



Creating A Single Global Electronic Market

1

2

3

4

5 **ebXML Registry Information Model v1.0**

6 **ebXML Registry Project Team**

7 **8 May 2001**

8

9 **1 Status of this Document**

10

11 This document specifies an ebXML DRAFT STANDARD for the *eBusiness*
12 community.

13

14 Distribution of this document is unlimited.

15

16 The document formatting is based on the Internet Society's Standard RFC
17 format.

18

19 ***This version:***

20 <http://www.ebxml.org/specs/ebRIM.pdf>

21

22 ***Latest version:***

23 <http://www.ebxml.org/specs/ebRIM.pdf>

24

25

26

26 **2 ebXML participants**

27 We would like to recognize the following for their significant participation to the
28 development of this document.

29

30 Lisa Carnahan, NIST

31 Joe Dalman, Tie

32 Philippe DeSmedt, Viquity

33 Sally Fuger, AIAG

34 Len Gallagher, NIST

35 Steve Hanna, Sun Microsystems

36 Scott Hinkelman, IBM

37 Michael Kass, NIST

38 Jong.L Kim, Innodigital

39 Kyu-Chul Lee, Chungnam National University

40 Sangwon Lim, Korea Institute for Electronic Commerce

41 Bob Miller, GXS

42 Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan

43 Dale Moberg, Sterling Commerce

44 Ron Monzillo, Sun Microsystems

45 JP Morgenthal, eThink Systems, Inc.

46 Joel Munter, Intel

47 Farrukh Najmi, Sun Microsystems

48 Scott Nieman, Norstan Consulting

49 Frank Olken, Lawrence Berkeley National Laboratory

50 Michael Park, eSum Technologies

51 Bruce Peat, eProcess Solutions

52 Mike Rowley, Excelon Corporation

53 Waqar Sadiq, Vitria

54 Krishna Sankar, Cisco Systems Inc.

55 Kim Tae Soo, Government of Korea

56 Nikola Stojanovic, Encoda Systems, Inc.

57 David Webber, XML Global

58 Yutaka Yoshida, Sun Microsystems

59 Prasad Yendluri, webmethods

60 Peter Z. Zhoo, Knowledge For the new Millennium

61

62

62 **Table of Contents**

63

64 **1 STATUS OF THIS DOCUMENT 1**

65 **2 EBXML PARTICIPANTS 2**

66 **3 INTRODUCTION..... 6**

67 3.1 SUMMARY OF CONTENTS OF DOCUMENT 6

68 3.2 GENERAL CONVENTIONS 6

69 3.2.1 *Naming Conventions*..... 7

70 3.3 AUDIENCE..... 7

71 3.4 RELATED DOCUMENTS..... 7

72 **4 DESIGN OBJECTIVES 7**

73 4.1 GOALS 7

74 **5 SYSTEM OVERVIEW 8**

75 5.1 ROLE OF EBXML *REGISTRY* 8

76 5.2 *REGISTRY SERVICES*..... 8

77 5.3 WHAT THE REGISTRY INFORMATION MODEL DOES 8

78 5.4 HOW THE REGISTRY INFORMATION MODEL WORKS..... 8

79 5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED 9

80 5.6 *CONFORMANCE AS AN EBXML REGISTRY*..... 9

81 **6 REGISTRY INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW..... 9**

82 6.1 REGISTRYENTRY..... 10

83 6.2 SLOT 10

84 6.3 ASSOCIATION 11

85 6.4 EXTERNALIDENTIFIER..... 11

86 6.5 EXTERNALLINK 11

87 6.6 CLASSIFICATIONNODE..... 11

88 6.7 CLASSIFICATION 11

89 6.8 PACKAGE 11

90 6.9 AUDITABLEEVENT 11

91 6.10 USER..... 12

92 6.11 POSTALADDRESS 12

93 6.12 ORGANIZATION..... 12

94 **7 REGISTRY INFORMATION MODEL: DETAIL VIEW 12**

95 7.1 INTERFACE REGISTRYOBJECT 13

96 7.2 INTERFACE VERSIONABLE..... 15

97 7.3 INTERFACE REGISTRYENTRY 15

98 7.3.1 *Pre-defined RegistryEntry Status Types* 17

99 7.3.2 *Pre-defined Object Types*..... 18

100	7.3.3	<i>Pre-defined RegistryEntry Stability Enumerations</i>	19
101	7.4	INTERFACE SLOT.....	19
102	7.5	INTERFACE EXTRINSICOBJECT.....	20
103	7.6	INTERFACE INTRINSICOBJECT.....	21
104	7.7	INTERFACE PACKAGE.....	21
105	7.8	INTERFACE EXTERNALIDENTIFIER.....	22
106	7.9	INTERFACE EXTERNALLINK.....	22
107	8	REGISTRY AUDIT TRAIL	23
108	8.1	INTERFACE AUDITABLEEVENT.....	23
109	8.1.1	<i>Pre-defined Auditable Event Types</i>	24
110	8.2	INTERFACE USER.....	24
111	8.3	INTERFACE ORGANIZATION.....	25
112	8.4	CLASS POSTALADDRESS.....	26
113	8.5	CLASS TELEPHONENUMBER.....	26
114	8.6	CLASS PERSONNAME.....	27
115	9	REGISTRYENTRY NAMING	27
116	10	ASSOCIATION OF REGISTRYENTRY	28
117	10.1	INTERFACEASSOCIATION.....	28
118	10.1.1	<i>Pre-defined Association Types</i>	29
119	11	CLASSIFICATION OF REGISTRYENTRY	30
120	11.1	INTERFACE CLASSIFICATIONNODE.....	32
121	11.2	INTERFACE CLASSIFICATION.....	33
122	11.2.1	<i>Context Sensitive Classification</i>	34
123	11.3	EXAMPLE OF CLASSIFICATION SCHEMES.....	35
124	11.4	STANDARDIZED TAXONOMY SUPPORT.....	35
125	11.4.1	<i>Full-featured Taxonomy Based Classification</i>	36
126	11.4.2	<i>Light Weight Taxonomy Based Classification</i>	36
127	12	INFORMATION MODEL: SECURITY VIEW	37
128	12.1	INTERFACE ACCESSCONTROLPOLICY.....	38
129	12.2	INTERFACE PERMISSION.....	38
130	12.3	INTERFACE PRIVILEGE.....	38
131	12.4	INTERFACE PRIVILEGEATTRIBUTE.....	39
132	12.5	INTERFACE ROLE.....	39
133	12.6	INTERFACE GROUP.....	39
134	12.7	INTERFACE IDENTITY.....	40
135	12.8	INTERFACE PRINCIPAL.....	40
136	13	REFERENCES	41
137	14	DISCLAIMER	41
138	15	CONTACT INFORMATION	42

139 **COPYRIGHT STATEMENT** 43

140 **Table of Figures**

141 Figure 1: Information Model Public View10

142 Figure 2: Information Model Inheritance View 13

143 Figure 3: Example of Registry Entry Association28

144 Figure 4: Example showing a Classification Tree31

145 Figure 5: Information Model Classification View32

146 Figure 6: Classification Instance Diagram32

147 Figure 7: Context Sensitive Classification35

148 Figure 8: Information Model: Security View37

149 **Table of Tables**

150 Table 1: Sample Classification Schemes35

151

152

152 3 Introduction

153 3.1 Summary of Contents of Document

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

155

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

159 3.2 General Conventions

- 160 o *UML* diagrams are used as a way to concisely describe concepts. They are
161 not intended to convey any specific *Implementation* or methodology
162 requirements.
- 163 o Interfaces are often used in *UML* diagrams. They are used instead of *Classes*
164 with attributes to provide an abstract definition without implying any specific
165 *Implementation*. Specifically, they do not imply that objects in the *Registry* will
166 be accessed directly via these interfaces. Objects in the *Registry* are
167 accessed via interfaces described in the ebXML Registry Services
168 Specification. Each get method in every interface has an explicit indication of
169 the attribute name that the get method maps to. For example getName
170 method maps to an attribute named `name`.
- 171 o The term “*repository item*” is used to refer to an object that has been
172 submitted to a Registry for storage and safekeeping (e.g. an XML document
173 or a DTD). Every repository item is described by a RegistryEntry instance.
- 174 o The term “RegistryEntry” is used to refer to an object that provides metadata
175 about a repository item.
- 176 o The term “RegistryObject” is used to refer to the base interface in the
177 information model to avoid the confusion with the common term “object”.
178 However, when the term “object” is used to refer to a *class* or an interface in
179 the information model, it may also mean RegistryObject because almost all
180 classes are descendants of RegistryObject.

181

182 The information model does not deal with the actual content of the repository. All
183 *Elements* of the information model represent metadata about the content and not
184 the content itself.

