



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 3/17/2001

This version: Version 0.59

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/RegistryInfoModelv0.59.pdf

Latest version:

http://www.ebxml.org/project_teams/registry/private/RegistryInfoModel.pdf

Previous version:

http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.58.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 INTRODUCTION6**

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

72 3.2 GENERAL CONVENTIONS6

73 3.3 AUDIENCE.....6

74 3.4 RELATED DOCUMENTS7

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

76 4.1 GOALS7

77 4.2 CAVEATS AND ASSUMPTIONS7

78 **5 SYSTEM OVERVIEW7**

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

80 5.2 REGISTRY SERVICES8

81 5.3 WHAT THE REGISTRY INFORMATION MODEL DOES8

82 5.4 HOW THE REGISTRY INFORMATION MODEL WORKS8

83 5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED8

84 **6 REGISTRY INFORMATION MODEL: PUBLIC VIEW.....8**

85 6.1 REGISTRYENTRY9

86 6.2 SLOT10

87 6.3 ASSOCIATION.....10

88 6.4 EXTERNALIDENTIFIER10

89 6.5 EXTERNALLINK10

90 6.6 CLASSIFICATIONNODE.....10

91 6.7 CLASSIFICATION10

92 6.8 PACKAGE10

93 6.9 AUDITABLEEVENT.....11

94 6.10 USER.....11

95 6.11 POSTALADDRESS11

96 6.12 ORGANIZATION.....11

97 **7 REGISTRY INFORMATION MODEL: DETAIL VIEW.....11**

98 7.1 INTERFACE *OBJECT*.....12

99 7.2 INTERFACE *VERSIONABLE*.....14

100 7.3 INTERFACE *REGISTRYENTRY*.....14

101 7.3.1 *Pre-defined RegistryEntry Status Types*16

102 7.3.2 *Pre-Defined Object Types*.....17

103 7.3.3 *Pre-defined RegistryEntry Stability Enumerations*18

104 7.4 *INTERFACE SLOT*18

105 7.5 *INTERFACE EXTRINSIC OBJECT*.....19

106 7.6 *INTERFACE INTRINSIC OBJECT*.....20

107 7.7 *INTERFACE PACKAGE*.....20

108 7.8 *INTERFACE EXTERNALIDENTIFIER*.....21

109 7.9 *INTERFACE EXTERNALLINK*.....21

110 **8 REGISTRY AUDIT TRAIL**.....**22**

111 8.1 *INTERFACE AUDITABLEEVENT*22

112 8.1.1 *Pred-defined Auditable Event Types*23

113 8.2 *INTERFACE USER*.....23

114 8.3 *INTERFACE ORGANIZATION*.....24

115 8.4 *CLASS POSTALADDRESS*25

116 8.5 *CLASS TELEPHONENUMBER*26

117 8.6 *CLASS PERSONNAME*.....26

118 **9 REGISTRY ENTRY NAMING****26**

119 **10 ASSOCIATION OF REGISTRY ENTRIES****27**

120 10.1 *INTERFACE ASSOCIATION*.....27

121 10.1.1 *Pre-defined Association Types*28

122 **11 CLASSIFICATION OF REGISTRY ENTRIES****29**

123 11.1 *INTERFACE CLASSIFICATIONNODE*.....31

124 11.2 *INTERFACE CLASSIFICATION*32

125 11.2.1 *Context Sensitive Classification*33

126 11.3 *EXAMPLE OF CLASSIFICATION SCHEMES*34

127 11.4 *STANDARDIZED TAXONOMY SUPPORT*35

128 11.4.1 *Full-featured Taxonomy Based Classification*35

129 11.4.2 *Light Weight Taxonomy Based Classification*35

130 **12 INFORMATION MODEL: SECURITY VIEW****36**

131 12.1 *INTERFACE ACCESSCONTROLPOLICY*.....37

132 12.2 *INTERFACE PERMISSION*38

133 12.3 *INTERFACE PRIVILEGE*.....38

134 12.4 *INTERFACE PRIVILEGEATTRIBUTE*.....39

135 12.5 *INTERFACE ROLE*39

136 12.6 *INTERFACE GROUP*.....39

137 12.7 *INTERFACE IDENTITY*39

138 12.8 *INTERFACE PRINCIPAL*.....40

139 **13 REFERENCES****41**

140 **14 DISCLAIMER****41**

141 **15 CONTACT INFORMATION**.....**42**

142 **COPYRIGHT STATEMENT**.....43

143 **Table of Figures**

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

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

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

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

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

149 Figure 6: Classification Instance Diagram 31

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

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

152 **Table of Tables**

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

154

155

155 **3 Introduction**

156 **3.1 Summary of Contents of Document**

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

158

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

162 **3.2 General Conventions**

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

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

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

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

179

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

183

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

186

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

190 **3.3 Audience**

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

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

195 **3.4 Related Documents**

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

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

207 **4 Design Objectives**

208 **4.1 Goals**

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

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

217 **4.2 Caveats and Assumptions**

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

221 **5 System Overview**

222 **5.1 Role of ebXML Registry**

223 The Registry provides a stable store where content submitted by a Submitting
224 Organization is persisted. Such content is used to facilitate ebXML-based
225 business to business (B2B) partnerships and transactions. Submitted content
226 may be XML schema and documents, process descriptions, UML models,
227 information about parties and even software components.

