



Creating A Single Global Electronic Market

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30

## **ebXML Registry Information Model**

### **ebXML Registry Project Team**

#### **Working Draft 3/19/2001**

**This version: Version 0.60**

## **1 Status of this Document**

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

Distribution of this document is unlimited.

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

***This version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryInfoModelv0.60.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.60.pdf)

***Latest version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryInfoModel.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModel.pdf)

***Previous version:***

[http://www.ebxml.org/project\\_teams/registry/private/RegistryInfoModelv0.59.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.59.pdf)

## 30 **2 ebXML participants**

31 The authors wish to acknowledge the support of the members of the Registry  
32 Project Team who contributed ideas to this specification by the group's  
33 discussion e-mail list, on conference calls and during face-to-face meetings.

34

35 Lisa Carnahan – NIST

36 Joe Dalman - Tie

37 Philippe DeSmedt - Viquity

38 Sally Fuger – AIAG

39 Len Gallagher - NIST

40 Steve Hanna - Sun Microsystems

41 Scott Hinkelman - IBM

42 Michael Kass, NIST

43 Jong.L Kim – Innodigital

44 Sangwon Lim, Korea Institute for Electronic Commerce

45 Bob Miller - GXS

46 Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan

47 Dale Moberg – Sterling Commerce

48 Ron Monzillo – Sun Microsystems

49 JP Morgenthal – eThink Systems, Inc.

50 Joel Munter - Intel

51 Farrukh Najmi - Sun Microsystems

52 Scott Nieman - Norstan Consulting

53 Frank Olken – Lawrence Berkeley National Laboratory

54 Michael Park - eSum Technologies

55 Bruce Peat - eProcess Solutions

56 Mike Rowley – Excelon Corporation

57 Waqar Sadiq - Vitria

58 Krishna Sankar - Cisco Systems Inc.

59 Kim Tae Soo - Government of Korea

60 Nikola Stojanovic - Encoda Systems, Inc.

61 David Webber – XML Global

62 Yutaka Yoshida - Sun Microsystems

63 Prasad Yendluri - webmethods

64 Peter Z. Zhoo - Knowledge For the new Millennium

65

66

66 **Table of Contents**

67

68 **1 STATUS OF THIS DOCUMENT.....1**

69 **2 EBXML PARTICIPANTS.....2**

70 **3 INTRODUCTION .....6**

71 3.1 SUMMARY OF CONTENTS OF DOCUMENT.....6

72 3.2 GENERAL CONVENTIONS .....6

73 3.3 AUDIENCE.....6

74 3.4 RELATED DOCUMENTS .....7

75 **4 DESIGN OBJECTIVES.....7**

76 4.1 GOALS .....7

77 4.2 CAVEATS AND ASSUMPTIONS .....7

78 **5 SYSTEM OVERVIEW .....7**

79 5.1 ROLE OF EBXML REGISTRY.....7

80 5.2 REGISTRY SERVICES .....8

81 5.3 WHAT THE REGISTRY INFORMATION MODEL DOES .....8

82 5.4 HOW THE REGISTRY INFORMATION MODEL WORKS .....8

83 5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED .....8

84 5.6 CONFORMANCE AS AN EBXML REGISTRY.....8

85 **6 REGISTRY INFORMATION MODEL: PUBLIC VIEW.....9**

86 6.1 REGISTRYENTRY .....10

87 6.2 SLOT.....10

88 6.3 ASSOCIATION.....10

89 6.4 EXTERNALIDENTIFIER .....10

90 6.5 EXTERNALLINK .....10

91 6.6 CLASSIFICATIONNODE.....10

92 6.7 CLASSIFICATION .....11

93 6.8 PACKAGE.....11

94 6.9 AUDITABLEEVENT.....11

95 6.10 USER.....11

96 6.11 POSTALADDRESS.....11

97 6.12 ORGANIZATION.....11

98 **7 REGISTRY INFORMATION MODEL: DETAIL VIEW.....11**

99 7.1 INTERFACE *OBJECT*.....12

100 7.2 INTERFACE *VERSIONABLE*.....14

101 7.3 INTERFACE *REGISTRYENTRY*.....14

102 7.3.1 *Pre-defined RegistryEntry Status Types*.....16

103        7.3.2    *Pre-Defined Object Types*.....17

104        7.3.3    *Pre-defined RegistryEntry Stability Enumerations* .....18

105        7.4      *INTERFACE SLOT* .....18

106        7.5      *INTERFACE EXTRINSIC OBJECT*.....19

107        7.6      *INTERFACE INTRINSIC OBJECT*.....20

108        7.7      *INTERFACE PACKAGE*.....20

109        7.8      *INTERFACE EXTERNALIDENTIFIER*.....21

110        7.9      *INTERFACE EXTERNALLINK*.....21

111    **8    REGISTRY AUDIT TRAIL**.....**22**

112        8.1      *INTERFACE AUDITABLEEVENT* .....22

113        8.1.1    *Pred-defined Auditable Event Types* .....23

114        8.2      *INTERFACE USER*.....23

115        8.3      *INTERFACE ORGANIZATION*.....24

116        8.4      *CLASS POSTALADDRESS* .....25

117        8.5      *CLASS TELEPHONENUMBER* .....26

118        8.6      *CLASS PERSONNAME*.....26

119    **9    REGISTRY ENTRY NAMING** .....**26**

120    **10    ASSOCIATION OF REGISTRY ENTRIES** .....**27**

121        10.1     *INTERFACE ASSOCIATION*.....27

122        10.1.1   *Pre-defined Association Types* .....28

123    **11    CLASSIFICATION OF REGISTRY ENTRIES** .....**29**

124        11.1     *INTERFACE CLASSIFICATIONNODE*.....31

125        11.2     *INTERFACE CLASSIFICATION* .....32

126        11.2.1   *Context Sensitive Classification* .....33

127        11.3     *EXAMPLE OF CLASSIFICATION SCHEMES* .....34

128        11.4     *STANDARDIZED TAXONOMY SUPPORT* .....35

129        11.4.1   *Full-featured Taxonomy Based Classification* .....35

130        11.4.2   *Light Weight Taxonomy Based Classification*.....35

131    **12    INFORMATION MODEL: SECURITY VIEW** .....**36**

132        12.1     *INTERFACE ACCESSCONTROLPOLICY*.....37

133        12.2     *INTERFACE PERMISSION* .....38

134        12.3     *INTERFACE PRIVILEGE*.....38

135        12.4     *INTERFACE PRIVILEGEATTRIBUTE*.....39

136        12.5     *INTERFACE ROLE* .....39

137        12.6     *INTERFACE GROUP*.....39

138        12.7     *INTERFACE IDENTITY* .....39

139        12.8     *INTERFACE PRINCIPAL*.....40

140    **13    REFERENCES** .....**41**

141    **14    DISCLAIMER** .....**41**

142 **15 CONTACT INFORMATION.....42**

143 **COPYRIGHT STATEMENT.....43**

144 **Table of Figures**

145 Figure 1: Information Model Public View..... 9

146 Figure 2: Information Model Inheritance View..... 12

147 Figure 3: Example of Registry Entry Association..... 27

148 Figure 4: Example showing a Classification Tree..... 30

149 Figure 5: Information Model Classification View..... 31

150 Figure 6: Classification Instance Diagram ..... 31

151 Figure 7: Context Sensitive Classification..... 34

152 Figure 8: Information Model: Security View..... 37

153 **Table of Tables**

154 Table 1: Sample Classification Schemes..... 35

155

156

## 156 **3 Introduction**

### 157 **3.1 Summary of Contents of Document**

158 This document specifies the information model for the ebXML Registry.

159

160 A separate document, *ebXML Registry Services Specification* [RS], describes  
161 how to build Registry Services that provide access to the information content in  
162 the ebXML Registry.

### 163 **3.2 General Conventions**

164 o UML diagrams are used as a way to concisely describe concepts. They are  
165 not intended to convey any specific implementation or methodology  
166 requirements.

167 o Interfaces are often used in UML diagrams. They are used instead of classes  
168 with attributes to provide an abstract definition without implying any specific  
169 implementation. Specifically, they do not imply that objects in the Registry will  
170 be accessed directly via these interfaces. Objects in the Registry are  
171 accessed via interfaces described in the *ebXML Registry Services*  
172 *Specification*. Each get method in every interface has an explicit indication of  
173 the attribute name that the get method maps to. For example getName  
174 method maps to an attribute named `name`.

175 o The term “*repository item*” is used to refer to actual Registry content (e.g. a  
176 DTD, as opposed to metadata about the DTD). It is important to note that the  
177 information model is not modeling actual repository items.

178 o The term “*RegistryEntry*” is used to refer to an object that provides metadata  
179 about content instance (*repository item*).