185

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

188

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

192

193 **3.2.1 Naming Conventions**

194

195 In order to enforce a consistent capitalization and naming convention in this
196 document, "Upper Camel Case" (*UCC*) and "Lower Camel Case" (*LCC*)
197 Capitalization styles are used in the following conventions

198

199

- Element name is in *UCC* convention
(example: <UpperCamelCaseElement/>).
- Attribute name is in *LCC* convention
(example: <UpperCamelCaseElement
lowerCamelCaseAttribute="Whatever"/>).
- *Class*, *Interface* names use *UCC* convention
(examples: *ClassificationNode*, *Versionable*).
- *Method* name uses *LCC* convention
(example: *getName()*, *setName()*)

200

201

202

203

204

205

206

207

208

209

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

210

3.3 Audience

211

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

213

214

- o Implementers of ebXML *Registry Services*
- o Implementers of ebXML *Registry Clients*

215

3.4 Related Documents

216

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

218

219

220

221

222

223

224

225

226

- a) ebXML Registry Services Specification [ebRS] - defines the actual
Registry Services based on this information model
- b) ebXML Collaboration-Protocol Profile and Agreement Specification
[ebCPP] - defines how profiles can be defined for a *Party* and how two
Parties' profiles may be used to define a *Party* agreement
- c) ebXML Business Process Specification Schema [ebBPSS]
- d) ebXML Technical Architecture Specification [ebTA]

227

4 Design Objectives

228

4.1 Goals

229

The goals of this version of the specification are to:

- 230 o Communicate what information is in the *Registry* and how that information is
- 231 organized
- 232 o Leverage as much as possible the work done in the *OASIS* [OAS] and the
- 233 *ISO 11179* [ISO] Registry models
- 234 o Align with relevant works within other ebXML working groups
- 235 o Be able to evolve to support future ebXML *Registry* requirements
- 236 o Be compatible with other ebXML specifications
- 237

238 **5 System Overview**

239 **5.1 Role of ebXML Registry**

240
241 The *Registry* provides a stable store where information submitted by a

242 *Submitting Organization* is made persistent. Such information is used to facilitate

243 ebXML-based *Business to Business* (B2B) partnerships and transactions.

244 Submitted content may be *XML* schema and documents, process descriptions,

245 *Core Components*, context descriptions, *UML* models, information about parties

246 and even software components.

247 **5.2 Registry Services**

248 A set of *Registry Services* that provide access to *Registry* content to clients of the

249 *Registry* is defined in the ebXML Registry Services Specification [ebRS]. This

250 document does not provide details on these services but may occasionally refer

251 to them.

252 **5.3 What the Registry Information Model Does**

253 The Registry Information Model provides a blueprint or high-level schema for the

254 ebXML *Registry*. Its primary value is for implementers of ebXML *Registries*. It

255 provides these implementers with information on the type of metadata that is

256 stored in the *Registry* as well as the relationships among metadata *Classes*.

257 The Registry information model:

- 258 o Defines what types of objects are stored in the *Registry*
- 259 o Defines how stored objects are organized in the *Registry*
- 260 o Is based on ebXML metamodels from various working groups
- 261

262 **5.4 How the Registry Information Model Works**

263 Implementers of the ebXML *Registry* MAY use the information model to

264 determine which *Classes* to include in their *Registry Implementation* and what

265 attributes and methods these *Classes* may have. They MAY also use it to

266 determine what sort of database schema their *Registry Implementation* may
267 need.

268 [Note]The information model is meant to be
269 illustrative and does not prescribe any
270 specific *Implementation* choices.
271

272 **5.5 Where the Registry Information Model May Be Implemented**

273 The Registry Information Model MAY be implemented within an ebXML *Registry*
274 in the form of a relational database schema, object database schema or some
275 other physical schema. It MAY also be implemented as interfaces and *Classes*
276 within a *Registry Implementation*.

277 **5.6 Conformance to an ebXML Registry**

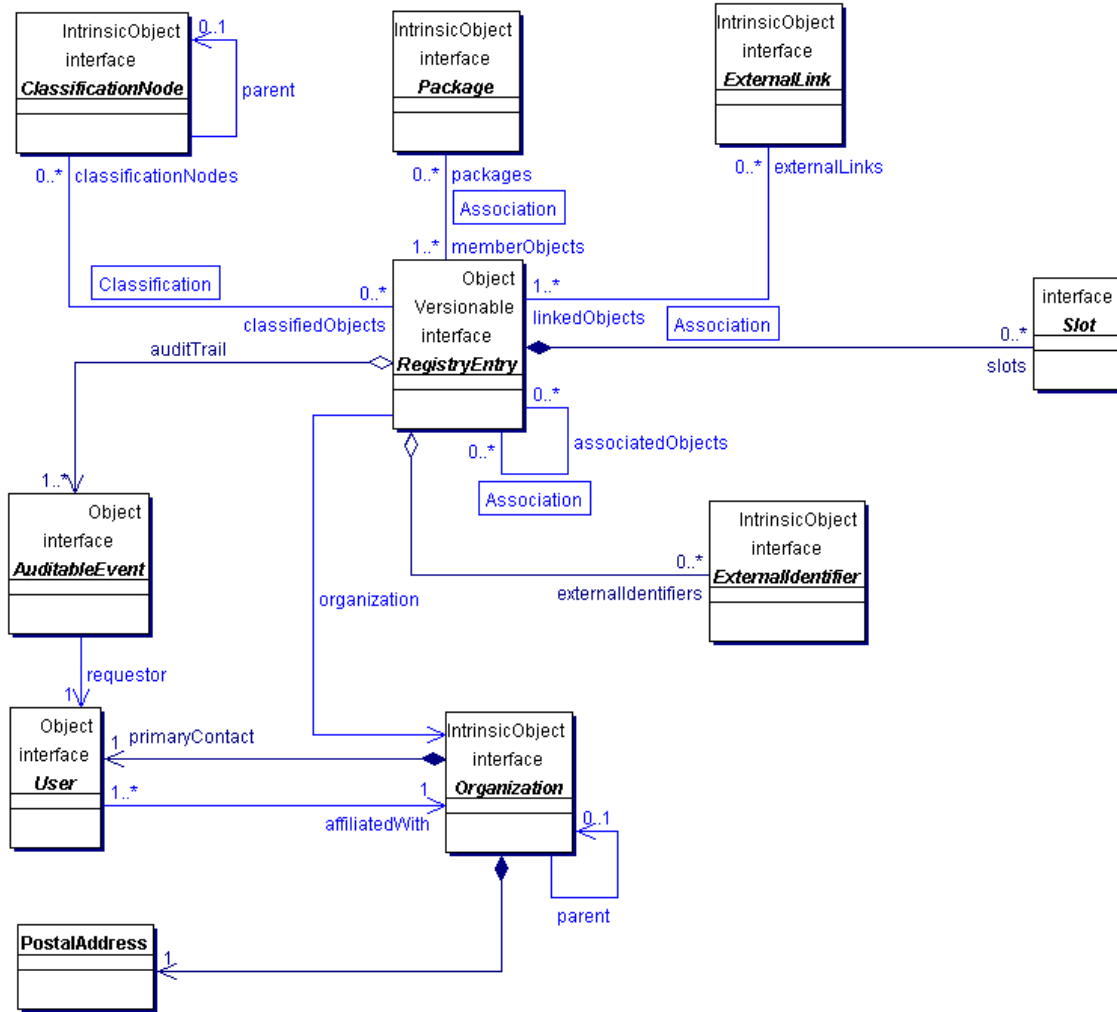
278
279 If an *Implementation* claims *Conformance* to this specification then it supports all
280 required information model *Classes* and interfaces, their attributes and their
281 semantic definitions that are visible through the ebXML *Registry Services*.

282 **6 Registry Information Model: High Level Public View**

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

285
286 Figure 1 shows the high level public view of the objects in the *Registry* and their
287 relationships as a *UML Class Diagram*. It does not show *Inheritance*, *Class*
288 attributes or *Class* methods.

289 The reader is again reminded that the information model is not modeling actual
290 repository items.
291



292

293

Figure 1: Information Model High Level Public View

294 **6.1 RegistryEntry**

295 The central object in the information model is a RegistryEntry. An Instance of
 296 RegistryEntry exists for each content Instance submitted to the Registry.
 297 Instances of the RegistryEntry Class provide metadata about a repository item.
 298 The actual repository item (e.g. a DTD) is not contained in an Instance of the
 299 RegistryEntry Class. Note that most Classes in the information model are
 300 specialized sub-classes of RegistryEntry. Each RegistryEntry is related to exactly
 301 one repository item.

302 **6.2 Slot**

303 Slot Instances provide a dynamic way to add arbitrary attributes to RegistryEntry
 304 Instances. This ability to add attributes dynamically to RegistryEntry Instances
 305 enables extensibility within the Registry Information Model.

306 **6.3 Association**

307 Association *Instances* are RegistryEntries that are used to define many-to-many
308 associations between objects in the information model. Associations are
309 described in detail in section 10.

310 **6.4 ExternalIdentifier**

311 ExternalIdentifier *Instances* provide additional identifier information to
312 RegistryEntry such as DUNS number, Social Security Number, or an alias name
313 of the organization.

314 **6.5 ExternalLink**

315 ExternalLink *Instances* are RegistryEntries that model a named URI to content
316 that is not managed by the *Registry*. Unlike managed content, such external
317 content may change or be deleted at any time without the knowledge of the
318 *Registry*. RegistryEntry may be associated with any number of ExternalLinks.
319 Consider the case where a *Submitting Organization* submits a repository item
320 (e.g. a *DTD*) and wants to associate some external content to that object (e.g.
321 the *Submitting Organization's* home page). The ExternalLink enables this
322 capability. A potential use of the ExternalLink capability may be in a GUI tool that
323 displays the ExternalLinks to a RegistryEntry. The user may click on such links
324 and navigate to an external web page referenced by the link.

325 **6.6 ClassificationNode**

326 ClassificationNode *Instances* are RegistryEntries that are used to define tree
327 structures where each node in the tree is a ClassificationNode. *Classification*
328 trees constructed with ClassificationNodes are used to define *Classification*
329 schemes or ontologies. ClassificationNode is described in detail in section 11.

330 **6.7 Classification**

331 Classification *Instances* are RegistryEntries that are used to classify repository
332 items by associating their RegistryEntry *Instance* with a ClassificationNode within
333 a *Classification* scheme. Classification is described in detail in section 11.