228 **5.2 Registry Services**

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

233 **5.3 What the Registry Information Model Does**

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

238 The Registry information model:

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

242

243 **5.4 How the Registry Information Model Works**

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

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

253 **5.5 Where the Registry Information Model May Be Implemented**

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

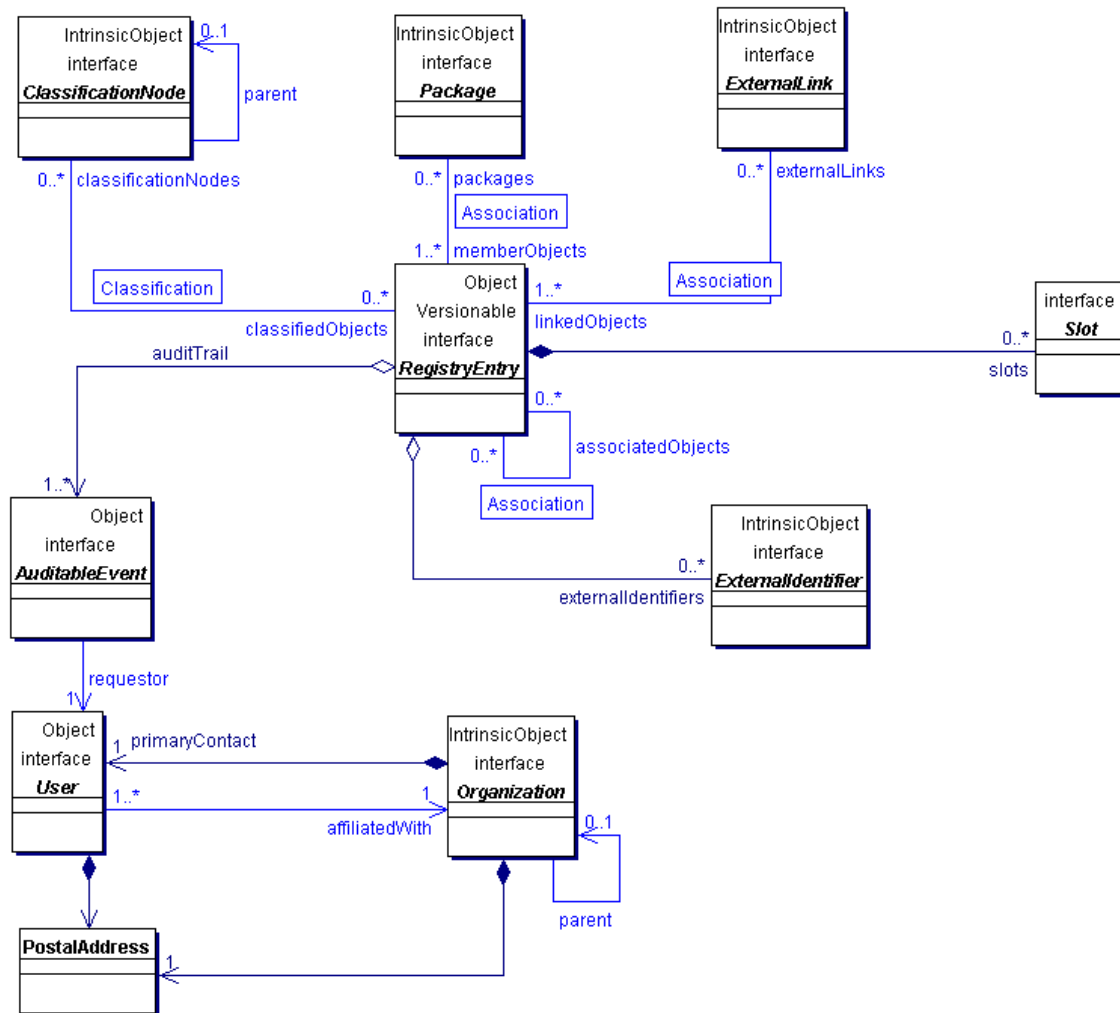
258 **6 Registry Information Model: Public View**

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

261

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

265 The reader is again reminded that the information model is not modeling actual
 266 repository items.
 267



268
 269

Figure 1: Information Model Public View

270 **6.1 RegistryEntry**

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

279 **6.2 Slot**

280 Slot instances provide a dynamic way to add arbitrary attributes to RegistryEntry
281 instances. This ability to add attributes dynamically to RegistryEntry instances
282 enables extensibility within the Registry Information Model.

283 **6.3 Association**

284 Association instances are RegistryEntries that are used to define many-to-many
285 associations between objects in the information model. Associations are
286 described in detail in chapter 10.

287 **6.4 ExternalIdentifier**

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

291 **6.5 ExternalLink**

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

302 **6.6 ClassificationNode**

303 ClassificationNode instances are RegistryEntries that are used to define tree
304 structures where each node in the tree is a ClassificationNode. Classification
305 trees constructed with ClassificationNodes are used to define classification
306 schemes or ontologies. ClassificationNode is described in detail in chapter 11.

307 **6.7 Classification**

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

311 **6.8 Package**

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

316 **6.9 AuditableEvent**

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

319 **6.10 User**

320 User instances are Objects that are used to provide information about registered
321 users within the registry. User objects are used in audit trail for RegistryEntries.
322 User is described in detail in chapter 8.
323

324 **6.11 PostalAddress**

325 PostalAddress is a simple reusable entity class that defines attributes of a postal
326 address.
327

328 **6.12 Organization**

329 Organization instances are RegistryEntries that provide information on
330 organizations such as a Submitting Organization. Each Organization instance
331 may have a reference to a parent Organization.