180

181 The information model *does not* contain *any* elements that are the actual content  
182 of the Registry (*repository item*). All elements of the information model represent  
183 metadata about the content and not the content itself.

184

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

187

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

### 191 **3.3 Audience**

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

- 194 o Implementers of ebXML Registry Services
- 195 o Implementers of ebXML Registry Clients

### 196 **3.4 Related Documents**

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

- 199 a) *ebXML Registry Business Domain Model* [BDM] - defines requirements  
200 for ebXML Registry Services
- 201 b) *ebXML Registry Services Specification* [RS] - defines the actual Registry  
202 services based on this information model
- 203 c) *Collaboration Protocol Agreement Specification* [CPA] (under  
204 development) - defines how profiles can be defined for a party and how  
205 two parties' profiles may be used to define a party agreement
- 206 d) *ebXML Business Process Specification Schema* [BPM]  
207

## 208 **4 Design Objectives**

### 209 **4.1 Goals**

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

- 211 o Communicate what information is in the Registry and how that information is  
212 organized
- 213 o Leverage as much as possible the work done in the OASIS [OAS] and the  
214 ISO 11179 [ISO] Registry models
- 215 o Align with relevant works in progress within other ebXML working groups
- 216 o Be able to evolve to support future ebXML Registry requirements
- 217 o Be compatible with other ebXML specifications

### 218 **4.2 Caveats and Assumptions**

219 The Registry Information Model specification is first in a series of phased  
220 deliverables. Later versions of the document will include additional functionality  
221 planned for current and future development.

## 222 **5 System Overview**

### 223 **5.1 Role of ebXML Registry**

224 The Registry provides a stable store where content submitted by a Submitting  
225 Organization is made persistent. Such content is used to facilitate ebXML-based  
226 business to business (B2B) partnerships and transactions. Submitted content  
227 may be XML schema and documents, process descriptions, core components,

228 context descriptions, UML models, information about parties and even software  
229 components.

## 230 **5.2 Registry Services**

231 A set of Registry Services that provide access to Registry content to clients of the  
232 Registry is defined in the *ebXML Registry Services Specification* [RS]. This  
233 document does not provide details on these services but may occasionally refer  
234 to them.

## 235 **5.3 What the Registry Information Model Does**

236 The Registry Information Model provides a blueprint or high-level schema for the  
237 ebXML Registry. Its primary value is for implementers of ebXML Registries. It  
238 provides these implementers with information on the type of metadata that is  
239 stored in the Registry as well as the relationships among metadata classes.

240 The Registry information model:

- 241 o Defines what types of objects are stored in the Registry
- 242 o Defines how stored objects are organized in the Registry
- 243 o Is based on ebXML metamodels from various working groups

244

## 245 **5.4 How the Registry Information Model Works**

246 Implementers of the ebXML Registry may use the information model to  
247 determine which classes to include in their Registry implementation and what  
248 attributes and methods these classes may have. They may also use it to  
249 determine what sort of database schema their Registry implementation may  
250 need.

251 [Note]The information model is meant to be  
252 illustrative and does not prescribe any  
253 specific implementation choices.  
254

## 255 **5.5 Where the Registry Information Model May Be Implemented**

256 The Registry Information Model may be implemented within an ebXML Registry  
257 in form of a relational database schema, object database schema or some other  
258 physical schema. It may also be implemented as interfaces and classes within a  
259 Registry implementation.

## 260 **5.6 Conformance as an ebXML Registry**

261

262 If an implementation claims conformance to this specification then it supports all  
263 required information model classes and interfaces, their attributes and their  
264 semantic definitions that are visible through the ebXML Registry Services.

ebXML Registry Information Model

265 **6 Registry Information Model: Public View**

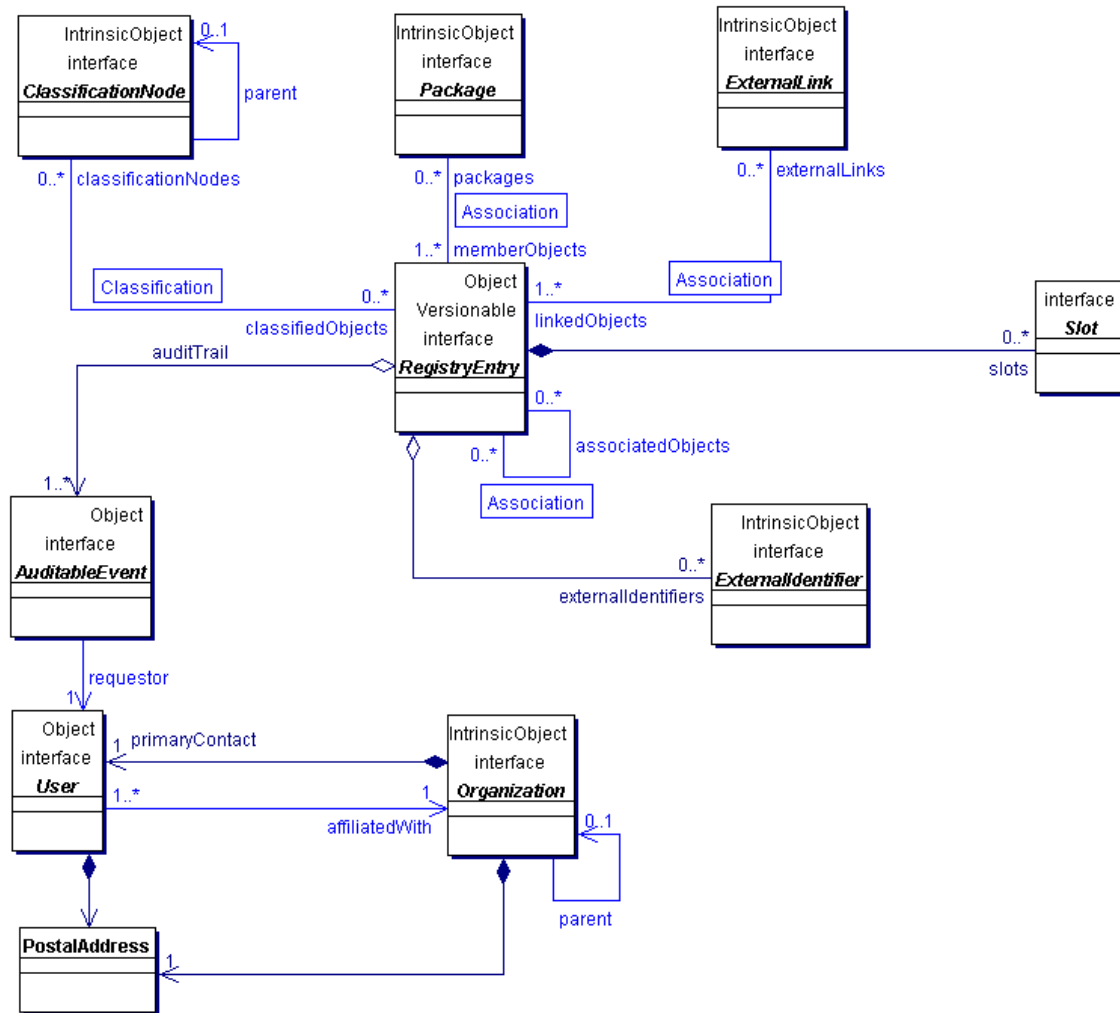
266 This chapter provides a high level public view of the most visible objects in the  
 267 Registry.

268

269 Figure 1 shows the public view of the objects in the Registry and their  
 270 relationships as a UML class diagram. It does not show inheritance, class  
 271 attributes or class methods.

272 The reader is again reminded that the information model is not modeling actual  
 273 repository items.

274



275

276

**Figure 1: Information Model Public View**

## 277 **6.1 RegistryEntry**

278 The central object in the information model is a RegistryEntry. An instance of  
279 RegistryEntry exists for each content instance submitted to the Registry.  
280 Instances of the RegistryEntry class provide metadata about a repository item in  
281 the Registry. The actual repository item (e.g. a DTD) is not contained in an  
282 instance of the RegistryEntry class. Note that most classes in the information  
283 model are specialized sub-classes of RegistryEntry. Each RegistryEntry is  
284 related to exactly one repository item, however, in the future revision of this  
285 document, it may be related to multiple repository items.

## 286 **6.2 Slot**

287 Slot instances provide a dynamic way to add arbitrary attributes to RegistryEntry  
288 instances. This ability to add attributes dynamically to RegistryEntry instances  
289 enables extensibility within the Registry Information Model.

## 290 **6.3 Association**

291 Association instances are RegistryEntries that are used to define many-to-many  
292 associations between objects in the information model. Associations are  
293 described in detail in chapter 10.