334 **6.8 Package**

335 Package *Instances* are RegistryEntries that group logically related
336 RegistryEntries together. One use of a Package is to allow operations to be
337 performed on an entire *Package* of objects. For example all objects belonging to
338 a Package may be deleted in a single request.

339 **6.9 AuditableEvent**

340 AuditableEvent *Instances* are Objects that are used to provide an audit trail for
341 RegistryEntries. AuditableEvent is described in detail in section 8.

342 **6.10 User**

343 User *Instances* are Objects that are used to provide information about registered
344 users within the *Registry*. User objects are used in audit trail for RegistryEntries.
345 User is described in detail in section 8.
346

347 **6.11 PostalAddress**

348 PostalAddress is a simple reusable *Entity Class* that defines attributes of a postal
349 address.
350

351 **6.12 Organization**

352 Organization *Instances* are RegistryEntries that provide information on
353 organizations such as a *Submitting Organization*. Each Organization *Instance*
354 may have a reference to a parent Organization.

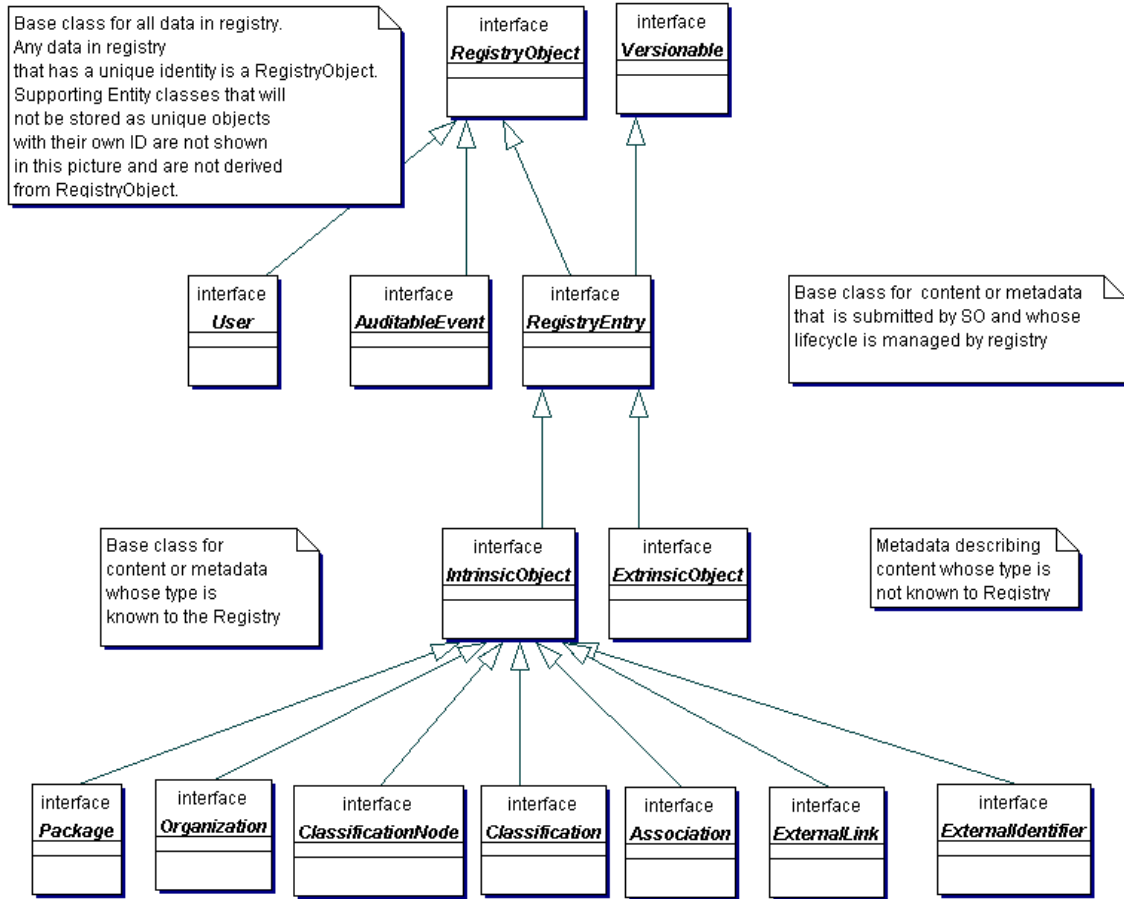
355 **7 Registry Information Model: Detail View**

356 This section covers the information model *Classes* in more detail than the Public
357 View. The detail view introduces some additional *Classes* within the model that
358 were not described in the public view of the information model.
359

360 Figure 2 shows the *Inheritance* or “is a” relationships between the *Classes* in the
361 information model. Note that it does not show the other types of relationships,
362 such as “has a” relationships, since they have already been shown in a previous
363 figure. *Class* attributes and *class* methods are also not shown. Detailed
364 description of methods and attributes of most interfaces and *Classes* will be
365 displayed in tabular form following the description of each *Class* in the model.
366

367 The interface Association will be covered in detail separately in section 10. The
368 interfaces Classification and ClassificationNode will be covered in detail
369 separately in section 11.
370

371 The reader is again reminded that the information model is not modeling actual
372 repository items.



373
374
375

Figure 2: Information Model *Inheritance View*

376 **7.1 Interface RegistryObject**

377 **All Known Subinterfaces:**

- 378 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
 379 [ExtrinsicObject](#), [IntrinsicObject](#), [RegistryEntry](#), [Organization](#), [Package](#),
 380 [User](#), [AuditableEvent](#), [ExternalIdentifier](#)

381
 382 RegistryObject provides a common base interface for almost all objects in the
 383 information model. Information model *Classes* whose *Instances* have a unique
 384 identity and an independent life cycle are descendants of the RegistryObject
 385 *Class*.

386
 387 Note that Slot and PostalAddress are not descendants of the RegistryObject
 388 *Class* because their *Instances* do not have an independent existence and unique
 389 identity. They are always a part of some other *Class's Instance* (e.g. Organization
 390 has a PostalAddress).

391
392

Method Summary of RegistryObject	
AccessControlPolicy	<p>getAccessControlPolicy()</p> <p>Gets the AccessControlPolicy object associated with this RegistryObject. An AccessControlPolicy defines the <i>Security Model</i> associated with the RegistryObject in terms of “who is permitted to do what” with that RegistryObject. Maps to attribute named <code>accessControlPolicy</code>.</p>
String	<p>getDescription()</p> <p>Gets the context independent textual description for this RegistryObject. Maps to attribute named <code>description</code>.</p>
String	<p>getName()</p> <p>Gets user friendly, context independent name for this RegistryObject. Maps to attribute named <code>name</code>.</p>
String	<p>getID()</p> <p>Gets the universally unique ID, as defined by [UUID], for this RegistryObject. Maps to attribute named <code>id</code>.</p>
void	<p>setDescription(String description)</p> <p>Sets the context, independent textual description for this RegistryObject.</p>
void	<p>setName(String name)</p> <p>Sets user friendly, context independent name for this RegistryObject.</p>
void	<p>setID(String id)</p> <p>Sets the universally unique ID, as defined by [UUID], for this RegistryObject.</p>

393
394

394 **7.2 Interface Versionable**

395 **All Known Subinterfaces:**

396 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
 397 [ExtrinsicObject](#), [IntrinsicObject](#), [RegistryEntry](#), [Organization](#), [Package](#),
 398 [ExternalIdentifier](#)

399
 400 The Versionable interface defines the behavior common to *Classes* that are
 401 capable of creating versions of their *Instances*. At present all RegistryEntry
 402 *Classes* are REQUIRED to implement the Versionable interface.
 403

Method Summary of Versionable	
int	getMajorVersion () Gets the major revision number for this version of the Versionable object. Maps to attribute named <code>majorVersion</code> .
int	getMinorVersion () Gets the minor revision number for this version of the Versionable object. Maps to attribute named <code>minorVersion</code> .
void	setMajorVersion (int majorVersion) Sets the major revision number for this version of the Versionable object.
void	setMinorVersion (int minorVersion) Sets the minor revision number for this version of the Versionable object.

404

405 **7.3 Interface RegistryEntry**

406 **All Superinterfaces:**

407 [RegistryObject](#), [Versionable](#)

408 **All Known Subinterfaces:**

409 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
 410 [ExtrinsicObject](#), [IntrinsicObject](#), [Organization](#), [Package](#), [ExternalIdentifier](#)

411
 412 RegistryEntry is a common base *Class* for all metadata describing submitted
 413 content whose life cycle is managed by the *Registry*. Metadata describing
 414 content submitted to the *Registry* is further specialized by the ExtrinsicObject and
 415 IntrinsicObject subclasses of RegistryEntry.
 416
 417
 418
 419