332 **7 Registry Information Model: Detail View**

333 This chapter covers the information model classes in more detail than the Public
334 View. The detail view introduces some additional classes within the model that
335 were not described in the public view of the information model.
336

337 Figure 2 shows the inheritance or “is a” relationships between the classes in the
338 information model. Note that it does not show the other types of relationships,
339 such as “has a” relationships, since they have already been shown in a previous
340 figure. Class attributes and class methods are also not shown. Detailed
341 description of methods and attributes of most interfaces and classes will be
342 displayed in tabular form following the description of each class in the model.
343

344 The interface Association will be covered in detail separately in chapter 10. The
345 interfaces Classification and ClassificationNode will be covered in detail
346 separately in chapter 11.
347

348 The reader is again reminded that the information model is not modeling actual
349 repository items.

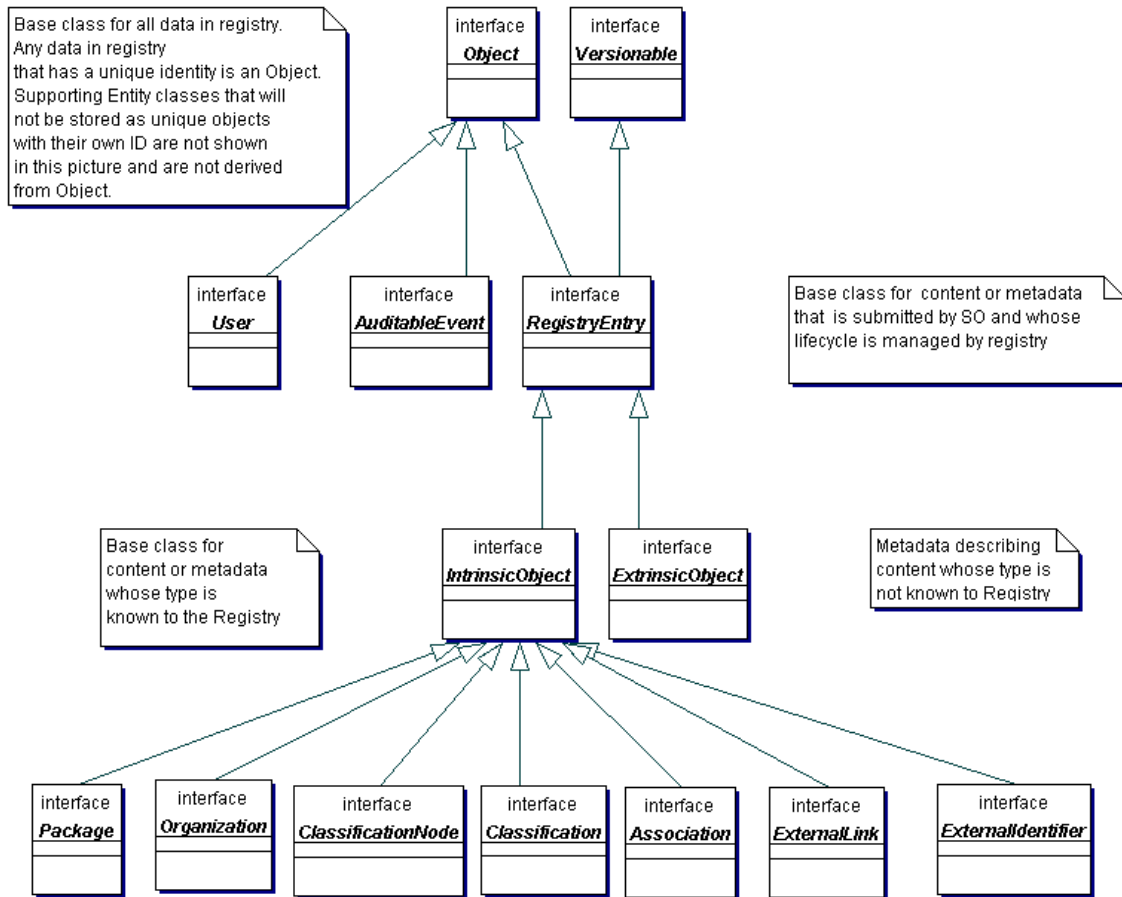


Figure 2: Information Model Inheritance View

350
351
352

7.1 Interface Object

All Known Subinterfaces:

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

353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368

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

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

Method Summary	
AccessControlPolicy	<p>getAccessControlPolicy() 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>getDescription() Gets the context independent textual description for this object. Maps to attribute named <code>description</code>.</p>
String	<p>getName() Gets user friendly context independent name of object in repository. Maps to attribute named <code>name</code>.</p>
String	<p>getID() Gets the universally unique ID (UUID) for this object. Maps to attribute named <code>id</code>.</p>
void	<p>setDescription(String description) Sets the context independent textual description for this object.</p>
void	<p>setName(String name) Sets user friendly context independent name of object in repository.</p>
void	<p>setID(String id) Sets the universally unique ID (UUID) for this object.</p>

369
370

370 **7.2 Interface *Versionable***

371 **All Known Subinterfaces:**

372 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
 373 [ExtrinsicObject](#), [IntrinsicObject](#), [RegistryEntry](#), [Organization](#), [Package](#)

374

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

378

Method Summary	
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) Gets 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.

379

380 **7.3 Interface *RegistryEntry***

381 **All Superinterfaces:**

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

383 **All Known Subinterfaces:**

384 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
 385 [ExtrinsicObject](#), [IntrinsicObject](#), [Organization](#), [Package](#)

386

387 RegistryEntry is a common base class for all metadata describing submitted
 388 content whose life cycle is managed by the registry. Metadata describing content
 389 submitted to the registry is further specialized by the ExtrinsicObject and
 390 IntrinsicObject subclasses of RegistryEntry.

