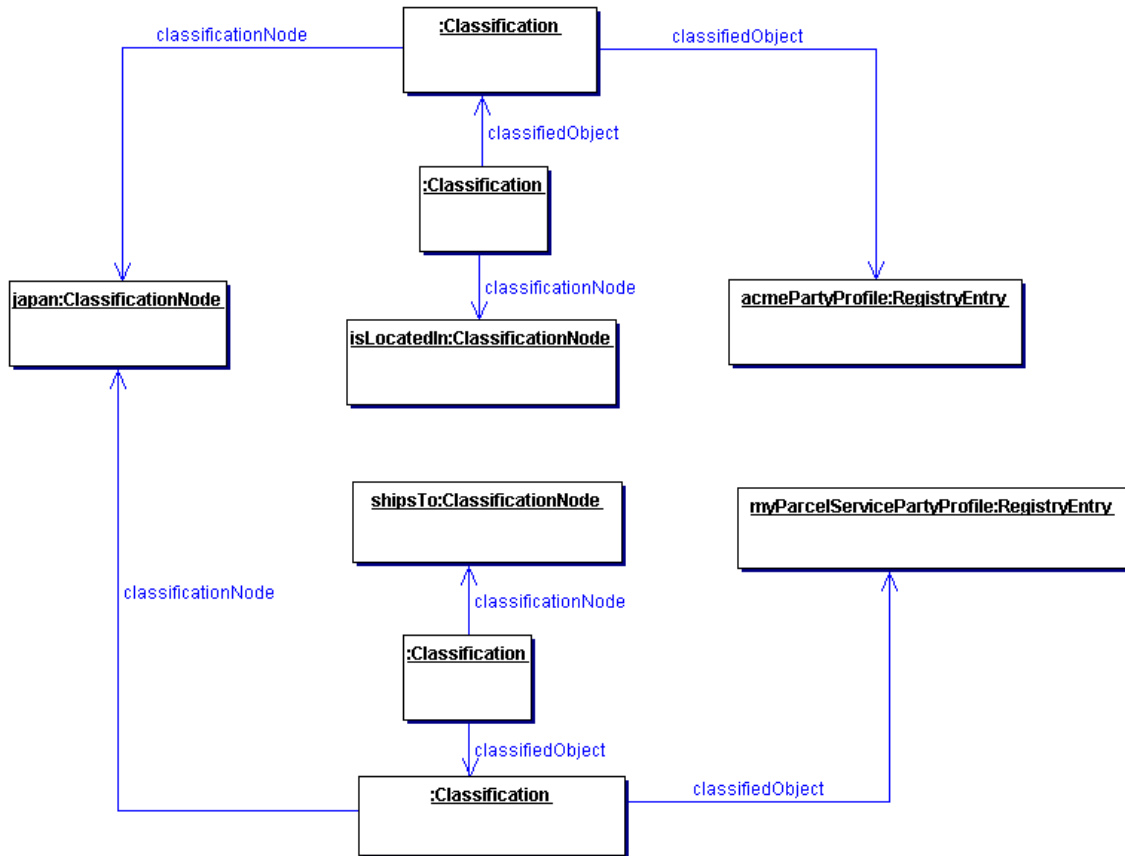


Figure 9: Context Sensitive Classification

1407

1408

1409

1410

1411

1412 Thus, in order to support the possibility of Classification within multiple contexts,
 1413 a Classification is itself classified by any number of Classifications that bind the
 1414 first Classification to ClassificationNodes that provide the missing contexts.

1415

1416 In summary, the generalized support for Classification schemes in the
 1417 information model allows:

- 1418 ○ A RegistryObject to be classified by defining an internal Classification that
 1419 associates it with a ClassificationNode in a ClassificationScheme.
- 1420 ○ A RegistryObject to be classified by defining an external Classification that
 1421 associates it with a value in an external ClassificationScheme.
- 1422 ○ A RegistryObject to be classified along multiple facets by having multiple
 1423 Classifications that associate it with multiple ClassificationNodes or value
 1424 within a ClassificationScheme.
- 1425 ○ A Classification defined for a RegistryObject to be qualified by the
 1426 contexts in which it is being classified.

1427

1428

1429 **10.3.7 Method Summary**

1430 In addition to its attributes, the Classification class also defines the following
 1431 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>

1432
 1433
 1434
 1435
 1436
 1437
 1438
 1439
 1440
 1441
 1442
 1443

1444 **10.4 Example of Classification Schemes**

1445 The following table lists some examples of possible *Classification* schemes
 1446 enabled by the information model. These schemes are based on a subset of
 1447 contextual concepts identified by the ebXML Business Process and Core
 1448 Components Project Teams. This list is meant to be illustrative not prescriptive.

1449
 1450

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"	

1451

Table 1: Sample Classification Schemes

1452

1453

1454

1455

1456

1457

1458

1459 **11 Information Model: Security View**

1460 This section describes the aspects of the information model that relate to the
 1461 security features of the *Registry*.