Method Summary of RegistryEntry	
Collection	<p>getAssociatedObjects() Returns the collection of RegistryObjects associated with this RegistryObject. Maps to attribute named <code>associatedObjects</code>.</p>
Collection	<p>getAuditTrail() Returns the complete audit trail of all requests that effected a state change in this RegistryObject as an ordered Collection of AuditableEvent objects. Maps to attribute named <code>auditTrail</code>.</p>
Collection	<p>getClassificationNodes() Returns the collection of ClassificationNodes associated with this RegistryObject. Maps to attribute named <code>classificationNodes</code>.</p>
Collection	<p>getExternalLinks() Returns the collection of ExternalLinks associated with this RegistryObject. Maps to attribute named <code>externalLinks</code>.</p>
Collection	<p>getExternalIdentifiers() Returns the collection of ExternalIdentifiers associated with this RegistryObject. Maps to attribute named <code>externalIdentifiers</code>.</p>
String	<p>getObjectType() Gets the pre-defined object type associated with this RegistryEntry. This SHOULD be the name of a object type as described in 7.3.2. Maps to attribute named <code>objectType</code>.</p>
Collection	<p>getOrganizations() Returns the collection of Organizations associated with this RegistryObject. Maps to attribute named <code>organizations</code>.</p>
Collection	<p>getPackages() Returns the collection of Packages associated with this RegistryObject. Maps to attribute named <code>packages</code>.</p>
String	<p>getStatus() Gets the life cycle status of the RegistryEntry within the <i>Registry</i>. This SHOULD be the name of a RegistryEntry status type as described in 7.3.1. Maps to attribute named <code>status</code>.</p>
String	<p>getUserVersion() Gets the userVersion attribute of the RegistryEntry within the <i>Registry</i>. The userVersion is the version for the RegistryEntry as assigned by the user.</p>
void	<p>setUserVersion(String UserVersion) Sets the userVersion attribute of the RegistryEntry within the <i>Registry</i>.</p>
String	<p>getStability() Gets the stability indicator for the RegistryEntry within the</p>

	<i>Registry</i> . The stability indicator is provided by the submitter as a guarantee of the level of stability for the content. This SHOULD be the name of a stability type as described in 7.3.3. Maps to attribute named <code>stability</code> .
Date	getExpirationDate() Gets expirationDate attribute of the RegistryEntry within the <i>Registry</i> . This attribute defines a time limit upon the stability guarantee provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes STABILITY_DYNAMIC implying that content can change at any time and in any manner. A null value implies that there is no expiration on stability attribute. Maps to attribute named <code>expirationDate</code> .
void	setExpirationDate(Date expirationDate) Sets expirationDate attribute of the RegistryEntry within the <i>Registry</i> .
Collection	getSlots() Gets the collection of slots that have been dynamically added to this RegistryObject. Maps to attribute named <code>slots</code> .
void	addSlots(Collection newSlots) Adds one or more slots to this RegistryObject. Slot names MUST be locally unique within this RegistryObject. Any existing slots are not effected.
void	removeSlots(Collection slotNames) Removes one or more slots from this RegistryObject. Slots to be removed are identified by their name.

420

Methods inherited from interface RegistryObject
getAccessControlPolicy , getDescription , getName , getID , setDescription , setName , setID

421

Methods inherited from interface Versionable
getMajorVersion , getMinorVersion , setMajorVersion , setMinorVersion

422 **7.3.1 Pre-defined RegistryEntry Status Types**

423 The following table lists pre-defined choices for RegistryEntry status attribute.
 424 These pre-defined status types are defined as a *Classification* scheme. While the
 425 scheme may easily be extended, a *Registry* MUST support the status types listed
 426 below.
 427

Name	Description
------	-------------

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

428 **7.3.2 Pre-defined Object Types**

429 The following table lists pre-defined object types. Note that for an ExtrinsicObject
 430 there are many types defined based on the type of repository item the
 431 ExtrinsicObject catalogs. In addition there there are object types defined for
 432 IntrinsicObject sub-classes that may have concrete *Instances*.

433
 434 These pre-defined object types are defined as a *Classification* scheme. While the
 435 scheme may easily be extended a *Registry* MUST support the object types listed
 436 below.

437

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.
Process	An ExtrinsicObject of this type catalogues a process description document.
Role	An ExtrinsicObject of this type catalogues an <i>XML</i> description of a <i>Role</i> in a <i>Collaboration Protocol Profile (CPP)</i> .
ServiceInterface	An ExtrinsicObject of this type catalogues an <i>XML</i> description of a service interface as defined by [ebCPP].
SoftwareComponent	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or <i>Class</i> library).

Transport	An ExtrinsicObject of this type catalogues an <i>XML</i> description of a transport configuration as defined by [ebCPP].
UMLModel	An ExtrinsicObject of this type catalogues a <i>UML</i> model.
XMLSchema	An ExtrinsicObject of this type catalogues an <i>XML</i> schema (<i>DTD</i> , <i>XML Schema</i> , <i>RELAX grammar</i> , etc.).
Package	A Package object
ExternalLink	An ExternalLink object
ExternalIdentifier	An ExternalIdentifier object
Association	An Association object
Classification	A Classification object
ClassificationNode	A ClassificationNode object
AuditableEvent	An AuditableEvent object
User	A User object
Organization	An Organization object

438

439 **7.3.3 Pre-defined RegistryEntry Stability Enumerations**

440 The following table lists pre-defined choices for RegistryEntry stability attribute.
 441 These pre-defined stability types are defined as a *Classification* scheme. While
 442 the scheme may easily be extended, a *Registry* MAY support the stability types
 443 listed below.

444

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.

445

446

447 **7.4 Interface Slot**

448

449 Slot *Instances* provide a dynamic way to add arbitrary attributes to RegistryEntry
 450 *Instances*. This ability to add attributes dynamically to RegistryEntry *Instances*
 451 enables extensibility within the Registry Information Model.

452

453 In this model, a RegistryEntry may have 0 or more Slots. A slot is composed of a
 454 name, a slotType and a collection of values. The name of slot is locally unique
 455 within the RegistryEntry *Instance*. Similarly, the value of a Slot is locally unique
 456 within a slot *Instance*. Since a Slot represent an extensible attribute whose value
 457 may be a collection, therefore a Slot is allowed to have a collection of values
 458 rather than a single value. The slotType attribute may optionally specify a type or
 459 category for the slot.
 460
 461

Method Summary of Slot	
String	getName() Gets the name of this RegistryObject. Maps to attribute named name.
void	setName(String name) Sets the name of this RegistryObject. Slot names are locally unique within a RegistryEntry <i>Instance</i> .
String	getSlotType() Gets the slotType or category for this slot. Maps to attribute named slotType.
void	setSlotType(String slotType) Sets the slotType or category for this slot.
Collection	getValues() Gets the collection of values for this RegistryObject. The type for each value is String. Maps to attribute named values.
void	setValues(Collection values) Sets the collection of values for this RegistryObject.

462

463 **7.5 Interface ExtrinsicObject**

464 **All Superinterfaces:**

465 [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

466

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

470

471 Examples of content described by ExtrinsicObject include *Collaboration Protocol*
 472 *Profiles (CPP)*, *Business Process* descriptions, and schemas.

473

Method Summary of Extrinsic Object	
String	getContentURI() Gets the URI to the content catalogued by this ExtrinsicObject. A <i>Registry</i> MUST guarantee that this URI is resolvable. Maps to attribute named <code>contentURI</code> .
String	getMimeType() Gets the mime type associated with the content catalogued by this ExtrinsicObject. Maps to attribute named <code>mimeType</code> .
boolean	isOpaque() Determines whether the content catalogued by this ExtrinsicObject is opaque to (not readable by) the <i>Registry</i> . In some situations, a <i>Submitting Organization</i> may submit content that is encrypted and not even readable by the <i>Registry</i> . Maps to attribute named <code>opaque</code> .
void	setContentURI(String uri) Sets the URI to the content catalogued by this ExtrinsicObject.
void	setMimeType(String mimeType) Sets the mime type associated with the content catalogued by this ExtrinsicObject.
void	setOpaque(boolean isOpaque) Sets whether the content catalogued by this ExtrinsicObject is opaque to (not readable by) the <i>Registry</i> .

474

475 Note that methods inherited from the base interfaces of this interface are not
 476 shown.

477 **7.6 Interface IntrinsicObject**

478 **All Superinterfaces:**

479 [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

480 **All Known Subinterfaces:**

481 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#), [Organization](#),
 482 [Package](#), [ExternalIdentifier](#)

483

484 IntrinsicObject serve as a common base *Class* for derived *Classes* that catalogue
 485 submitted content whose type is known to the *Registry* and defined by the
 486 ebXML *Registry* specifications.

487

488 This interface currently does not define any attributes or methods. Note that
 489 methods inherited from the base interfaces of this interface are not shown.

490

491 **7.7 Interface Package**

492 **All Superinterfaces:**

493 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

494
495
496
497
498
499

Logically related RegistryEntries may be grouped into a Package. It is anticipated that *Registry Services* will allow operations to be performed on an entire *Package* of objects in the future.

Method Summary of Package

Collection	<p>getMemberObjects() Get the collection of RegistryEntries that are members of this Package. Maps to attribute named <code>memberObjects</code>.</p>
------------	---

500

501 7.8 Interface ExternalIdentifier

502 **All Superinterfaces:**

503 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

504
505
506
507
508
509
510
511
512

ExternalIdentifier *Instances* provide the additional identifier information to RegistryEntry such as DUNS number, Social Security Number, or an alias name of the organization. The attribute *name* inherited from RegistryObject is used to contain the identification scheme (Social Security Number, etc), and the attribute *value* contains the actual information. Each RegistryEntry may have 0 or more association(s) with ExternalIdentifier.

See Also:

Method Summary of ExternalIdentifier

String	<p><code>getValue()</code> Gets the value of this ExternalIdentifier. Maps to attribute named <code>value</code>.</p>
Void	<p><code>setValue(String value)</code> Sets the value of this ExternalIdentifier.</p>

513
514
515

Note that methods inherited from the base interfaces of this interface are not shown.

516 7.9 Interface ExternalLink

517 **All Superinterfaces:**

518 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

519
520
521
522
523

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

524
525

Method Summary of ExternalLink	
Collection	getLinkedObjects () Gets the collection of RegistryObjects that use this external link. Maps to attribute named <code>linkedObjects</code> .
URI	getExternalURI () Gets URI to the external content. Maps to attribute named <code>externalURI</code> .
void	setExternalURI (URI uri) Sets URI to the external content.

526
527
528

Note that methods inherited from the base interfaces of this interface are not shown.