391
 392
 393
 394

Method Summary	
Collection	<p>getAssociatedObjects() Returns the collection of Objects associated with this object. 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 object as an ordered Collection of <code>AuditableEvent</code> objects. Maps to attribute named <code>auditTrail</code>.</p>
Collection	<p>getClassificationNodes() Returns the collection of <code>ClassificationNodes</code> associated with this object. Maps to attribute named <code>classificationNodes</code>.</p>
Collection	<p>getExternalLinks() Returns the collection of <code>ExternalLinks</code> associated with this object. Maps to attribute named <code>externalLinks</code>.</p>
String	<p>getObjectType() 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>getPackages() Returns the collection of <code>Packages</code> associated with this object. Maps to attribute named <code>packages</code>.</p>
String	<p>getStatus() 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>getUserVersion() 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>setUserVersion(String UserVersion) Sets the <code>userVersion</code> attribute of the <code>RegistryEntry</code> within the Registry.</p>
String	<p>getStability() 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>getExpirationDate() 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	setExpirationDate (Date expirationDate) Sets expirationDate attribute of the RegistryEntry within the Registry.
Collection	getSlots () Gets the collection of slots that have been dynamically added to this object. Maps to attribute named slots.
void	addSlots (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	removeSlots (Collection slotNames) Removes one or more slots from this object. Slots to be removed are identified by their name.

395

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

396

Methods inherited from interface
getMajorVersion , getMinorVersion , setMajorVersion , setMinorVersion

397 **7.3.1 Pre-defined RegistryEntry Status Types**

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

402

Name	Description
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.
Withdrawn	Status of a RegistryEntry that catalogues content that has been withdrawn from the Registry.

403 **7.3.2 Pre-Defined Object Types**

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

408
 409 These pre-defined object types are defined as a Classification scheme. While the
 410 scheme may easily be extended a registry must support the object types listed
 411 below.

412

name	description
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	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.
CPP	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.
Process	An ExtrinsicObject of this type catalogues a process description document.
Role	An ExtrinsicObject of this type catalogues an XML description of a Role in a Collaboration Protocol Profile (CPP).
ServiceInterface	An ExtrinsicObject of this type catalogues an XML description of a service interface as defined by [CPA].
SoftwareComponent	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or class library).
Transport	An ExtrinsicObject of this type catalogues an XML description of a transport configuration as defined by [CPA].
UMLModel	An ExtrinsicObject of this type catalogues a UML model.
XMLSchema	An ExtrinsicObject of this type catalogues an XML schema

	(DTD, XML Schema, RELAX grammar, 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

413

414 **7.3.3 Pre-defined RegistryEntry Stability Enumerations**

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

419

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.

420

421

422 **7.4 Interface Slot**

423

424 Slot instances provide a dynamic way to add arbitrary attributes to RegistryEntry
 425 instances. This ability to add attributes dynamically to RegistryEntry instances
 426 enables extensibility within the Registry Information Model.

427

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

432 may be a collection, therefore a Slot is allowed to have a collection of values
 433 rather than a single value. The slotType attribute may optionally specify a type or
 434 category for the slot.
 435
 436

Method Summary	
String	getName () Gets the name of this object. Maps to attribute named name.
void	setName (String name) Sets the name of this object. Slot names are locally unique within a RegistryEntry instance.
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 object. The type for each value is String. Maps to attribute named values.
void	setValues (Collection values) Sets the collection of values for this object.

437

438 **7.5 Interface *ExtrinsicObject***

439 **All Superinterfaces:**

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

441

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

445

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

448

Method Summary	
String	getContentURI () 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	getMimeType() Gets the mime type associated with the content catalogued by this <code>ExtrinsicObject</code> . Maps to attribute named <code>mimeType</code> .
boolean	isOpaque() 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	setContentURI(String uri) Sets the URI to the content catalogued by this <code>ExtrinsicObject</code> .
void	setMimeType(String mimeType) Sets the mime type associated with the content catalogued by this <code>ExtrinsicObject</code> .
void	setOpaque(boolean isOpaque) Sets whether the content catalogued by this <code>ExtrinsicObject</code> is opaque to (not readable by) the Registry.

449

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

452 **7.6 Interface *IntrinsicObject***

453 **All Superinterfaces:**

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

455 **All Known Subinterfaces:**

456 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#), [Organization](#),
457 [Package](#)

458

459

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

462

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

465

466 **7.7 Interface *Package***

467 **All Superinterfaces:**

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

469

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

473
 474

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

475

476 **7.8 Interface *ExternalIdentifier***

477 **All Superinterfaces:**

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

479

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

486 **See Also:**

487

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

488

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

491 **7.9 Interface *ExternalLink***

492 **All Superinterfaces:**

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

494

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

498

499

Method Summary	
Collection	getLinkedObjects () Gets the collection of objects 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.

500
501
502

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

503 **8 Registry Audit Trail**

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

509
510
511
512
513
514
515

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`.

516 **8.1 Interface *AuditableEvent***

517 **All Superinterfaces:**

518 [Object](#)

519
520
521
522
523
524

`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`.

525
526
527

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

528
529
530
531

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 state of a `RegistryEntry`. Specifically, read-only requests do not generate an

532 AuditableEvent. No AuditableEvent is generated for a RegistryEntry when it is
 533 classified, assigned to a Package or associated with another Object.
 534
 535

536 **8.1.1 Pred-defined Auditable Event Types**

537 The following table lists pre-defined auditable event types. These pre-defined
 538 event types are defined as a Classification scheme. While the scheme may
 539 easily be extended, a registry must support the event types listed below.
 540

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.

541

Method Summary	
User	getUser() Gets the User that sent the request that generated this event. Maps to attribute named <code>user</code> .
String	getEventType() 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> .
RegistryEntry	getRegistryEntry() Gets the RegistryEntry associated with this AuditableEvent. Maps to attribute named <code>registryEntry</code> .
Timestamp	getTimestamp() Gets the Timestamp for when this event occurred. Maps to attribute named <code>timestamp</code> .