1462

1463 Figure 10 shows the view of the objects in the *Registry* from a security
 1464 perspective. It shows object relationships as a *UML Class* diagram. It does not
 1465 show *Class* attributes or *Class* methods that will be described in subsequent
 1466 sections. It is meant to be illustrative not prescriptive.

1467

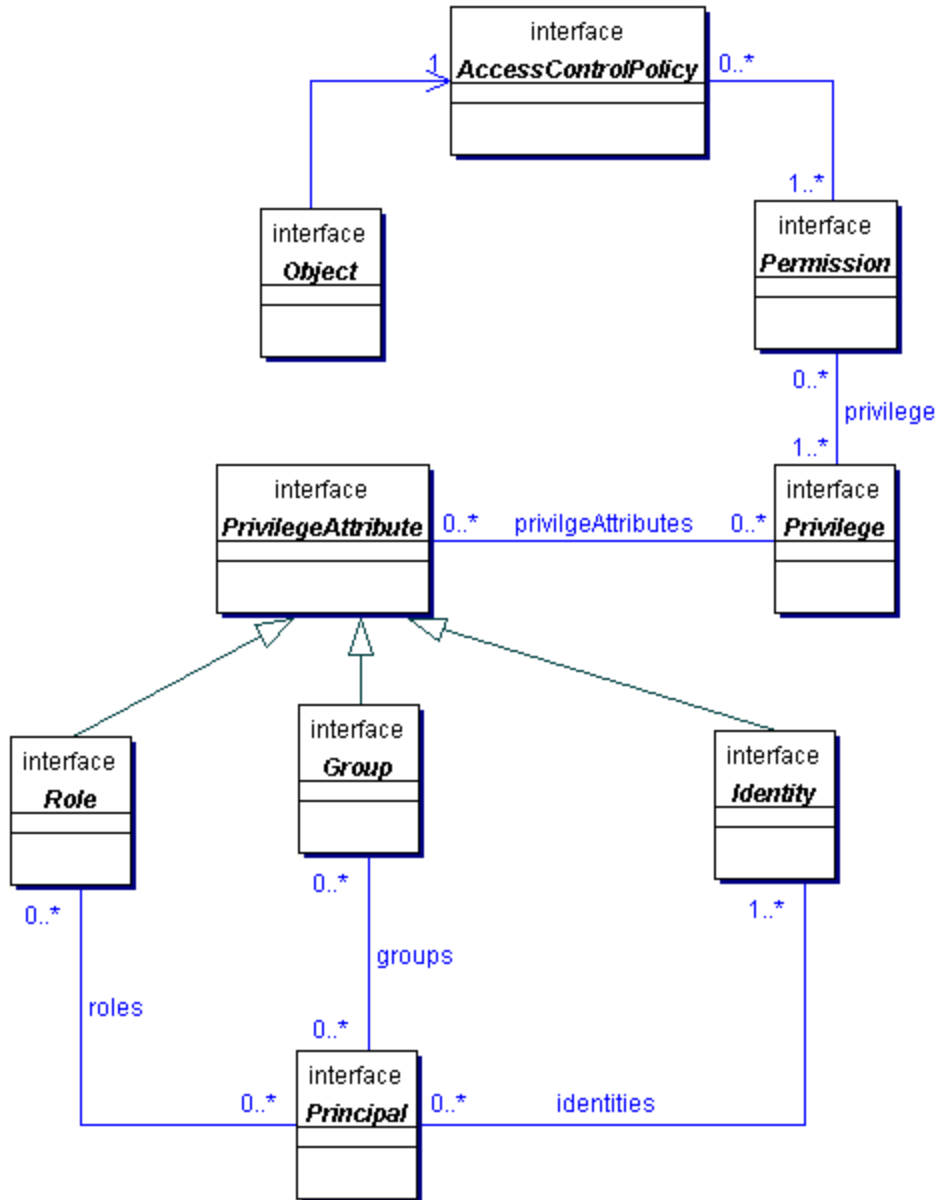


Figure 10: Information Model: Security View

1468
 1469
 1470
 1471
 1472
 1473
 1474
 1475
 1476
 1477
 1478

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.

1479

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

1480

1481 11.2 Class Permission

1482

1483 The Permission object is used for authorization and access control to
 1484 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are
 1485 defined in an AccessControlPolicy object.

1486

1487 A Permission object authorizes access to a method in a RegistryObject if the
 1488 requesting Principal has any of the Privileges defined in the Permission.

1489 **See Also:**

1490 [Privilege](#), [AccessControlPolicy](#)

1491

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> .

1492

1493 11.3 Class Privilege

1494

1495 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute
 1496 can be a Group, a Role, or an Identity.

1497

1498 A requesting Principal **MUST** have all of the PrivilegeAttributes specified in a
 1499 Privilege in order to gain access to a method in a protected RegistryObject.
 1500 Permissions defined in the RegistryObject's AccessControlPolicy define the
 1501 Privileges that can authorize access to specific methods.

1502

1503 This mechanism enables the flexibility to have object access control policies that
 1504 are based on any combination of Roles, Identities or Groups.