## 294 **6.4 ExternalIdentifier**

295 ExternalIdentifier instances provide additional identifier information to  
296 RegistryEntry such as DUNS number, Social Security Number, or an alias name  
297 of the organization.

## 298 **6.5 ExternalLink**

299 ExternalLink instances are RegistryEntries that model a named URI to content  
300 that is not managed by the Registry. Unlike managed content, such external  
301 content may change or be deleted at any time without the knowledge of the  
302 registry. RegistryEntry may be associated with any number of ExternalLinks.  
303 Consider the case where a Submitting Organization submits a repository item  
304 (e.g. a DTD) and wants to associate some external content to that object (e.g.  
305 the Submitting Organization's home page). The ExternalLink enables this  
306 capability. A potential use of the ExternalLink capability may be in a GUI tool that  
307 displays the ExternalLinks to a RegistryEntry. The user may click on such links  
308 and navigate to an external web page referenced by the link.

## 309 **6.6 ClassificationNode**

310 ClassificationNode instances are RegistryEntries that are used to define tree  
311 structures where each node in the tree is a ClassificationNode. Classification  
312 trees constructed with ClassificationNodes are used to define classification  
313 schemes or ontologies. ClassificationNode is described in detail in chapter 11.

## 314 **6.7 Classification**

315 Classification instances are RegistryEntries that are used to classify repository  
316 item by associating their RegistryEntry instance with a ClassificationNode within  
317 a classification scheme. Classification is described in detail in chapter 11.

## 318 **6.8 Package**

319 Package instances are RegistryEntries that group logically related  
320 RegistryEntries together. One use of a Package is to allow operations to be  
321 performed on an entire package of objects. For example all objects belonging to  
322 a Package may be deleted in a single request.

## 323 **6.9 AuditableEvent**

324 AuditableEvent instances are Objects that are used to provide an audit trail for  
325 RegistryEntries. AuditableEvent is described in detail in chapter 8.

## 326 **6.10 User**

327 User instances are Objects that are used to provide information about registered  
328 users within the registry. User objects are used in audit trail for RegistryEntries.  
329 User is described in detail in chapter 8.  
330

## 331 **6.11 PostalAddress**

332 PostalAddress is a simple reusable entity class that defines attributes of a postal  
333 address.  
334

## 335 **6.12 Organization**

336 Organization instances are RegistryEntries that provide information on  
337 organizations such as a Submitting Organization. Each Organization instance  
338 may have a reference to a parent Organization.

## 339 **7 Registry Information Model: Detail View**

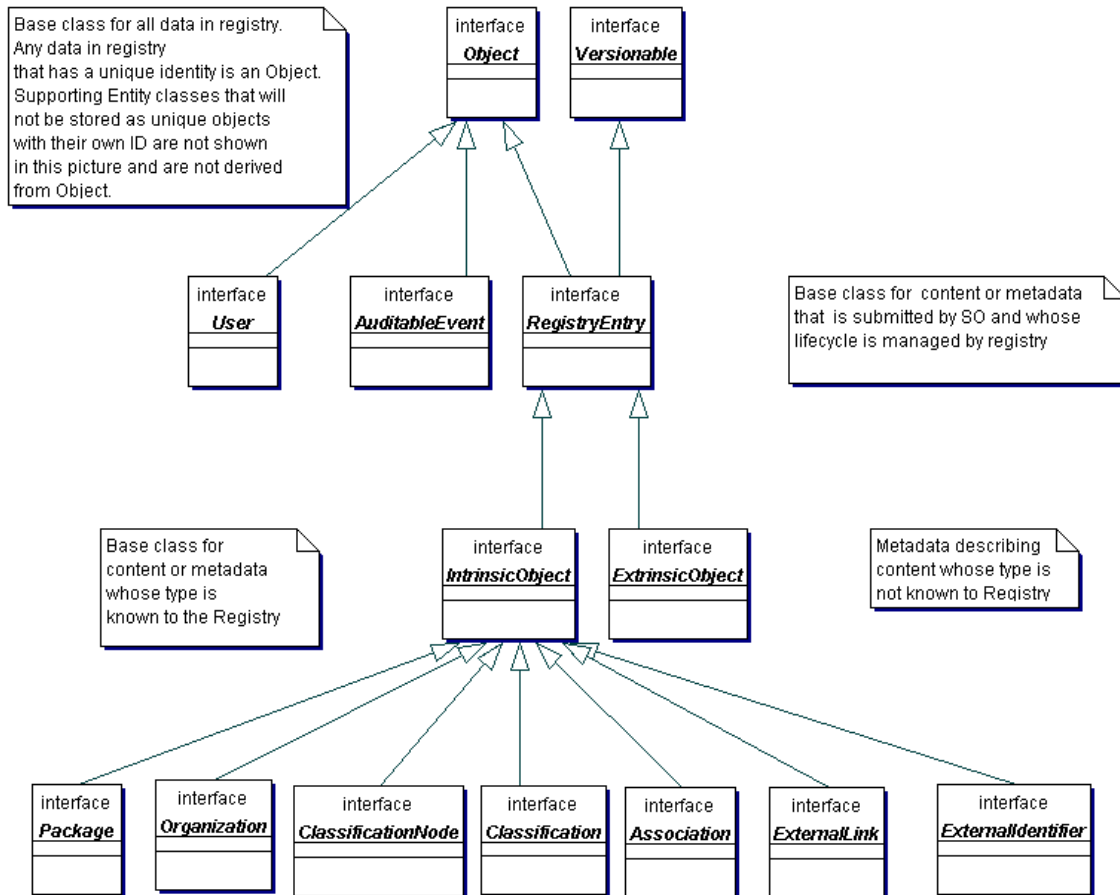
340 This chapter covers the information model classes in more detail than the Public  
341 View. The detail view introduces some additional classes within the model that  
342 were not described in the public view of the information model.  
343

344 Figure 2 shows the inheritance or “is a” relationships between the classes in the  
345 information model. Note that it does not show the other types of relationships,  
346 such as “has a” relationships, since they have already been shown in a previous  
347 figure. Class attributes and class methods are also not shown. Detailed  
348 description of methods and attributes of most interfaces and classes will be  
349 displayed in tabular form following the description of each class in the model.  
350

351 The interface Association will be covered in detail separately in chapter 10. The  
 352 interfaces Classification and ClassificationNode will be covered in detail  
 353 separately in chapter 11.

354

355 The reader is again reminded that the information model is not modeling actual  
 356 repository items.



357

358

Figure 2: Information Model Inheritance View

359

## 360 7.1 Interface Object

### 361 All Known Subinterfaces:

- 362 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),  
 363 [ExtrinsicObject](#), [IntrinsicObject](#), [RegistryEntry](#), [Organization](#), [Package](#),  
 364 [Submission](#)

365

366 Object provides a common base interface for almost all objects in the information  
 367 model. Information model classes whose instances have a unique identity and an  
 368 independent life cycle are descendants of the Object class.

369

370 Note that Slot and PostalAddress are not descendants of the Object class  
 371 because their instances do not have an independent existence and unique  
 372 identity. They are always a part of some other class's instance (e.g. Organization  
 373 has a PostalAddress).

374  
 375

<b>Method Summary</b>	
<a href="#">AccessControlPolicy</a>	<p><a href="#">getAccessControlPolicy()</a></p> <p>Gets the AccessControlPolicy object associated with this Object. An AccessControlPolicy defines the security model associated with the Object in terms of “who is permitted to do what” with that Object. Maps to attribute named <code>accessControlPolicy</code>.</p>
String	<p><a href="#">getDescription()</a></p> <p>Gets the context independent textual description for this object. Maps to attribute named <code>description</code>.</p>
String	<p><a href="#">getName()</a></p> <p>Gets user friendly context independent name of object in repository. Maps to attribute named <code>name</code>.</p>
String	<p><a href="#">getID()</a></p> <p>Gets the universally unique ID (UUID) for this object. Maps to attribute named <code>id</code>.</p>
void	<p><a href="#">setDescription(String description)</a></p> <p>Sets the context independent textual description for this object.</p>
void	<p><a href="#">setName(String name)</a></p> <p>Sets user friendly context independent name of object in repository.</p>
void	<p><a href="#">setID(String id)</a></p> <p>Sets the universally unique ID (UUID) for this object.</p>

376  
 377

377 **7.2 Interface *Versionable***

378 **All Known Subinterfaces:**

379 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),  
 380 [ExtrinsicObject](#), [IntrinsicObject](#), [RegistryEntry](#), [Organization](#), [Package](#)

381 

---

382 The Versionable interface defines the behavior common to classes that are  
 383 capable of creating versions of their instances. At present all RegistryEntry  
 384 classes are required to implement the Versionable interface.