542

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

545 **8.2 Interface *User***

546 **All Superinterfaces:**

547 [Object](#)

548

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

551

Method Summary	
Organization	getOrganization() Gets the Submitting Organization 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	getName() Name of contact person. Maps to attribute named <code>name</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 URL to the web page for this contact. Maps to attribute named <code>url</code> .

552

553 **8.3 Interface *Organization***

554 **All Superinterfaces:**

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

556

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

560 primary contact within the organization. An Organization also has an address
 561 attribute.

562 **See Also:**

563

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

564

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

567

568 **8.4 Class *PostalAddress***

569

570

571 PostalAddress is a simple reusable entity class that defines attributes of a postal
 572 address.

573

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

574

575 **8.5 Class TelephoneNumber**

576
577

578
579
580

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

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 URL that can dial this number electronically

581

582 **8.6 Class PersonName**

583
584
585
586

A simple entity class for a person's name.

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.

587

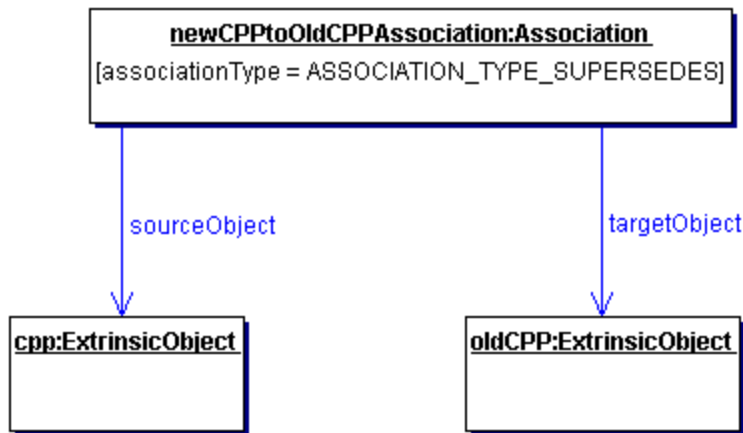
588 **9 Registry Entry Naming**

589 A RegistryEntry has a name that may or may not be unique within the Registry.
590
591 In addition a RegistryEntry may have any number of context sensitive alternate
592 names that are valid only in the context of a particular classification scheme.

593 Alternate contextual naming will be addressed in a later version of the Registry
 594 Information Model.
 595

596 **10 Association of Registry Entries**

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



603
 604
 605

Figure 3: Example of Registry Entry Association

606 **10.1 Interface Association**

607 **All Superinterfaces:**

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

609 **All Known Subinterfaces:**

610 [Classification](#)

611
 612

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

613
 614

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

615
 616
 617
 618
 619

Method Summary

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 Object that is the source of this Association. Maps to attribute named <code>sourceObject</code> .
String	getSourceRole() Gets the name of the role played by the source Object in this Association. Maps to attribute named <code>sourceRole</code> .
Object	getTargetObject() Gets the Object that is the target of this Association. Maps to attribute named <code>targetObject</code> .
String	getTargetRole() Gets the name of the role played by the target Object 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 role played by the source Object in this Association.
void	setTargetRole(String targetRole) Sets the name of the role played by the destination Object in this Association.

620 **10.1.1 Pre-defined Association Types**

621 The following table lists pre-defined association types. These pre-defined
 622 association types are defined as a Classification scheme. While the scheme may
 623 easily be extended a registry must support the association types listed below.
 624