1505 **See Also:**

1506 [PrivilegeAttribute](#), [Permission](#)

1507

1508

1509

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

1510

1511 11.4 Class PrivilegeAttribute

1512 **All Known Subclasses:**

1513 [Group](#), [Identity](#), [Role](#)

1514

1515 PrivilegeAttribute is a common base *Class* for all types of security attributes that
1516 are used to grant specific access control privileges to a Principal. A Principal may
1517 have several different types of PrivilegeAttributes. Specific combination of
1518 PrivilegeAttributes may be defined as a Privilege object.

1519 **See Also:**

1520 [Principal](#), [Privilege](#)

1521 11.5 Class Role

1522 **All Superclasses:**

1523 [PrivilegeAttribute](#)

1524

1525 11.5.1 A security Role PrivilegeAttribute

1526 For example a hospital may have *Roles* such as Nurse, Doctor, Administrator
1527 etc. Roles are used to grant Privileges to Principals. For example a Doctor *Role*
1528 may be allowed to write a prescription but a Nurse *Role* may not.

1529 11.6 Class Group

1530 **All Superclasses:**

1531 [PrivilegeAttribute](#)

1532

1533 11.6.1 A security Group PrivilegeAttribute

1534 A Group is an aggregation of users that may have different Roles. For example
1535 a hospital may have a Group defined for Nurses and Doctors that are
1536 participating in a specific clinical trial (e.g., AspirinTrial group). Groups are used
1537 to grant Privileges to Principals. For example the members of the AspirinTrial
1538 group may be allowed to write a prescription for Aspirin (even though Nurse Role
1539 as a rule may not be allowed to write prescriptions).

1540

1541

1542 **11.7 Class Identity**1543 **All Superclasses:**1544 [PrivilegeAttribute](#)

1545

1546 **11.7.1 A security Identity PrivilegeAttribute**

1547 This is typically used to identify a person, an organization, or software service.

1548 Identity attribute may be in the form of a digital certificate.

1549 **11.8 Class Principal**

1550

1551 Principal is a generic term used by the security community to include both people

1552 and software systems. The Principal object is an entity that has a set of

1553 PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and

1554 optionally a set of role memberships, group memberships or security clearances.

1555 A principal is used to authenticate a requestor and to authorize the requested

1556 action based on the PrivilegeAttributes associated with the Principal.

1557 **See Also:**1558 PrivilegeAttributes, [Privilege](#), [Permission](#)

1559

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> .

1560

1561

1561 **12 References**

- 1562 [ebGLOSS] ebXML Glossary,
1563 http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 1564 [OAS] OASIS Information Model
1565 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>
- 1566 [ISO] ISO 11179 Information Model
1567 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 1569 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use
1570 in RFCs to Indicate Requirement Levels
1571 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>
- 1572 [ebRS] ebXML Registry Services Specification
1573 <http://www.oasisopen.org/committees/regrep/documents/2.0/specs/ebRS.pdf>
1574 [pdf](http://www.oasisopen.org/committees/regrep/documents/2.0/specs/ebRS.pdf)
- 1575 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
1576 <http://www.ebxml.org/specrafts/>
1577
- 1578 [UUID] DCE 128 bit Universal Unique Identifier
1579 http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
1580 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
1581 [TR/REC-xml](http://www.w3.org/TR/REC-xml)
1582
- 1583 [XPath] XML Path Language (XPath) Version 1.0
1584 <http://www.w3.org/TR/xpath>
1585
- 1586 [NCName] Namespaces in XML 19990114
1587 <http://www.w3.org/TR/REC-xml-names/#NT-NCName>.

1588 **13 Disclaimer**

- 1589 The views and specification expressed in this document are those of the authors
1590 and are not necessarily those of their employers. The authors and their
1591 employers specifically disclaim responsibility for any problems arising from
1592 correct or incorrect implementation or use of this design.
1593

1593 **14 Contact Information**

1594

1595 Team Leader

1596 Name: Lisa Carnahan
1597 Company: NIST
1598 Street: 100 Bureau Drive STOP 8970
1599 City, State, Postal Code: Gaithersburg, MD 20899-8970
1600 Country: USA
1601 Phone: (301) 975-3362
1602 Email: lisa.carnahan@nist.gov

1603

1604 Editor

1605 Name: Sally Fuger
1606 Company: Automotive Industry Action Group
1607 Street: 26200 Lahser Road, Suite 200
1608 City, State, Postal Code: Southfield, MI 48034
1609 Country: USA
1610 Phone: (248) 358-9744
1611 Email: sfuger@aiag.org

1612

1613 Technical Editor

1614 Name: Farrukh S. Najmi
1615 Company: Sun Microsystems
1616 Street: 1 Network Dr., MS BUR02-302
1617 City, State, Postal Code: Burlington, MA, 01803-0902
1618 Country: USA
1619 Phone: (781) 442-0703
1620 Email: najmi@east.sun.com

1621

1622

1622 Copyright Statement

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."