385

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

386

387 **7.3 Interface *RegistryEntry***

388 **All Superinterfaces:**

389 [Object](#), [Versionable](#)

390 **All Known Subinterfaces:**

391 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),  
 392 [ExtrinsicObject](#), [IntrinsicObject](#), [Organization](#), [Package](#)

393

394 

---

395 RegistryEntry is a common base class for all metadata describing submitted  
 396 content whose life cycle is managed by the registry. Metadata describing content  
 397 submitted to the registry is further specialized by the ExtrinsicObject and  
 398 IntrinsicObject subclasses of RegistryEntry.

398

399

400

401 

---

<b>Method Summary</b>	
Collection	<p><a href="#">getAssociatedObjects()</a> Returns the collection of Objects associated with this object. Maps to attribute named <code>associatedObjects</code>.</p>
Collection	<p><a href="#">getAuditTrail()</a> Returns the complete audit trail of all requests that effected a state change in this object as an ordered Collection of <code>AuditableEvent</code> objects. Maps to attribute named <code>auditTrail</code>.</p>
Collection	<p><a href="#">getClassificationNodes()</a> Returns the collection of <code>ClassificationNodes</code> associated with this object. Maps to attribute named <code>classificationNodes</code>.</p>
Collection	<p><a href="#">getExternalLinks()</a> Returns the collection of <code>ExternalLinks</code> associated with this object. Maps to attribute named <code>externalLinks</code>.</p>
String	<p><a href="#">getObjectType()</a> Gets the pre-defined object type associated with this <code>RegistryEntry</code>. 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><a href="#">getPackages()</a> Returns the collection of <code>Packages</code> associated with this object. Maps to attribute named <code>packages</code>.</p>
String	<p><a href="#">getStatus()</a> Gets the life cycle status of the <code>RegistryEntry</code> within the Registry. This should be the name of a <code>RegistryEntry</code> status type as described in 7.3.1. Maps to attribute named <code>status</code>.</p>
String	<p><a href="#">getUserVersion()</a> Gets the <code>userVersion</code> attribute of the <code>RegistryEntry</code> within the Registry. The <code>userVersion</code> is the version for the <code>RegistryEntry</code> as assigned by the user.</p>
void	<p><a href="#">setUserVersion(String UserVersion)</a> Sets the <code>userVersion</code> attribute of the <code>RegistryEntry</code> within the Registry.</p>
String	<p><a href="#">getStability()</a> Gets the stability indicator for the <code>RegistryEntry</code> within the Registry. The stability indicator is provided by the submitter as a guarentee 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>.</p>
Date	<p><a href="#">getExpirationDate()</a> Gets <code>expirationDate</code> attribute of the <code>RegistryEntry</code> within the Reastrv. This attribute defines a time limit upon the stability</p>

	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 expirationDate.
void	<a href="#">setExpirationDate</a> (Date expirationDate) Sets expirationDate attribute of the RegistryEntry within the Registry.
Collection	<a href="#">getSlots</a> () Gets the collection of slots that have been dynamically added to this object. Maps to attribute named slots.
void	<a href="#">addSlots</a> (Collection newSlots) Adds one or more slots to this object. Slot names must be locally unique within this object. Any existing slots are not effected.
void	<a href="#">removeSlots</a> (Collection slotNames) Removes one or more slots from this object. Slots to be removed are identified by their name.

402

<b>Methods inherited from interface</b>
<a href="#">getAccessControlPolicy</a> , <a href="#">getDescription</a> , <a href="#">getName</a> , <a href="#">getID</a> , <a href="#">setDescription</a> , <a href="#">setName</a> , <a href="#">setID</a>

403

<b>Methods inherited from interface</b>
<a href="#">getMajorVersion</a> , <a href="#">getMinorVersion</a> , <a href="#">setMajorVersion</a> , <a href="#">setMinorVersion</a>

404 **7.3.1 Pre-defined RegistryEntry Status Types**

405 The following table lists pre-defined choices for RegistryEntry status attribute.  
 406 These pre-defined status types are defined as a Classification scheme. While the  
 407 scheme may easily be extended, a registry must support the status types listed  
 408 below.

409

<b>Name</b>	<b>Description</b>
Submitted	Status of a RegistryEntry that catalogues content that has been submitted to the Registry.
Approved	Status of a RegistryEntry that catalogues content that has been submitted to the Registry and has been subsequently approved.
Deprecated	Status of a RegistryEntry that catalogues content that has

	been submitted to the Registry and has been subsequently deprecated.
<b>Withdrawn</b>	Status of a RegistryEntry that catalogues content that has been withdrawn from the Registry.

410 **7.3.2 Pre-Defined Object Types**

411 The following table lists pre-defined object types. Note that for an ExtrinsicObject  
 412 there are many types defined based on the type of repository item the  
 413 ExtrinsicObject catalogs. In addition there there are object types defined for  
 414 IntrinsicObject sub-classes that may have concrete instances.

415  
 416 These pre-defined object types are defined as a Classification scheme. While the  
 417 scheme may easily be extended a registry must support the object types listed  
 418 below.

419

<b>name</b>	<b>description</b>
<b>Unknown</b>	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
<b>CPA</b>	An ExtrinsicObject of this type catalogues an XML document Collaboration Protocol Agreement (CPA) representing a technical agreement between two parties on how they plan to communicate with each other using a specific protocol.
<b>CPP</b>	An ExtrinsicObject of this type catalogues an XML document called Collaboration Protocol Profile (CPP) that provides information about a party participating in a business transaction.
<b>Process</b>	An ExtrinsicObject of this type catalogues a process description document.
<b>Role</b>	An ExtrinsicObject of this type catalogues an XML description of a Role in a Collaboration Protocol Profile (CPP).
<b>ServiceInterface</b>	An ExtrinsicObject of this type catalogues an XML description of a service interface as defined by [CPA].
<b>SoftwareComponent</b>	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or class library).
<b>Transport</b>	An ExtrinsicObject of this type catalogues an XML description of a transport configuration as defined by [CPA].
<b>UMLModel</b>	An ExtrinsicObject of this type catalogues a UML model.
<b>XMLSchema</b>	An ExtrinsicObject of this type catalogues an XML schema

	(DTD, XML Schema, RELAX grammar, etc.).
<b>Package</b>	A Package object
<b>ExternalLink</b>	An ExternalLink object
<b>ExternalIdentifier</b>	An ExternalIdentifier object
<b>Association</b>	An Association object
<b>Classification</b>	A Classification object
<b>ClassificationNode</b>	A ClassificationNode object
<b>AuditableEvent</b>	An AuditableEvent object
<b>User</b>	A User object
<b>Organization</b>	An Organization object

420

421 **7.3.3 Pre-defined RegistryEntry Stability Enumerations**

422 The following table lists pre-defined choices for RegistryEntry stability attribute.  
 423 These pre-defined stability types are defined as a Classification scheme. While  
 424 the scheme may easily be extended, a registry must support the stability types  
 425 listed below.

426

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.

427

428

429 **7.4 Interface Slot**

430

431 Slot instances provide a dynamic way to add arbitrary attributes to RegistryEntry  
 432 instances. This ability to add attributes dynamically to RegistryEntry instances  
 433 enables extensibility within the Registry Information Model.

434

435 In this model, a RegistryEntry may have 0 or more Slots. A slot is composed of a  
 436 name, a slotType and a collection of values. The name of slot is locally unique  
 437 within the RegistryEntry instance. Similarly, the value of a Slot is locally unique  
 438 within a slot instance. Since a Slot represent an extensible attribute whose value

439 may be a collection, therefore a Slot is allowed to have a collection of values  
 440 rather than a single value. The slotType attribute may optionally specify a type or  
 441 category for the slot.  
 442  
 443

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

444

445 **7.5 Interface *ExtrinsicObject***

446 **All Superinterfaces:**

447 [RegistryEntry](#), [Object](#), [Versionable](#)

448

449 ExtrinsicObjects provide metadata that describes submitted content whose type  
 450 is not intrinsically known to the registry and therefore must be described by  
 451 means of additional attributes (e.g., mime type).

452

453 Examples of content described by ExtrinsicObject include Collaboration Protocol  
 454 Profiles (CPP), business process descriptions, and schemas.

455

<b>Method Summary</b>	
String	<a href="#">getContentURI</a> () Gets the URI to the content catalogued by this ExtrinsicObject.