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

625

626 **11 Classification of Registry Entries**

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

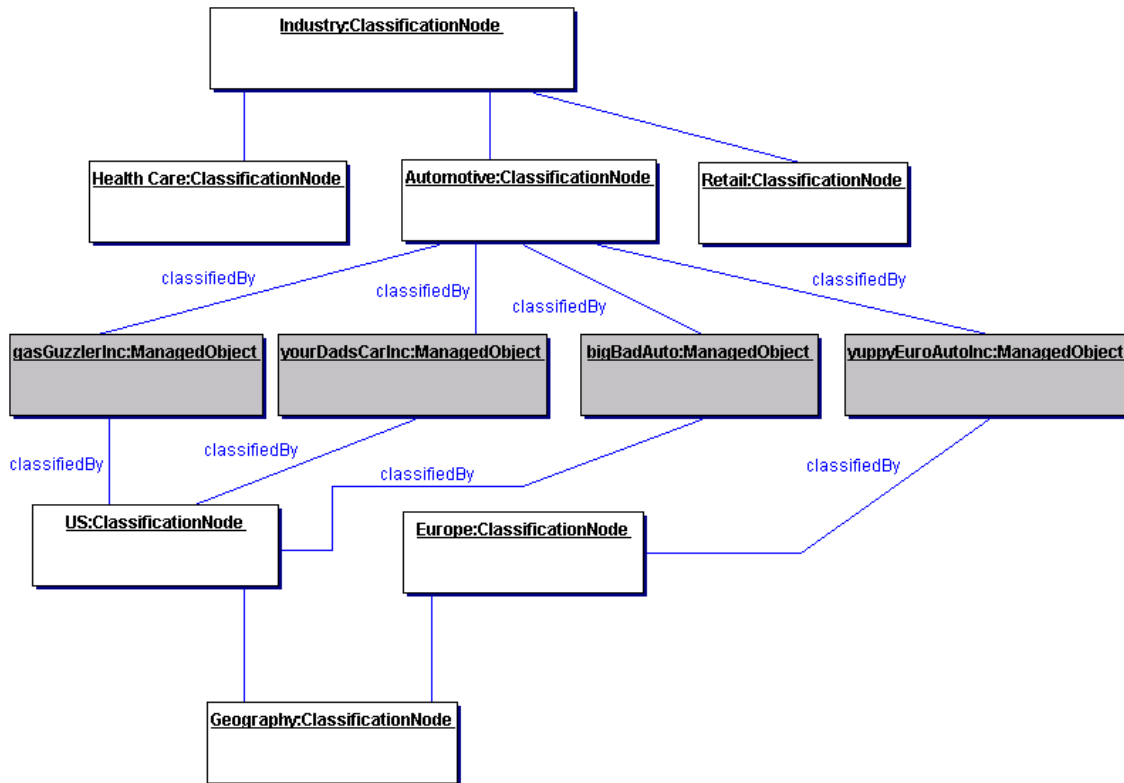
629

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

633

634 A general classification scheme can be viewed as a classification tree. In the
 635 example shown in Figure 4, RegistryEntries representing Collaboration Protocol
 636 Profiles are shown as shaded boxes. Each Collaboration Protocol Profile
 637 represents an automobile manufacturer. Each Collaboration Protocol Profile is
 638 classified by the ClassificationNode named Automotive under the root
 639 ClassificationNode named Industry. Furthermore, the US Automobile
 640 manufacturers are classified by the US ClassificationNode under the Geography

641 ClassificationNode. Similarly, a European automobile manufacturer is classified
 642 by the Europe ClassificationNode under the Geography ClassificationNode.
 643
 644 The example shows how a RegistryEntry may be classified by multiple
 645 classification schemes. A classification scheme is defined by a
 646 ClassificationNode that is the root of a classification tree (e.g. Industry,
 647 Geography).

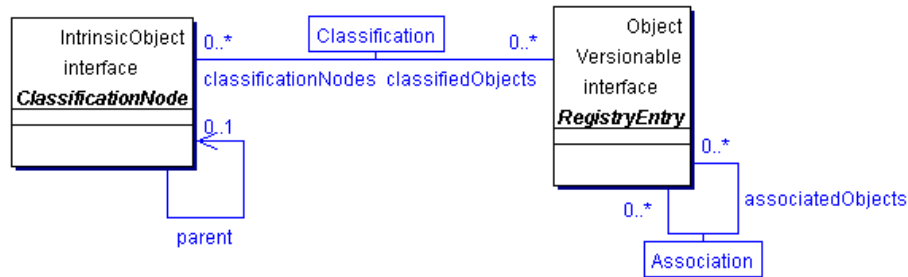


648
 649

Figure 4: Example showing a Classification Tree

650 [Note]It is important to point out that the dark
 651 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are
 652 not part of the classification tree. The leaf
 653 nodes of the classification tree are *Health*
 654 *Care*, *Automotive*, *Retail*, *US* and *Europe*. The
 655 dark nodes are associated with the
 656 classification tree via a Classification
 657 instance that is not shown in the picture
 658

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

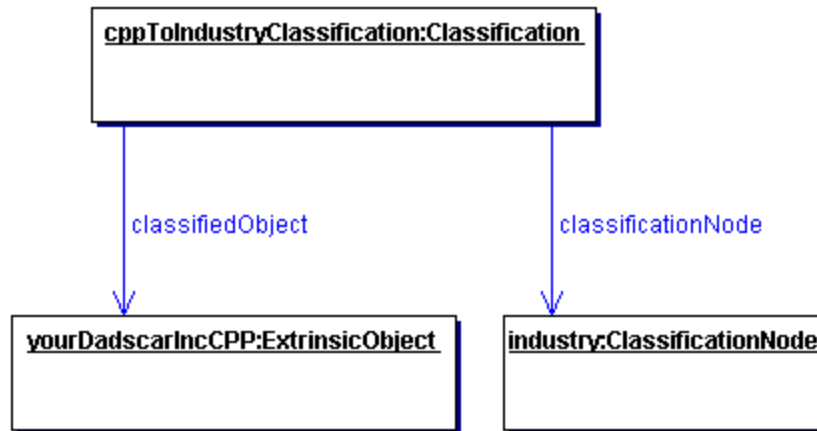


662

663

Figure 5: Information Model Classification View

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



668

669

Figure 6: Classification Instance Diagram

670 **11.1 Interface *ClassificationNode***

671 **All Superinterfaces:**

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

673

674 ClassificationNode instances are used to define tree structures where each node
 675 in the tree is a ClassificationNode. Such classification trees constructed with
 676 ClassificationNodes are used to define classification schemes or ontologies.

677 **See Also:**

678 [Classification](#)

679

680

Method Summary	
Collection	getClassifiedObjects() Get the collection of ReaistrvEntries classified by

	this <code>ClassificationNode</code> . Maps to attribute named <code>classifiedObjects</code> .
ClassificationNode	getParent() Gets the parent <code>ClassificationNode</code> for this <code>ClassificationNode</code> . Maps to attribute named <code>parent</code> .
String	getPath() 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	setParent(ClassificationNode parent) Sets the parent <code>ClassificationNode</code> for this <code>ClassificationNode</code> .
String	getCode() Gets the code for this <code>ClassificationNode</code> . See [11.4] for details. Maps to attribute named <code>code</code> .
void	setCode(String code) Sets the parent code for this <code>ClassificationNode</code> . See [11.4] for details.

681
682
683
684
685
686
687
688
689
690
691

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`.