529 8 Registry Audit Trail

530 This section describes the information model *Elements* that support the audit trail
531 capability of the *Registry*. Several *Classes* in this section are *Entity Classes* that
532 are used as wrappers to model a set of related attributes. These *Entity Classes*
533 do not have any associated behavior. They are analogous to the “struct”
534 construct in the C programming language.

535
536
537
538
539
540
541

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

542 8.1 Interface AuditableEvent

543 **All Superinterfaces:**

544 [RegistryObject](#)

545
546
547
548
549
550

`AuditableEvent` *Instances* provide a long-term record of *Events* that effect a change of state in a `RegistryEntry`. A `RegistryEntry` is associated with an ordered `Collection` of `AuditableEvent` *Instances* that provide a complete audit trail for that `RegistryObject`.

551 `AuditableEvents` are usually a result of a client-initiated request. `AuditableEvent`
552 *Instances* are generated by the *Registry Service* to log such *Events*.

553
554
555
556

Often such *Events* effect a change in the life cycle of a `RegistryEntry`. For example a client request could Create, Update, Deprecate or Delete a `RegistryEntry`. No `AuditableEvent` is created for requests that do not alter the

557 state of a RegistryEntry. Specifically, read-only requests do not generate an
 558 AuditableEvent. No AuditableEvent is generated for a RegistryEntry when it is
 559 classified, assigned to a Package or associated with another RegistryObject.
 560
 561

562 **8.1.1 Pre-defined Auditable Event Types**

563 The following table lists pre-defined auditable event types. These pre-defined
 564 event types are defined as a *Classification* scheme. While the scheme may
 565 easily be extended, a *Registry* MUST support the event types listed below.
 566

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

567

Method Summary of AuditableEvent	
<u>User</u>	getUser() Gets the User that sent the request that generated this <i>Event</i> . Maps to attribute named <code>user</code> .
String	getEventType() The type of this <i>Event</i> as defined by the name attribute of an event type as defined in section 8.1.1. Maps to attribute named <code>eventType</code> .
<u>RegistryEntry</u>	getRegistryEntry() Gets the RegistryEntry associated with this AuditableEvent. Maps to attribute named <code>registryEntry</code> .
Timestamp	getTimestamp() Gets the Timestamp for when this <i>Event</i> occurred. Maps to attribute named <code>timestamp</code> .

568

569 Note that methods inherited from the base interfaces of this interface are not
 570 shown.

571 **8.2 Interface User**

572 **All Superinterfaces:**

573 [RegistryObject](#)

574

575 User *Instances* are used in an AuditableEvent to keep track of the identity of the
 576 requestor that sent the request that generated the AuditableEvent.
 577

Method Summary of User	
Organization	getOrganization() Gets the <i>Submitting Organization</i> that sent the request that effected this change. Maps to attribute named <code>organization</code> .
PostalAddress	getAddress() Gets the postal address for this user. Maps to attribute named <code>address</code> .
String	getEmail() Gets the email address for this user. Maps to attribute named <code>email</code> .
TelephoneNumber	getFax() The FAX number for this user. Maps to attribute named <code>fax</code> .
TelephoneNumber	getMobilePhone() The mobile telephone number for this user. Maps to attribute named <code>mobilePhone</code> .
PersonName	getPersonName() Name of contact person. Maps to attribute named <code>personName</code> .
TelephoneNumber	getPager() The pager telephone number for this user. Maps to attribute named <code>pager</code> .
TelephoneNumber	getTelephone() The default (land line) telephone number for this user. Maps to attribute named <code>telephone</code> .
URL	getUrl() The <i>URL</i> to the web page for this contact. Maps to attribute named <code>url</code> .

578

579 **8.3 Interface Organization**

580 **All Superinterfaces:**

581 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

582

583 Organization *Instances* provide information on organizations such as a
 584 *Submitting Organization*. Each Organization *Instance* may have a reference to a
 585 parent Organization. In addition it may have a contact attribute defining the
 586 primary contact within the organization. An Organization also has an address
 587 attribute.

588

Method Summary of Organization	
PostalAddress	<p>getAddress()</p> <p>Gets the PostalAddress for this Organization. Maps to attribute named <code>address</code>.</p>
User	<p>getPrimaryContact()</p> <p>Gets the primary Contact for this Organization. The primary contact is a reference to a User object. Maps to attribute named <code>primaryContact</code>.</p>
TelephoneNumber	<p>getFax()</p> <p>Gets the FAX number for this Organization. Maps to attribute named <code>fax</code>.</p>
Organization	<p>getParent()</p> <p>Gets the parent Organization for this Organization. Maps to attribute named <code>parent</code>.</p>
TelephoneNumber	<p>getTelephone()</p> <p>Gets the main telephone number for this Organization. Maps to attribute named <code>telephone</code>.</p>

589

590

Note that methods inherited from the base interfaces of this interface are not shown.

591

592

593

8.4 Class PostalAddress

594

595

596

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

597

598

Field Summary	
String	<p>city</p> <p>The city.</p>
String	<p>country</p> <p>The country.</p>
String	<p>postalCode</p> <p>The postal or zip code.</p>
String	<p>state</p> <p>The state or province.</p>
String	<p>street</p> <p>The street.</p>

599

600

8.5 Class TelephoneNumber

601

602
 603 A simple reusable *Entity Class* that defines attributes of a telephone number.
 604

Field Summary	
String	areaCode Area code.
String	countryCode country code.
String	extension internal extension if any.
String	number The telephone number suffix not including the country or area code.
String	url A <i>URL</i> that can dial this number electronically.

605

606 8.6 Class *PersonName*

607
 608 A simple *Entity Class* for a person's name.

609
 610

Field Summary	
String	firstName The first name for this person.
String	lastName The last name (surname) for this person.
String	middleName The middle name for this person.

611

612 9 RegistryEntry Naming

613 A RegistryEntry has a name that may or may not be unique within the *Registry*.

614

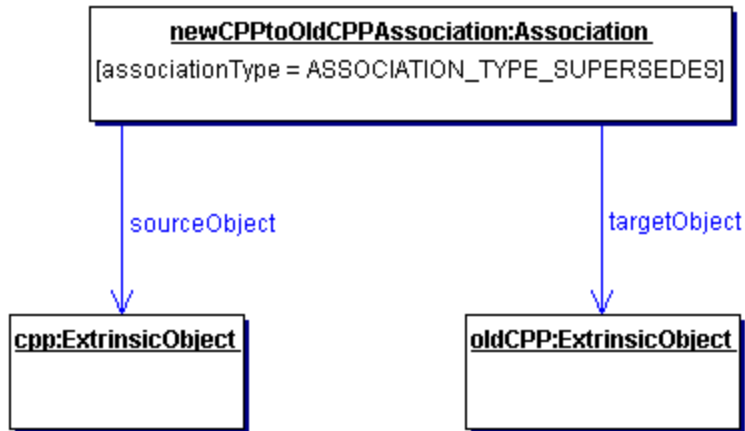
615 In addition a RegistryEntry may have any number of context sensitive alternate names that are valid only in the context of a particular *Classification* scheme.

616 Alternate contextual naming will be addressed in a later version of the Registry Information Model.

617
 618
 619

620 **10 Association of RegistryEntry**

621 A RegistryEntry may be associated with 0 or more RegistryObjects. The
 622 information model defines an Association *Class*. An *Instance* of the Association
 623 *Class* represents an association between a RegistryEntry and another
 624 RegistryObject. An example of such an association is between ExtrinsicObjects
 625 that catalogue a new *Collaboration Protocol Profile (CPP)* and an older
 626 *Collaboration Protocol Profile* where the newer *CPP* supersedes the older *CPP*
 627 as shown in Figure 3.



628

629

Figure 3: Example of RegistryEntry Association

630

631 **10.1 Interface Association**

632 **All Superinterfaces:**

633 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

634

635

636 Association *Instances* are used to define many-to-many associations between
 637 RegistryObjects in the information model.

638

639 An *Instance* of the Association *Class* represents an association between two
 640 RegistryObjects.

641

642

643

Method Summary of Association	
String	getAssociationType() Gets the association type for this Association. This MUST be the name attribute of an association type as defined by 10.1.1. Maps to attribute named <code>associationType</code> .
Object	getSourceObject()

	Gets the RegistryObject that is the source of this Association. Maps to attribute named <code>sourceObject</code> .
String	getSourceRole() Gets the name of the <i>Role</i> played by the source RegistryObject in this Association. Maps to attribute named <code>sourceRole</code> .
Object	getTargetObject() Gets the RegistryObject that is the target of this Association. Maps to attribute named <code>targetObject</code> .
String	getTargetRole() Gets the name of the <i>Role</i> played by the target RegistryObject in this Association. Maps to attribute named <code>targetRole</code> .
boolean	isBidirectional() Determine whether this Association is bi-directional. Maps to attribute named <code>bidirectional</code> .
void	setBidirectional(boolean bidirectional) Set whether this Association is bi-directional.
void	setSourceRole(String sourceRole) Sets the name of the <i>Role</i> played by the source RegistryObject in this Association.
void	setTargetRole(String targetRole) Sets the name of the <i>Role</i> played by the destination RegistryObject in this Association.

644 **10.1.1 Pre-defined Association Types**

645 The following table lists pre-defined association types. These pre-defined
 646 association types are defined as a *Classification* scheme. While the scheme may
 647 easily be extended a *Registry* MUST support the association types listed below.
 648

name	description
RelatedTo	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source Package object has the target RegistryEntry object as a member. Reserved for use in Packaging of RegistryEntries.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryEntry object. Reserved for use in associating ExternalLinks with RegistryEntries.
ExternallyIdentifies	Defines that the source ExternalIdentifier object identifies the target RegistryEntry object. Reserved for use in associating ExternalIdentifiers with RegistryEntries.