	A registry must guarantee that this URI is resolvable. Maps to attribute named <code>contentURI</code> .
String	<a href="#">getMimeType()</a> Gets the mime type associated with the content catalogued by this <code>ExtrinsicObject</code> . Maps to attribute named <code>contentType</code> .
boolean	<a href="#">isOpaque()</a> Determines whether the content catalogued by this <code>ExtrinsicObject</code> is opaque to (not readable by) the Registry. In some situations, a Submitting Organization may submit content that is encrypted and not even readable by the Registry. Maps to attribute named <code>opaque</code> .
void	<a href="#">setContentURI(String uri)</a> Sets the URI to the content catalogued by this <code>ExtrinsicObject</code> .
void	<a href="#">setMimeType(String mimeType)</a> Sets the mime type associated with the content catalogued by this <code>ExtrinsicObject</code> .
void	<a href="#">setOpaque(boolean isOpaque)</a> Sets whether the content catalogued by this <code>ExtrinsicObject</code> is opaque to (not readable by) the Registry.

456

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

459 **7.6 Interface *IntrinsicObject***

460 **All Superinterfaces:**

461 [RegistryEntry](#), [Object](#), [Versionable](#)

462 **All Known Subinterfaces:**

463 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#), [Organization](#),  
464 [Package](#)

465

466 `IntrinsicObject` serve as a common base class for derived classes that catalogue  
467 submitted content whose type is known to the Registry and defined by the  
468 ebXML registry specifications.

469

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

472

473 **7.7 Interface *Package***

474 **All Superinterfaces:**

475 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

476

477 Logically related registry entries may be grouped into a Package. It is anticipated  
 478 that Registry Services will allow operations to be performed on an entire package  
 479 of objects in the future.

480  
 481

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

482

483 **7.8 Interface *ExternalIdentifier***

484 **All Superinterfaces:**

485 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

486

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

493 **See Also:**

494

Method Summary	
<a href="#">String</a>	<b>getValue()</b> Gets the value of this ExternalIdentifier. Maps to attribute named <code>value</code> .
Void	<b>setValue(String value)</b> Sets the value of this ExternalIdentifier.

495

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

498 **7.9 Interface *ExternalLink***

499 **All Superinterfaces:**

500 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

501

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

505

506

<b>Method Summary</b>	
Collection	<a href="#">getLinkedObjects</a> () Gets the collection of objects that use this external link. Maps to attribute named <code>linkedObjects</code> .
URI	<a href="#">getExternalURI</a> () Gets URI to the external content. Maps to attribute named <code>externalURI</code> .
void	<a href="#">setExternalURI</a> (URI uri) Sets URI to the external content.

507  
508  
509

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

## 510 **8 Registry Audit Trail**

511 This chapter describes the information model elements that support the audit trail  
512 capability of the Registry. Several classes in this chapter are entity classes that  
513 are used as wrappers to model a set of related attributes. These entity classes  
514 do not have any associated behavior. They are analogous to the “struct”  
515 construct in the C programming language.

516  
517  
518  
519  
520  
521  
522

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`.

### 523 **8.1 Interface *AuditableEvent***

524 **All Superinterfaces:**

525

[Object](#)

526  
527  
528  
529  
530

---

`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 `Object`.

531

`AuditableEvents` are usually a result of a client-initiated request. `AuditableEvent` instances are generated by the Registry service to log such events.

534

535 Often such events effect a change in the life cycle of a `RegistryEntry`. For  
536 example a client request could Create, Update, Deprecate or Delete a  
537 `RegistryEntry`. No `AuditableEvent` is created for requests that do not alter the  
538 state of a `RegistryEntry`. Specifically, read-only requests do not generate an

539 AuditableEvent. No AuditableEvent is generated for a RegistryEntry when it is  
 540 classified, assigned to a Package or associated with another Object.  
 541  
 542

543 **8.1.1 Pred-defined Auditable Event Types**

544 The following table lists pre-defined auditable event types. These pre-defined  
 545 event types are defined as a Classification scheme. While the scheme may  
 546 easily be extended, a registry must support the event types listed below.  
 547

Name	description
Created	An event that created a RegistryEntry.
Deleted	An event that deleted a RegistryEntry.
Deprecated	An event that deprecated a RegistryEntry.
Updated	An event that updated the state of a RegistryEntry.
Versioned	An event that versioned a RegistryEntry.

548

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

549

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

552 **8.2 Interface *User***

553 **All Superinterfaces:**

554 [Object](#)

555

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

558

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

559

560 **8.3 Interface *Organization***

561 **All Superinterfaces:**

562 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

563

564 Organization instances provide information on organizations such as a  
 565 Submitting Organization. Each Organization instance may have a reference to a  
 566 parent Organization. In addition it may have a contact attribute defining the

567 primary contact within the organization. An Organization also has an address  
 568 attribute.  
 569 **See Also:**  
 570

Method Summary	
<a href="#">PostalAddress</a>	<p><a href="#">getAddress()</a>                      Gets the PostalAddress for this Organization. Maps to attribute named <code>address</code>.</p>
<a href="#">User</a>	<p><a href="#">getPrimaryContact()</a>                      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><a href="#">getFax()</a>                      Gets the FAX number for this Organization. Maps to attribute named <code>fax</code>.</p>
<a href="#">Organization</a>	<p><a href="#">getParent()</a>                      Gets the parent Organization for this Organization. Maps to attribute named <code>parent</code>.</p>
TelephoneNumber	<p><a href="#">getTelephone()</a>                      Gets the main telephone number for this Organization. Maps to attribute named <code>telephone</code>.</p>

571  
 572 Note that methods inherited from the base interfaces of this interface are not  
 573 shown.  
 574

575 **8.4 Class *PostalAddress***

576

577

578 PostalAddress is a simple reusable entity class that defines attributes of a postal  
 579 address.

580

Field Summary	
String	<p><a href="#">city</a>                      The city</p>
String	<p><a href="#">country</a>                      The country</p>
String	<p><a href="#">postalCode</a>                      The postal or zip code</p>
String	<p><a href="#">state</a>                      The state</p>
String	<p><a href="#">street</a>                      The street</p>

581

582 **8.5 Class *TelephoneNumber***

583

584

585

586

587

---

A simple reusable entity class that defines attributes of a telephone number.

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

588

589 **8.6 Class *PersonName***

590

591

592

593

---

A simple entity class for a person's name.

Field Summary	
String	<a href="#">firstName</a> The first name for this person.
String	<a href="#">lastName</a> The last name (surname) for this person.
String	<a href="#">middleName</a> The middle name for this person.

594

595 **9 Registry Entry Naming**

596

597

598

599

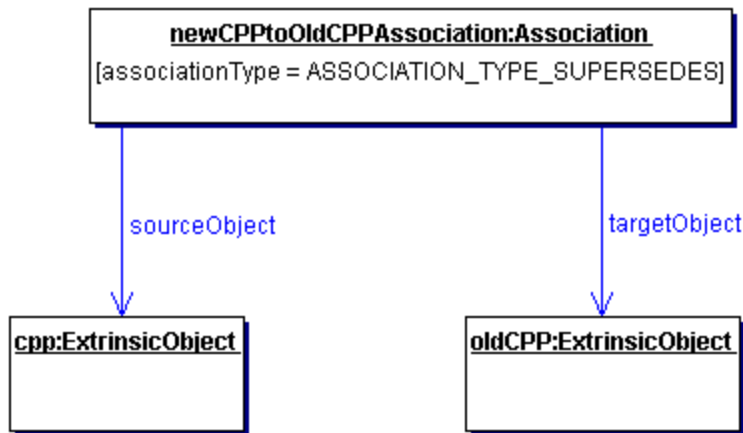
A RegistryEntry has a name that may or may not be unique within the Registry.

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.

600 Alternate contextual naming will be addressed in a later version of the Registry  
 601 Information Model.  
 602

603 **10 Association of Registry Entries**

604 A RegistryEntry may be associated with 0 or more objects. The information  
 605 model defines an Association class. An instance of the Association class  
 606 represents an association between a RegistryEntry and another Object. An  
 607 example of such an association is between ExtrinsicObjects that catalogue a new  
 608 Collaboration Protocol Profile (CPP) and an older Collaboration Protocol Profile  
 609 where the newer CPP supersedes the older CPP as shown in Figure 3.



610  
 611  
 612

Figure 3: Example of Registry Entry Association

613 **10.1 Interface Association**

614 **All Superinterfaces:**

615 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

616 **All Known Subinterfaces:**

617 [Classification](#)

618  
 619

Association instances are used to define many-to-many associations between objects in the information model.

621  
 622

An instance of the Association class represents an association between two Objects.

