



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 Working Group**

**Working Draft 12/27/2000 4:40 AM**

**This version: Version 0.52**

## **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/registryInfoModelv0.52.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.52.pdf)

***Latest version:***

[http://www.ebxml.org/project\\_teams/registry/private/registryInfoModelv0.52.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.52.pdf)

***Previous version:***

[http://www.ebxml.org/project\\_teams/registry/private/registryInfoModelv0.51.pdf](http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.51.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 Bob Miller - GXS

45 Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan

46 Ron Monzillo – Sun Microsystems

47 Joel Munter - Intel

48 Farrukh Najmi - Sun Microsystems

49 Scott Nieman - Norstan Consulting

50 Frank Olken – Lawrence Berkeley National Laboratory

51 Micheal Park - eSum Technologies

52 Bruce Peat - eProcess Solutions

53 Waqar Sadiq - Vitria

54 Krishna Sankar - CISCO

55 Kim Tae Soo - Government of Korea

56 Nikola Stojanovic -

57 David Webber - XML Global

58 Yutaka Yoshida - Sun Microsystems

59 Prasad Yendluri - webmethods

60 Peter Z. Zhoo - Knowledge For the new Millennium

61

62

62 **Table of Contents**

63

64 **1 STATUS OF THIS DOCUMENT.....1**

65 **2 EBXML PARTICIPANTS.....2**

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

67 3.1 SUMMARY OF CONTENTS OF DOCUMENT.....6

68 3.2 DOCUMENT VERSION HISTORY.....6

69 3.3 GENERAL CONVENTIONS .....6

70 3.4 AUDIENCE.....7

71 3.5 RELATED DOCUMENTS .....7

72 **4 DESIGN OBJECTIVES.....7**

73 4.1 GOALS .....7

74 4.2 CAVEATS AND ASSUMPTIONS .....8

75 **5 SYSTEM OVERVIEW .....8**

76 5.1 ROLE OF EBXML REGISTRY.....8

77 5.2 REGISTRY SERVICES .....8

78 5.3 WHAT THE REGISTRY INFORMATION MODEL DOES .....8

79 5.4 HOW THE REGISTRY INFORMATION MODEL WORKS .....8

80 5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED .....9

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

82 6.1 MANAGEDOBJECT .....10

83 6.2 ASSOCIATION.....10

84 6.3 EXTERNALLINK .....10

85 6.4 CLASSIFICATIONNODE.....10

86 6.5 CLASSIFICATION .....10

87 6.6 PACKAGE.....10

88 6.7 AUDITABLEEVENT.....10

89 6.8 POSTALADDRESS .....11

90 6.9 CONTACT.....11

91 6.10 ORGANIZATION.....11

92 **7 REGISTRY INFORMATION MODEL: DETAIL VIEW.....11**

93 7.1 INTERFACE *OBJECT*.....12

94 7.2 INTERFACE *VERSIONABLE*.....13

95 7.3 INTERFACE *MANAGEDOBJECT*.....14

96 7.4 INTERFACE *EXTRINSICOBJECT* *EXTRINSICOBJECT* .....15

97 7.4.1 *Pre-Defined Extrinsic Object Types* .....16

98 7.5 INTERFACE *INTRINSICOBJECT* *INTRINSICOBJECT* .....17

99 7.6 INTERFACE *EXTERNALLINK* EXTERNALLINK .....17

100 **8 REGISTRY AUDIT TRAIL.....18**

101 8.1 INTERFACE AUDITABLEEVENT .....18

102 8.2 INTERFACE AUDITABLEIDENTITY .....19

103 8.3 INTERFACE ORGANIZATION .....19

104 8.4 CLASS CONTACT.....20

105 8.5 CLASS POSTALADDRESS.....21

106 8.6 CLASS TELEPHONENUMBER .....21

107 8.7 CLASS PERSONNAME.....22

108 **9 MANAGED OBJECT NAMING .....22**

109 **10 ASSOCIATION OF MANAGED OBJECTS .....22**

110 10.1 INTERFACE *ASSOCIATION*.....23

111 **11 CLASSIFICATION OF MANAGED OBJECTS .....25**

112 11.1 INTERFACE *CLASSIFICATIONNODE* CLASSIFICATIONNODE .....27

113 11.2 INTERFACE *CLASSIFICATION* .....28

114 11.2.1 *Context Sensitive Classification* .....28

115 11.3 EXAMPLE OF CLASSIFICATION SCHEMES [9].....30

116 **12 QUERYING OF MANAGED OBJECTS .....30**

117 12.1 OBJECT QUERY USE CASES .....30

118 12.1.1 *Browse and Drill Down Query* .....30

119 12.1.2 *Ad Hoc Queries Based on Object Metadata And Content*.....31

120 12.1.3 *Keyword Search Query*.....31

121 12.2 CLASSIFICATION BASED QUERY SUPPORT .....31

122 **13 INFORMATION MODEL: SECURITY VIEW .....31**

123 13.1 INTERFACE ACCESSCONTROLPOLICY.....32

124 13.2 INTERFACE PERMISSION.....33

125 13.3 INTERFACE PRIVILEGE .....33

126 13.4 INTERFACE PRIVILEGEATTRIBUTE.....34

127 13.5 INTERFACE SECURITYCLEARANCE .....34

128 13.6 INTERFACE ROLE .....34

129 13.7 INTERFACE GROUP.....34

130 13.8 INTERFACE IDENTITY.....34

131 13.9 INTERFACE PRINCIPAL.....35

132 **14 DISCLAIMER .....36**

133 **15 CONTACT INFORMATION.....37**

134 **COPYRIGHT STATEMENT.....38**

135 **Table of Figures**

136 Figure 1: Information Model Public View..... 9  
 137 Figure 2 ..... 12  