ContainedBy	Defines that source RegistryObject is contained by the target RegistryObject.
Contains	Defines that source RegistryObject contains 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.
SupersededBy	Defines that the source RegistryObject is superseded by the target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
UsedBy	Defines that the source RegistryObject is used by the target RegistryObject in some manner.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
ReplacedBy	Defines that the source RegistryObject is replaced by the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.

649

650 [Note] In some association types, such as Extends and
 651 Implements, although the association is between
 652 RegistryObjects, the actual relationship
 653 specified by that type is between repository
 654 items pointed by RegistryObjects.

655 **11 Classification of RegistryEntry**

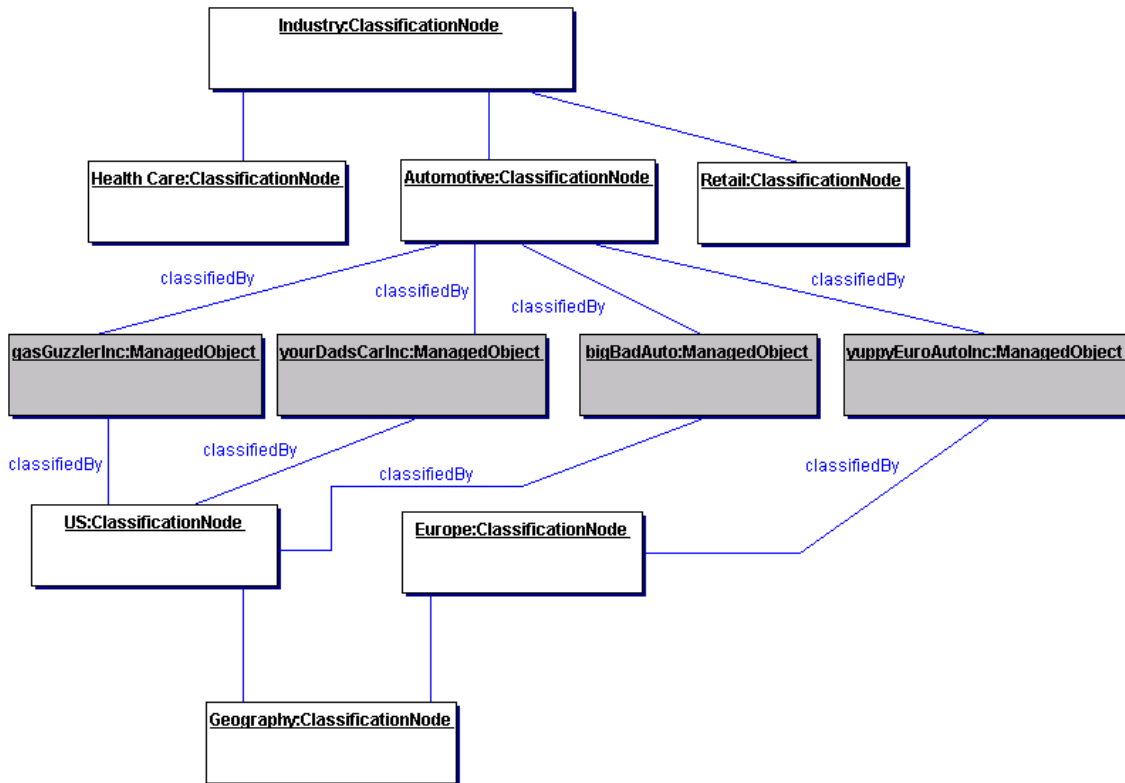
656 This section describes the how the information model supports *Classification* of
 657 RegistryEntry. It is a simplified version of the *OASIS* classification model [OAS].
 658

659 A RegistryEntry may be classified in many ways. For example the RegistryEntry
 660 for the same *Collaboration Protocol Profile (CPP)* may be classified by its
 661 industry, by the products it sells and by its geographical location.
 662

663 A general *Classification* scheme can be viewed as a *Classification* tree. In the
 664 example shown in Figure 4, RegistryEntries representing *Collaboration Protocol*
 665 *Profiles* are shown as shaded boxes. Each *Collaboration Protocol Profile*
 666 represents an automobile manufacturer. Each *Collaboration Protocol Profile* is
 667 classified by the ClassificationNode named Automotive under the root
 668 ClassificationNode named Industry. Furthermore, the US Automobile

669 manufacturers are classified by the US ClassificationNode under the Geography
 670 ClassificationNode. Similarly, a European automobile manufacturer is classified
 671 by the Europe ClassificationNode under the Geography ClassificationNode.
 672

673 The example shows how a RegistryEntry may be classified by multiple
 674 *Classification* schemes. A *Classification* scheme is defined by a
 675 ClassificationNode that is the root of a *Classification* tree (e.g. Industry,
 676 Geography).

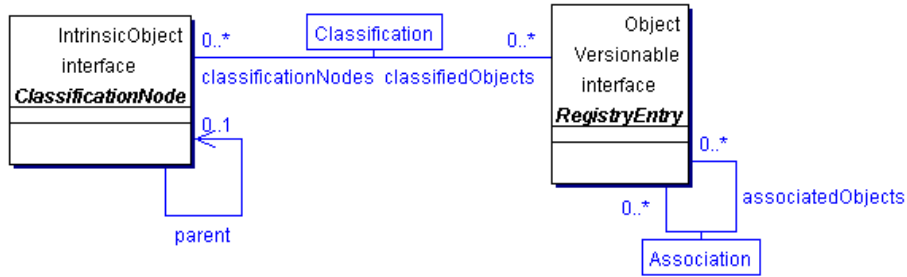


677
 678

Figure 4: Example showing a *Classification* Tree

679 [Note]It is important to point out that the dark
 680 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are
 681 not part of the *Classification* tree. The leaf
 682 nodes of the *Classification* tree are Health
 683 Care, Automotive, Retail, US and Europe. The
 684 dark nodes are associated with the
 685 *Classification* tree via a *Classification*
 686 *Instance* that is not shown in the picture
 687

688 In order to support a general *Classification* scheme that can support single level
 689 as well as multi-level *Classifications*, the information model defines the *Classes*
 690 and relationships shown in Figure 5.



691

692

Figure 5: Information Model *Classification* View

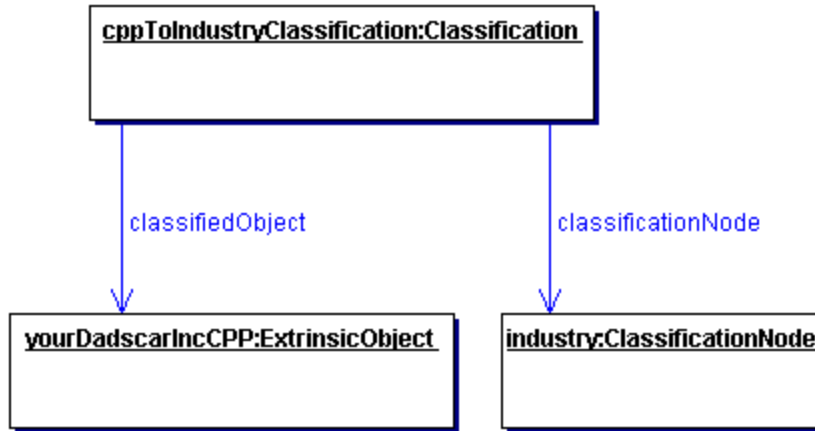
693

A *Classification* is a specialized form of an *Association*. Figure 6 shows an example of an *ExtrinsicObject Instance* for a *Collaboration Protocol Profile (CPP)* object that is classified by a *ClassificationNode* representing the Industry that it belongs to.

694

695

696



697

698

Figure 6: Classification *Instance* Diagram

699 11.1 Interface *ClassificationNode*

700 **All Superinterfaces:**

701 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

702

703 *ClassificationNode Instances* are used to define tree structures where each node
704 in the tree is a *ClassificationNode*. Such *Classification* trees constructed with
705 *ClassificationNodes* are used to define *Classification* schemes or ontologies.

706 **See Also:**

707 [Classification](#)

708

709

Method Summary of *ClassificationNode*

Collection	getClassifiedObjects() Get the collection of <i>RegistryObjects</i> classified by this <i>ClassificationNode</i> . Maps to attribute named
------------	--

	<code>classifiedObjects.</code>
ClassificationNode	getParent() Gets the parent ClassificationNode for this ClassificationNode. Maps to attribute named <code>parent</code> .
String	getPath() Gets the path from the root ancestor of this ClassificationNode. The path conforms to the [XPATH] expression syntax (e.g. "/Geography/Asia/Japan"). Maps to attribute named <code>path</code> .
void	setParent(ClassificationNode parent) Sets the parent ClassificationNode for this ClassificationNode.
String	getCode() Gets the code for this ClassificationNode. See section 11.4 for details. Maps to attribute named <code>code</code> .
void	setCode(String code) Sets the code for this ClassificationNode. See section 11.4 for details.

710
711
712
713
714
715
716
717
718
719
720

Note that methods inherited from the base interfaces of this interface are not shown.

In Figure 4, several *Instances* of ClassificationNode are defined (all light colored boxes). A ClassificationNode has zero or one ClassificationNodes for its parent and zero or more ClassificationNodes for its immediate children. If a ClassificationNode has no parent then it is the root of a *Classification* tree. Note that the entire *Classification* tree is recursively defined by a single information model *Element* ClassificationNode.