692 **11.2 Interface *Classification***

693 **All Superinterfaces:**

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

695
696
697
698
699

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

700
701
702
703

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

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

706 **11.2.1 Context Sensitive Classification**

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

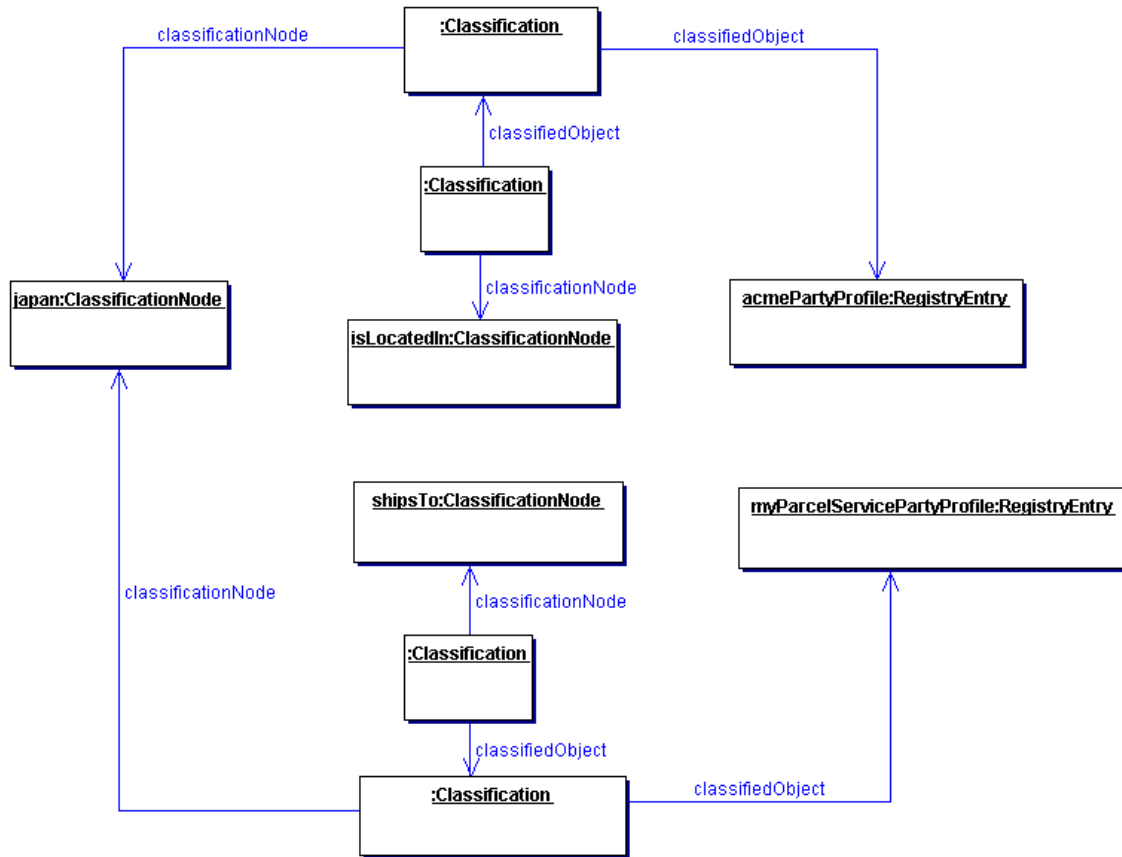


Figure 7: Context Sensitive Classification

718
719

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

723

724 In summary, the generalized support for classification schemes in the information
725 model allows:

- 726 o A RegistryEntry to be classified by defining a Classification that associates it
- 727 o with a ClassificationNode in a classification tree.
- 728 o A RegistryEntry to be classified along multiple facets by having multiple
- 729 o classifications that associate it with multiple ClassificationNodes.
- 730 o A classification defined for a RegistryEntry to be qualified by the contexts in
- 731 o which it is being classified.

732 11.3 Example of Classification Schemes

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

737

738

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"