623  
 624  
 625  
 626

**Method Summary**

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

627 **10.1.1 Pre-defined Association Types**

628 The following table lists pre-defined association types. These pre-defined  
 629 association types are defined as a Classification scheme. While the scheme may  
 630 easily be extended a registry must support the association types listed below.  
 631

name	description
RelatedTo	Defines that source object is an instance of target object.
Packages	Defines that the source Package object packages the target RegistryEntry object. Reserved for use in Packaging of Registry Entries.
ExternallyLinks	Defines that the source ExternalLink object externally

	links the target RegistryEntry object. Reserved for use in associating ExternalLinks with Registry Entries.
<b>ExternallyIdentifies</b>	Defines that the source ExternalIdentifier object identifies the target RegistryEntry object. Reserved for use in associating ExternalIdentifiers with Registry Entries.
<b>ContainedBy</b>	Defines that source object is contained by the target object.
<b>Contains</b>	Defines that source object contains the target object.
<b>Extends</b>	Defines that source object inherits from or specializes the target object.
<b>Implements</b>	Defines that source object implements the functionality defined by the target object.
<b>InstanceOf</b>	Defines that source object is an instance of target object.
<b>SupercededBy</b>	Defines that the source object is superseded by the target object.
<b>Supersedes</b>	Defines that the source object supersedes the target object.
<b>UsedBy</b>	Defines that the source object is used by the target object in some manner.
<b>Uses</b>	Defines that the source object uses the target object in some manner.
<b>ReplacedBy</b>	Defines that the source object is replaced by the target object in some manner.
<b>Replaces</b>	Defines that the source object replaces the target object in some manner.

632

633 **11 Classification of Registry Entries**

634 This section describes the how the information model supports classification of  
 635 RegistryEntries. It is a simplified version of the OASIS classification model [OAS].

636

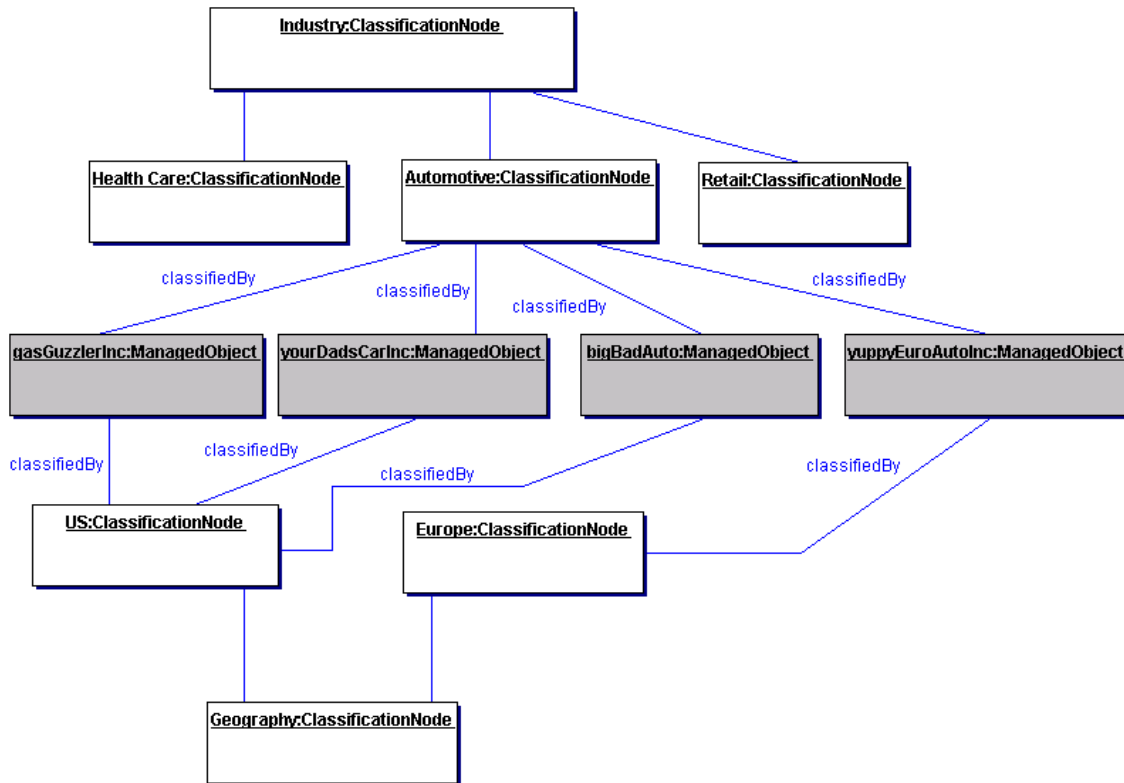
637 A RegistryEntry may be classified in many ways. For example the RegistryEntry  
 638 for the same Collaboration Protocol Profile (CPP) may be classified by its  
 639 industry, by the products it sells and by its geographical location.

640

641 A general classification scheme can be viewed as a classification tree. In the  
 642 example shown in Figure 4, RegistryEntries representing Collaboration Protocol  
 643 Profiles are shown as shaded boxes. Each Collaboration Protocol Profile  
 644 represents an automobile manufacturer. Each Collaboration Protocol Profile is  
 645 classified by the ClassificationNode named Automotive under the root  
 646 ClassificationNode named Industry. Furthermore, the US Automobile  
 647 manufacturers are classified by the US ClassificationNode under the Geography

648 ClassificationNode. Similarly, a European automobile manufacturer is classified  
 649 by the Europe ClassificationNode under the Geography ClassificationNode.

650  
 651 The example shows how a RegistryEntry may be classified by multiple  
 652 classification schemes. A classification scheme is defined by a  
 653 ClassificationNode that is the root of a classification tree (e.g. Industry,  
 654 Geography).

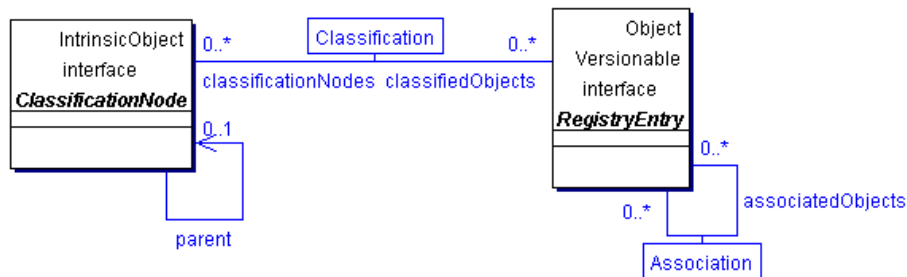


655  
 656

**Figure 4: Example showing a Classification Tree**

657 [Note]It is important to point out that the dark  
 658 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are  
 659 not part of the classification tree. The leaf  
 660 nodes of the classification tree are *Health*  
 661 *Care*, *Automotive*, *Retail*, *US* and *Europe*. The  
 662 dark nodes are associated with the  
 663 classification tree via a Classification  
 664 instance that is not shown in the picture  
 665

666 In order to support a general classification scheme that can support single level  
 667 as well as multi-level classifications, the information model defines the classes  
 668 and relationships shown in Figure 5.

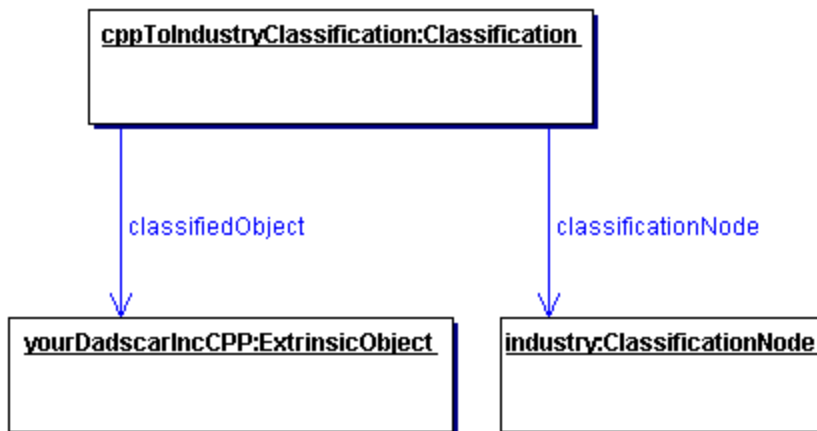


669

670

**Figure 5: Information Model Classification View**

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



675

676

**Figure 6: Classification Instance Diagram**

677 **11.1 Interface *ClassificationNode***

678 **All Superinterfaces:**

679 [IntrinsicObject](#), [RegistryEntry](#), [Object](#), [Versionable](#)

680  
 681 ClassificationNode instances are used to define tree structures where each node  
 682 in the tree is a ClassificationNode. Such classification trees constructed with  
 683 ClassificationNodes are used to define classification schemes or ontologies.

684 **See Also:**

685 [Classification](#)

686

687

Method Summary	
Collection	<a href="#">getClassifiedObjects()</a> Get the collection of ReaistrvEntries classified by

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

688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698

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`.

699 **11.2 Interface *Classification***

700 **All Superinterfaces:**

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

702  
703  
704  
705  
706

---

Classification instances are used to classify repository item by associating their `RegistryEntry` instance with a `ClassificationNode` instance within a classification scheme.