721 **11.2 Interface Classification**

722 **All Superinterfaces:**

723 [IntrinsicObject](#), [RegistryEntry](#), [RegistryObject](#), [Versionable](#)

724
725
726
727
728

Classification *Instances* are used to classify repository item by associating their RegistryEntry *Instance* with a ClassificationNode *Instance* within a *Classification* scheme.

729
730
731
732

In Figure 4, Classification *Instances* are not explicitly shown but are implied as associations between the RegistryEntries (shaded leaf node) and the associated ClassificationNode

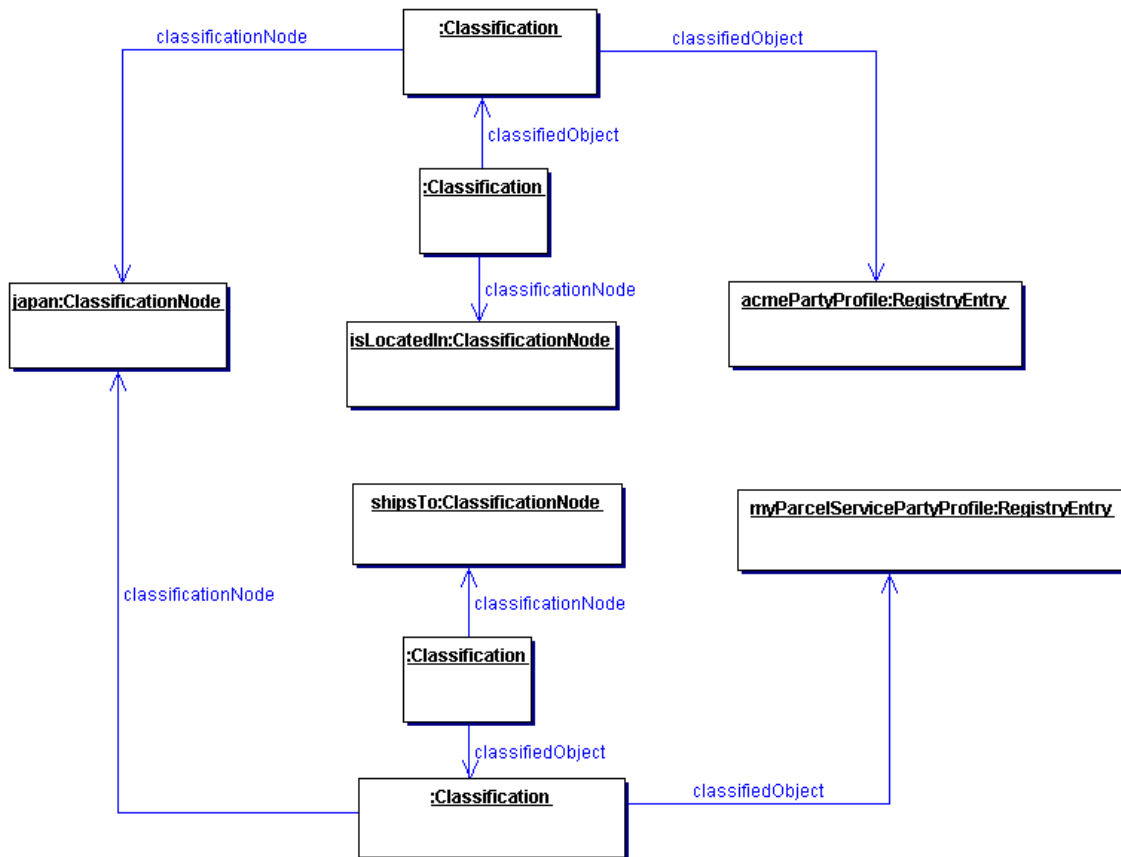
Method Summary of Classification	
RegistryObject	getClassifiedObject()

	Gets the RegistryObject that is classified by this Classification. Maps to attribute named <code>classifiedObject</code> .
RegistryObject	getClassificationNode () Gets the ClassificationNode that classifies the RegistryObject in this Classification. Maps to attribute named <code>classificationNode</code> .

733 Note that methods inherited from the base interfaces of this interface are not
734 shown.

735 **11.2.1 Context Sensitive Classification**

736 Consider the case depicted in Figure 7 where a *Collaboration Protocol Profile* for
737 ACME Inc. is classified by the Japan ClassificationNode under the Geography
738 *Classification* scheme. In the absence of the context for this *Classification* its
739 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it
740 mean that ACME ships products to Japan, or does it have some other meaning?
741 To address this ambiguity a Classification may optionally be associated with
742 another ClassificationNode (in this example named `isLocatedIn`) that provides the
743 missing context for the Classification. Another *Collaboration Protocol Profile* for
744 MyParcelService may be classified by the Japan ClassificationNode where this
745 Classification is associated with a different ClassificationNode (e.g. named
746 `shipsTo`) to indicate a different context than the one used by ACME Inc.



747

748

Figure 7: Context Sensitive Classification

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

752

753 In summary, the generalized support for *Classification* schemes in the
 754 information model allows:

- 755 o A RegistryEntry to be classified by defining a Classification that associates it
 756 with a ClassificationNode in a *Classification tree*.
- 757 o A RegistryEntry to be classified along multiple facets by having multiple
 758 *Classifications* that associate it with multiple ClassificationNodes.
- 759 o A *Classification* defined for a RegistryEntry to be qualified by the contexts in
 760 which it is being classified.

761 **11.3 Example of Classification Schemes**

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

766

767

Classification Scheme (Context)	Usage Example
Industry	Find all Parties in Automotive industry
Process	Find a ServiceInterface that implements a Process
Product	Find a <i>Business</i> that sells a product
Locale	Find a Supplier located in Japan
Temporal	Find Supplier that can ship with 24 hours
Role	Find All Suppliers that have a <i>Role</i> of "Seller"

768

Table 1: Sample Classification Schemes

769 **11.4 Standardized Taxonomy Support**

770 Standardized taxonomies also referred to as ontologies or coding schemes exist
 771 in various industries to provide a structured coded vocabulary. The ebXML
 772 *Registry* does not define support for specific taxonomies. Instead it provides a
 773 general capability to link RegistryEntries to codes defined by various taxonomies.

774

775 The information model provides two alternatives for using standardized
 776 taxonomies for *Classification* of RegistryEntries.

777 **11.4.1 Full-featured Taxonomy Based Classification**

778 The information model provides a full-featured taxonomy based *Classification*
779 alternative based *Classification* and *ClassificationNode Instances*. This
780 alternative requires that a standard taxonomy be imported into the *Registry* as a
781 *Classification* tree consisting of *ClassificationNode Instances*. This specification
782 does not prescribe the transformation tools necessary to convert standard
783 taxonomies into ebXML *Registry Classification* trees. However, the
784 transformation MUST ensure that:

- 785 1. The name attribute of the root *ClassificationNode* is the *name* of the
786 standard taxonomy (e.g. NAICS, ICD-9, SNOMED).
- 787 2. All codes in the standard taxonomy are preserved in the *code* attribute of
788 a *ClassificationNode*.
- 789 3. The intended structure of the standard taxonomy is preserved in the
790 *ClassificationNode* tree, thus allowing polymorphic browse and drill down
791 discovery. This means that is searching for entries classified by Asia will
792 find entries classified by descendants of Asia (e.g. Japan and Korea).

793 **11.4.2 Light Weight Taxonomy Based Classification**

794 The information model also provides a lightweight alternative for classifying
795 *RegistryEntry Instances* by codes defined by standard taxonomies, where the
796 submitter does not wish to import an entire taxonomy as a native *Classification*
797 scheme.

798
799 In this alternative the submitter adds one or more taxonomy related Slots to the
800 *RegistryEntry* for a submitted repository item. Each Slot's name identifies a
801 standardized taxonomy while the Slot's value is the code within the specified
802 taxonomy. Such taxonomy related Slots MUST be defined with a slotType of
803 *Classification*.

804
805 For example if a *RegistryEntry* has a Slot with name "NAICS", a slotType of
806 "Classification" and a value "51113" it implies that the *RegistryEntry* is classified
807 by the code for "Book Publishers" in the NAICS taxonomy. Note that in this
808 example, there is no need to import the entire NAICS taxonomy, nor is there any
809 need to create *Instances* of *ClassificationNode* or *Classification*.