739

Table 1: Sample Classification Schemes

740 **11.4 Standardized Taxonomy Support**

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

745

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

748 **11.4.1 Full-featured Taxonomy Based Classification**

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

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

764 **11.4.2 Light Weight Taxonomy Based Classification**

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

769

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

775

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

781

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

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

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

788

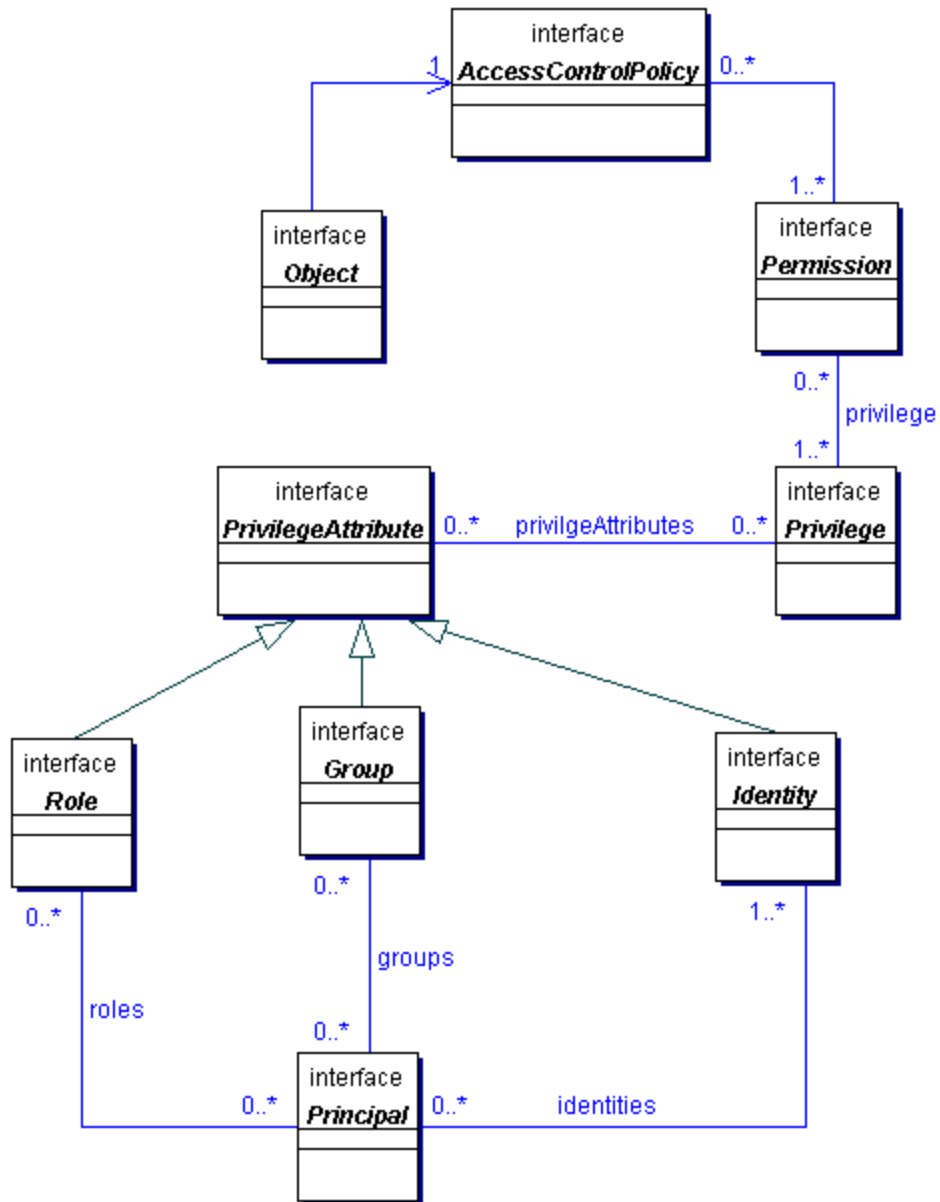
789 **12 Information Model: Security View**

790 This chapter describes the aspects of the information model that relate to the
791 security features of the Registry.

792

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

797



798

799

Figure 8: Information Model: Security View

800 **12.1 Interface *AccessControlPolicy***

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

804

805

806

807

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

808

809 **12.2 Interface *Permission***

810

811 The Permission object is used for authorization and access control to Objects in
 812 the Registry. The Permissions for an Object are defined in an
 813 AccessControlPolicy object.

814

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

817 **See Also:**

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

819

Method Summary	
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> .

820

821 **12.3 Interface *Privilege***

822

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

825

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

830

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

833 **See Also:**

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

835

836

837

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

838

839 **12.4 Interface *PrivilegeAttribute***

840 **All Known Subinterfaces:**

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

842

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

847 **See Also:**

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

849 **12.5 Interface *Role***

850 **All Superinterfaces:**

851 [PrivilegeAttribute](#)

852

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

857 **12.6 Interface *Group***

858 **All Superinterfaces:**

859 [PrivilegeAttribute](#)

860

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

868 **12.7 Interface *Identity***

869 **All Superinterfaces:**

870

[PrivilegeAttribute](#)

871

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

875 **12.8 Interface *Principal***

876

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

883 **See Also:**

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

885

Method Summary	
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> .

886

887

887 **13 References**

- 888 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 889 [TA] ebXML Technical Architecture
- 890 [OAS] OASIS Information Model
- 891 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>
- 892 [ISO] ISO 11179 Information Model
- 893 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 894
- 895 [BDM] Registry and Repository: Business Domain Model
- 896 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 897 [RS] ebXML Registry Services Specification
- 898 http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf
- 899
- 900 [BPM] ebXML Business Process Metamodel Specification Schema
- 901 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 902 [CPA] Trading-Partner Specification
- 903 http://www.ebxml.org/project_teams/trade_partner/private/
- 904 [CTB] Context table informal document from Core Components
- 905 http://www.ebxml.org/project_teams/core_components/ContextTable.doc
- 906 [XPATH] XML Path Language (XPath) Version 1.0
- 907 <http://www.w3.org/TR/xpath>
- 908

909 **14 Disclaimer**

- 910 The views and specification expressed in this document are those of the authors
- 911 and are not necessarily those of their employers. The authors and their
- 912 employers specifically disclaim responsibility for any problems arising from
- 913 correct or incorrect implementation or use of this design.
- 914

914 **15 Contact Information**

915

916 Team Leader

917 Name: Scott Nieman
 918 Company: Norstan Consulting
 919 Street: 5101 Shady Oak Road
 920 City, State, Postal Code: Minnetonka, MN 55343
 921 Country: USA
 922 Phone: 952.352.5889
 923 Email: Scott.Nieman@Norstan

924

925 Vice Team Lead

926 Name: Yutaka Yoshida
 927 Company: Sun Microsystems
 928 Street: 901 San Antonio Road, MS UMPK17-102
 929 City, State, Postal Code: Palo Alto, CA 94303
 930 Country: USA
 931 Phone: 650.786.5488
 932 Email: Yutaka.Yoshida@eng.sun.com

933

934 Editor

935 Name: Farrukh S. Najmi
 936 Company: Sun Microsystems
 937 Street: 1 Network Dr., MS BUR02-302
 938 City, State, Postal Code: Burlington, MA, 01803-0902
 939 Country: USA
 940 Phone: 781.442.0703
 941 Email: najmi@east.sun.com

942

943

943 Copyright Statement

944 Copyright © ebXML 2000. All Rights Reserved.

945

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

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

957

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

960

961 This document and the information contained herein is provided on an
962 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR
963 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE
964 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
965 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
966 PARTICULAR PURPOSE.