707 In Figure 4, `Classification` instances are not explicitly shown but are implied as  
708 associations between the `RegistryEntries` (shaded leaf node) and the associated  
709 `ClassificationNode`  
710

<b>Method Summary</b>	
<a href="#"><u>Object</u></a>	<b><a href="#"><u>getClassifiedObject()</u></a></b> Gets the Object that is classified by this Classification. Maps to attribute named <code>classifiedObject</code> .
<a href="#"><u>Object</u></a>	<b><a href="#"><u>getClassificationNode()</u></a></b> Gets the ClassificationNode that classifies the object in this Classification. Maps to attribute named <code>classificationNode</code> .

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

713 **11.2.1 Context Sensitive Classification**

714 Consider the case depicted in Figure 7 where a Collaboration Protocol Profile for  
 715 ACME Inc. is classified by the Japan ClassificationNode under the Geography  
 716 classification scheme. In the absence of the context for this classification its  
 717 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it  
 718 mean that ACME ships products to Japan, or does it have some other meaning?  
 719 To address this ambiguity a Classification may optionally be associated with  
 720 another ClassificationNode (in this example named `isLocatedIn`) that provides the  
 721 missing context for the Classification. Another Collaboration Protocol Profile for  
 722 MyParcelService may be classified by the Japan ClassificationNode where this  
 723 Classification is associated with a different ClassificationNode (e.g. named  
 724 `shipsTo`) to indicate a different context than the one used by ACME Inc.

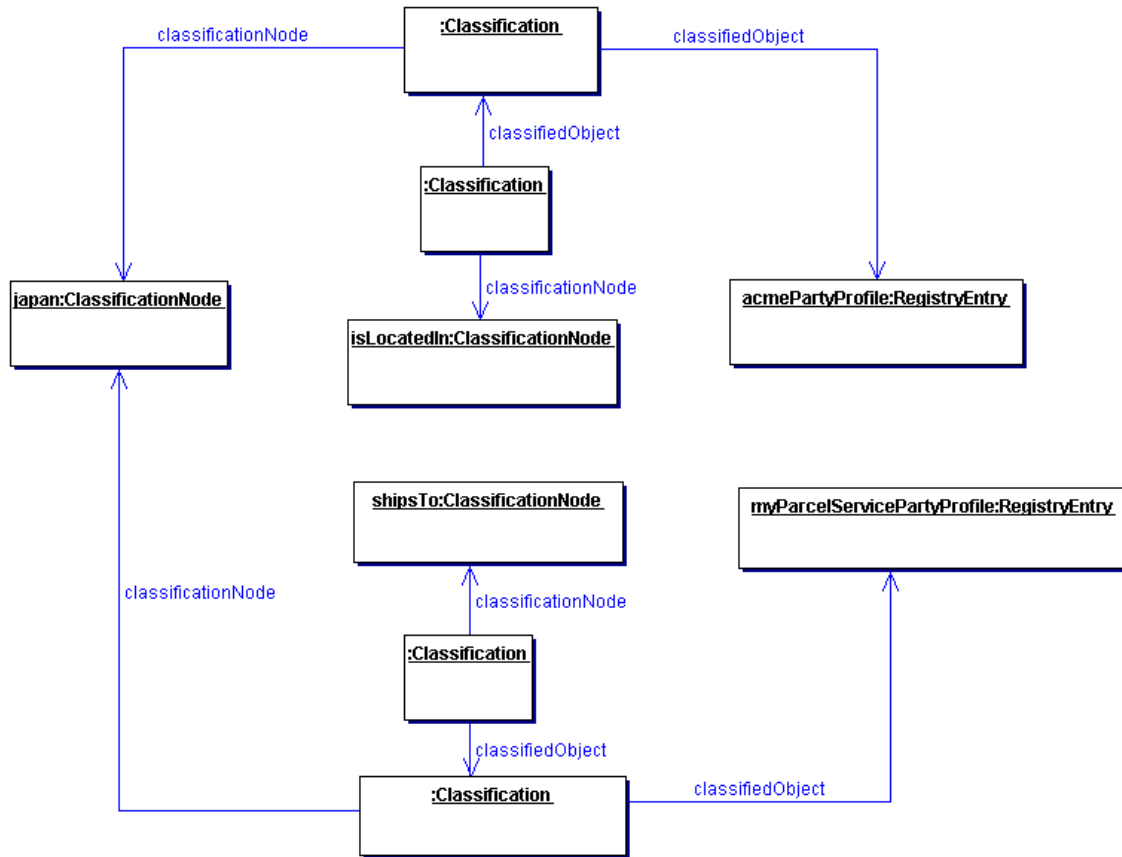


Figure 7: Context Sensitive Classification

725

726

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

730

731 In summary, the generalized support for classification schemes in the information  
 732 model allows:

- 733 o A RegistryEntry to be classified by defining a Classification that associates it
- 734 o with a ClassificationNode in a classification tree.
- 735 o A RegistryEntry to be classified along multiple facets by having multiple
- 736 o classifications that associate it with multiple ClassificationNodes.
- 737 o A classification defined for a RegistryEntry to be qualified by the contexts in
- 738 o which it is being classified.

739 **11.3 Example of Classification Schemes**

740 The following table lists some examples of possible classification schemes  
 741 enabled by the information model. These schemes are based on a subset of  
 742 contextual concepts identified by the ebXML Business Process and Core  
 743 Components Project Teams. This list is meant to be illustrative not prescriptive.

744

745

Classification Scheme (Context)	Usage Example
Industry	Find all Parties in Automotive industry
Process	Find a ServiceInterface that implements a Process
Product	Find a business 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 role of "Seller"

746

Table 1: Sample Classification Schemes

747

## 11.4 Standardized Taxonomy Support

748

749

750

751

752

753

754

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

The information model provides two alternatives for using standardized taxonomies for classification of RegistryItems.

755

### 11.4.1 Full-featured Taxonomy Based Classification

756

757

758

759

760

761

762

763

764

765

766

767

768

769

770

The information model provides a full-featured taxonomy based classification alternative based Classification and ClassificationNode instances. This alternative requires that a standard taxonomy be imported into the Registry as a classification tree consisting of ClassificationNode instances. This specification does not prescribe the transformation tools necessary to convert standard taxonomies into ebXML Registry classification trees. However, the transformation must ensure that:

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

771

### 11.4.2 Light Weight Taxonomy Based Classification

772

773

774

775

The information model also provides a lightweight alternative for classifying RegistryEntry instances by codes defined by standard taxonomies, where the submitter does not wish to import an entire taxonomy as a native classification scheme.

776

777 In this alternative the submitter adds one or more taxonomy related Slots to the  
778 RegistryEntry for a submitted repository item. Each Slot's name identifies a  
779 standardized taxonomy while the Slot's value is the code within the specified  
780 taxonomy. Such taxonomy related slots *must* be defined with a slotType of  
781 `Classification`.

782

783 For example if a RegistryEntry has a Slot with name "NAICS", a slotType of  
784 "Classification" and a value "51113" it implies that the RegistryEntry is classified  
785 by the code for "Book Publishers" in the NAICS taxonomy. Note that in this  
786 example, there is no need to import the entire NAICS taxonomy, nor is there any  
787 need to create instances of ClassificationNode or Classification.

788

789 The following points are noteworthy in this light weight classification alternative:

790 ?? Validation of the name and the value of the Classification" is responsibility  
791 of the SO and not of the ebXML Registry itself.

792 ?? Discovery is based on exact match on slot name and slot value rather  
793 than the flexible "browse and drill down discovery" available to the heavy  
794 weight classification alternative.