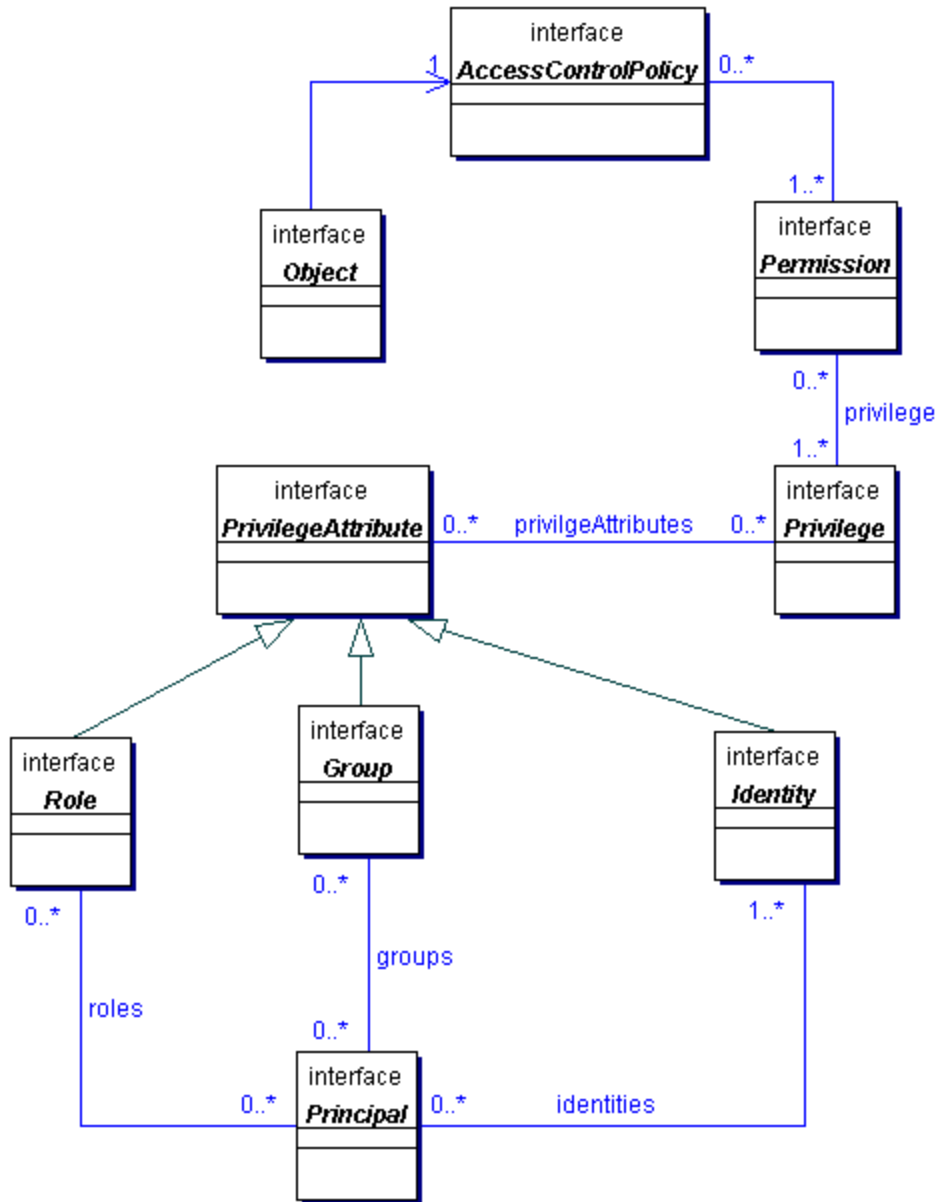
810
811 The following points are noteworthy in this light weight *Classification* alternative:
812

- 813 • Validation of the name and the value of the *Classification* is responsibility
814 of the SO and not of the ebXML *Registry* itself.
- 815 • Discovery is based on exact match on slot name and slot value rather
816 than the flexible "browse and drill down discovery" available to the heavy
817 weight *Classification* alternative.

818 **12 Information Model: Security View**

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

821
 822 Figure 8 shows the view of the objects in the *Registry* from a security
 823 perspective. It shows object relationships as a *UML Class* diagram. It does not
 824 show *Class* attributes or *Class* methods that will be described in subsequent
 825 sections. It is meant to be illustrative not prescriptive.
 826



827
 828

Figure 8: Information Model: Security View

829 **12.1 Interface AccessControlPolicy**

830 Every RegistryObject is associated with exactly one AccessControlPolicy which
 831 defines the policy rules that govern access to operations or methods performed
 832 on that RegistryObject. Such policy rules are defined as a collection of
 833 Permissions.

834
 835
 836
 837

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

838

839 **12.2 Interface Permission**

840
 841 The Permission object is used for authorization and access control to
 842 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are
 843 defined in an AccessControlPolicy object.

844
 845 A Permission object authorizes access to a method in a RegistryObject if the
 846 requesting Principal has any of the Privileges defined in the Permission.

847 **See Also:**

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

849

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> .

850

851 **12.3 Interface Privilege**

852
 853 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute
 854 can be a Group, a Role, or an Identity.

855

856 A requesting Principal MUST have all of the PrivilegeAttributes specified in a
 857 Privilege in order to gain access to a method in a protected RegistryObject.
 858 Permissions defined in the RegistryObject's AccessControlPolicy define the
 859 Privileges that can authorize access to specific methods.
 860
 861 This mechanism enables the flexibility to have object access control policies that
 862 are based on any combination of Roles, Identities or Groups.

863 **See Also:**
 864 [PrivilegeAttribute](#), [Permission](#)

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

868

869 **12.4 Interface PrivilegeAttribute**

870 **All Known Subinterfaces:**
 871 [Group](#), [Identity](#), [Role](#)

872
 873 PrivilegeAttribute is a common base *Class* for all types of security attributes that
 874 are used to grant specific access control privileges to a Principal. A Principal may
 875 have several different types of PrivilegeAttributes. Specific combination of
 876 PrivilegeAttributes may be defined as a Privilege object.

877 **See Also:**
 878 [Principal](#), [Privilege](#)

879 **12.5 Interface Role**

880 **All Superinterfaces:**
 881 [PrivilegeAttribute](#)

882
 883 A security Role PrivilegeAttribute. For example a hospital may have *Roles* such
 884 as Nurse, Doctor, Administrator etc. Roles are used to grant Privileges to
 885 Principals. For example a Doctor *Role* may be allowed to write a prescription but
 886 a Nurse *Role* may not.

887 **12.6 Interface Group**

888 **All Superinterfaces:**
 889 [PrivilegeAttribute](#)

890

891 A security Group PrivilegeAttribute. A Group is an aggregation of users that may
 892 have different Roles. For example a hospital may have a Group defined for
 893 Nurses and Doctors that are participating in a specific clinical trial (e.g.
 894 AspirinTrial group). Groups are used to grant Privileges to Principals. For
 895 example the members of the AspirinTrial group may be allowed to write a
 896 prescription for Aspirin (even though Nurse Role as a rule may not be allowed to
 897 write prescriptions).

898 **12.7 Interface Identity**

899 **All Superinterfaces:**

900 [PrivilegeAttribute](#)

901

902 A security Identity PrivilegeAttribute. This is typically used to identify a person, an
 903 organization, or software service. Identity attribute may be in the form of a digital
 904 certificate.

905 **12.8 Interface Principal**

906

907 Principal is a completely generic term used by the security community to include
 908 both people and software systems. The Principal object is an entity that has a set
 909 of PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and
 910 optionally a set of role memberships, group memberships or security clearances.
 911 A principal is used to authenticate a requestor and to authorize the requested
 912 action based on the PrivilegeAttributes associated with the Principal.

913 **See Also:**

914 PrivilegeAttributes, [Privilege](#), [Permission](#)

915

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> .

916

917

917 **13 References**

- 918 [ebGLOSS] ebXML Glossary,
919 http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 920 [ebTA] ebXML Technical Architecture Specification
921 http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.4.pdf
- 922 [OAS] OASIS Information Model
923 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>
- 924 [ISO] ISO 11179 Information Model
925 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba78525662100>
926 [5419d7/b83fc7816a6064c68525690e0065f913?OpenDocument](http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument)
- 927 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use
928 in RFCs to Indicate Requirement Levels
929 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>
- 930 [ebRS] ebXML Registry Services Specification
931 http://www.ebxml.org/specdrafts/ebXML_RS_v1.0.pdf
- 932 [ebBPSS] ebXML Business Process Specification Schema
933 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 934 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
935 <http://www.ebxml.org/specrafts/>
936
- 937 [UUID] DCE 128 bit Universal Unique Identifier
938 http://www.opengroup.org/onlinepubs/009629399/apdx.htm#tagcjh_20
939 <http://www.opengroup.org/publications/catalog/c706.htm><http://www.w3.org/TR/REC-xml>
940
- 941 [XPATH] XML Path Language (XPath) Version 1.0
942 <http://www.w3.org/TR/xpath>
943

944 **14 Disclaimer**

945 The views and specification expressed in this document are those of the authors
946 and are not necessarily those of their employers. The authors and their
947 employers specifically disclaim responsibility for any problems arising from
948 correct or incorrect implementation or use of this design.
949

949 **15 Contact Information**

950

951 Team Leader

952 Name: Scott Nieman
953 Company: Norstan Consulting
954 Street: 5101 Shady Oak Road
955 City, State, Postal Code: Minnetonka, MN 55343
956 Country: USA
957 Phone: 952.352.5889
958 Email: Scott.Nieman@Norstan

959

960 Vice Team Lead

961 Name: Yutaka Yoshida
962 Company: Sun Microsystems
963 Street: 901 San Antonio Road, MS UMPK17-102
964 City, State, Postal Code: Palo Alto, CA 94303
965 Country: USA
966 Phone: 650.786.5488
967 Email: Yutaka.Yoshida@eng.sun.com

968

969 Editor

970 Name: Farrukh S. Najmi
971 Company: Sun Microsystems
972 Street: 1 Network Dr., MS BUR02-302
973 City, State, Postal Code: Burlington, MA, 01803-0902
974 Country: USA
975 Phone: 781.442.0703
976 Email: najmi@east.sun.com

977

978

978 **Copyright Statement**

979 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved

980

981 This document and translations of it MAY be copied and furnished to others, and
982 derivative works that comment on or otherwise explain it or assist in its
983 implementation MAY be prepared, copied, published and distributed, in whole or
984 in part, without restriction of any kind, provided that the above copyright notice
985 and this paragraph are included on all such copies and derivative works.

986 However, this document itself MAY not be modified in any way, such as by
987 removing the copyright notice or references to ebXML, UN/CEFACT, or OASIS,
988 except as required to translate it into languages other than English.

989

990 The limited permissions granted above are perpetual and will not be revoked by
991 ebXML or its successors or assigns.

992

993 This document and the information contained herein is provided on an "AS IS"
994 basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
995 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
996 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
997 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
998 PURPOSE.

999