795

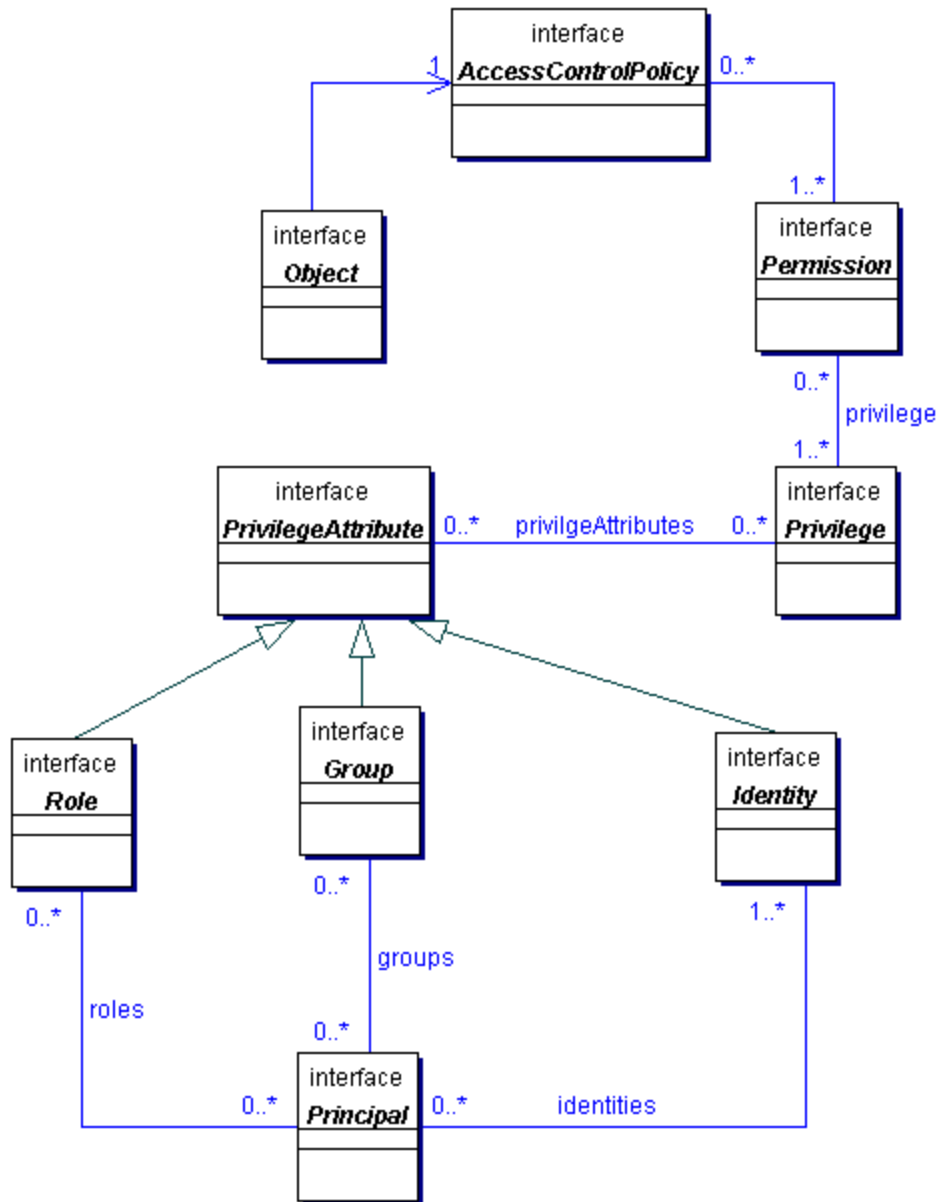
## 796 **12 Information Model: Security View**

797 This chapter describes the aspects of the information model that relate to the  
798 security features of the Registry.

799

800 Figure 8 shows the view of the objects in the Registry from a security  
801 perspective. It shows object relationships as a UML class diagram. It does not  
802 show class attributes or class methods that will be described in subsequent  
803 sections. It is meant to be illustrative not prescriptive.

804



805

806

Figure 8: Information Model: Security View

807 **12.1 Interface *AccessControlPolicy***

808 Every Object is associated with exactly one AccessControlPolicy which defines  
 809 the policy rules that govern access to operations or methods performed on that  
 810 Object. Such policy rules are defined as a collection of Permissions.

811

812

813

814

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

815

816 **12.2 Interface *Permission***

817

818 The Permission object is used for authorization and access control to Objects in  
 819 the Registry. The Permissions for an Object are defined in an  
 820 AccessControlPolicy object.

821

822 A Permission object authorizes access to a method in an Object if the requesting  
 823 Principal has *any* of the Privileges defined in the Permission.

824 **See Also:**

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

826

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

827

828 **12.3 Interface *Privilege***

829

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

832

833 A requesting Principal must have *all* of the PrivilegeAttributes specified in a  
 834 Privilege in order to gain access to a method in a protected Object. Permissions  
 835 defined in the Object's AccessControlPolicy define the Privileges that can  
 836 authorize access to specific methods.

837

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

840 **See Also:**

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

842

843

844

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

845

846 **12.4 Interface *PrivilegeAttribute***

847 **All Known Subinterfaces:**

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

849

850 PrivilegeAttribute is a common base class for all types of security attributes that  
 851 are used to grant specific access control privileges to a Principal. A Principal may  
 852 have several different types of PrivilegeAttributes. Specific combination of  
 853 PrivilegeAttributes may be defined as a Privilege object.

854 **See Also:**

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

856 **12.5 Interface *Role***

857 **All Superinterfaces:**

858 [PrivilegeAttribute](#)

859

860 A security Role PrivilegeAttribute. For example a hospital may have Roles such  
 861 as Nurse, Doctor, Administrator etc. Roles are used to grant Privileges to  
 862 Principals. For example a Doctor role may be allowed to write a prescription but a  
 863 Nurse role may not.

864 **12.6 Interface *Group***

865 **All Superinterfaces:**

866 [PrivilegeAttribute](#)

867

868 A security Group PrivilegeAttribute. A Group is an aggregation of users that may  
 869 have different roles. For example a hospital may have a Group defined for  
 870 Nurses and Doctors that are participating in a specific clinical trial (e.g.  
 871 AspirinTrial group). Groups are used to grant Privileges to Principals. For  
 872 example the members of the AspirinTrial group may be allowed to write a  
 873 prescription for Aspirin (even though Nurse role as a rule may not be allowed to  
 874 write prescriptions).

875 **12.7 Interface *Identity***

876 **All Superinterfaces:**

877 [PrivilegeAttribute](#)

878

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

882 **12.8 Interface *Principal***

883

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

890 **See Also:**

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

892

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

893

894

## 894 **13 References**

- 895 [GLS] ebXML Glossary, [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)
- 896 [TA] ebXML Technical Architecture
- 897 [OAS] OASIS Information Model
- 898 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 899 [ISO] ISO 11179 Information Model
- 900 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 901
- 902 [BDM] Registry and Repository: Business Domain Model
- 903 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 904 [RS] ebXML Registry Services Specification
- 905 [http://www.ebxml.org/project\\_teams/registry/private/RegistryServicesSpec](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf)
- 906 [ificationv0.83.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf)
- 907 [BPM] ebXML Business Process Metamodel Specification Schema
- 908 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 909 [CPA] Trading-Partner Specification
- 910 [http://www.ebxml.org/project\\_teams/trade\\_partner/private/](http://www.ebxml.org/project_teams/trade_partner/private/)
- 911 [CTB] Context table informal document from Core Components
- 912 [http://www.ebxml.org/project\\_teams/core\\_components/ContextTable.doc](http://www.ebxml.org/project_teams/core_components/ContextTable.doc)
- 913 [XPath] XML Path Language (XPath) Version 1.0
- 914 <http://www.w3.org/TR/xpath>
- 915

## 916 **14 Disclaimer**

- 917 The views and specification expressed in this document are those of the authors
- 918 and are not necessarily those of their employers. The authors and their
- 919 employers specifically disclaim responsibility for any problems arising from
- 920 correct or incorrect implementation or use of this design.
- 921

921 **15 Contact Information**

922

923 Team Leader

924 Name: Scott Nieman  
 925 Company: Norstan Consulting  
 926 Street: 5101 Shady Oak Road  
 927 City, State, Postal Code: Minnetonka, MN 55343  
 928 Country: USA  
 929 Phone: 952.352.5889  
 930 Email: Scott.Nieman@Norstan

931

932 Vice Team Lead

933 Name: Yutaka Yoshida  
 934 Company: Sun Microsystems  
 935 Street: 901 San Antonio Road, MS UMPK17-102  
 936 City, State, Postal Code: Palo Alto, CA 94303  
 937 Country: USA  
 938 Phone: 650.786.5488  
 939 Email: Yutaka.Yoshida@eng.sun.com

940

941 Editor

942 Name: Farrukh S. Najmi  
 943 Company: Sun Microsystems  
 944 Street: 1 Network Dr., MS BUR02-302  
 945 City, State, Postal Code: Burlington, MA, 01803-0902  
 946 Country: USA  
 947 Phone: 781.442.0703  
 948 Email: najmi@east.sun.com

949

950

**950 Copyright Statement**

951 Copyright © ebXML 2000. All Rights Reserved.

952

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

958 However, this document itself may not be modified in any way, such as by  
959 removing the copyright notice or references to the Internet Society or other  
960 Internet organizations, except as needed for the purpose of developing Internet  
961 standards in which case the procedures for copyrights defined in the Internet  
962 Standards process must be followed, or as required to translate it into languages  
963 other than English.

964

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

967

968 This document and the information contained herein is provided on an  
969 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR  
970 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE  
971 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
972 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
973 PARTICULAR PURPOSE.