 138 Figure 3 ..... 12  
 139 Figure 4: Information Model Inheritance View..... 12  
 140 Figure 5: Example of Managed Object Association ..... 23  
 141 Figure 6: Example showing a Classification Tree ..... 26  
 142 Figure 7: Information Model Classification View..... 27  
 143 Figure 8: Classification Instance Diagram ..... 27  
 144 Figure 9: Context Sensitive Classification..... 29  
 145 Figure 10: Information Model: Security View..... 32

146 **Table of Tables**

147 Table 1: Pre-defined Classification Schemes ..... 30

148

149

## 149 **3 Introduction**

### 150 **3.1 Summary of Contents of Document**

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

152

153 A separate document, *ebXML Registry Services Specification*[5], describes how  
154 to build Registry Services that provide access to the information content in the  
155 ebXML Registry.

### 156 **3.2 Document Version History**

157 o Version 0.1: Initial version

158 o Version 0.2: Changes based on review. Simplified OASIS classification  
159 model. Eliminated ambiguity between managed object content (the content)  
160 and ManagedObject the metadata about the content. Changes to Object  
161 class hierarchy and attributes.

162 o Version 0.3: Changes based on review dated 9/29/2000. Changed “managed  
163 object” to “managed object content”. Made Package a ManagedObject. Made  
164 ManagedObject to Submission association a many to many association.  
165 Logged post-Tokyo issues raised.

166 o Version 0.41: Changes based on Tokyo face-to-face meeting. Added **context**  
167 **sensitive classifications, reformatted document to conform to ebXML**  
168 **document standard.**

169 o Version 0.5: Major re-write. Cleanup of base information model. Added  
170 IntrinsicObject and ExtrinsicObject. Changed ExternalObject to ExternalLink.  
171 Added security information model.

172 o Version 0.51: Incorporated the information model aspects of the security  
173 specification.

### 174 **3.3 General Conventions**

175 o UML diagrams are used as a way to concisely describe concepts. They are  
176 not intended to convey any specific implementation or methodology  
177 requirements.

178 o Interfaces are often used in UML diagrams. They are used instead of classes  
179 with attributes to provide an abstract definition without implying any specific  
180 implementation. Specifically they do not imply that objects in the registry will  
181 be access directly via these interfaces. Objects in the registry are accessed  
182 via registry services interface described in [5].

183 o The term “*managed object content*” is used to refer to actual registry content  
184 (e.g. a DTD, as opposed to metadata)

- 185 o The term “*ManagedObject*” is used to refer to an object that provides  
186 metadata about content instance (*managed object content*).

187

188 The information model *does not* contain *any* elements that are the actual content  
189 of the Registry (*managed object content*). All elements of the information model  
190 represent metadata about the content and not the content itself.

191

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

194

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

### 198 **3.4 Audience**

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

- 201 o Implementers of ebXML Registry Services  
202 o Implementers of ebXML Registry Clients

### 203 **3.5 Related Documents**

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

- 206 a) *ebXML Registry Business Domain Model* [4] - defines requirements for  
207 ebXML Registry Services  
208 b) *ebXML Registry Services Specification* [5] - defines the actual registry  
209 services based on this information model  
210 c) *Trading-Partner Specification* [8] (under development) - defines how  
211 profiles can be defined for a party and how two parties' profiles may be  
212 used to define a party agreement  
213 d) *ebXML Business Process Specification Schema* [6]  
214 e) **Core Components specification [7]** (??? called Mary Kay to get name)  
215

## 216 **4 Design Objectives**

### 217 **4.1 Goals**

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

- 219 o Communicate what information is in the Registry and how that information is  
220 organized  
221 o Leverage as much as possible the work done in the OASIS [2] and the ISO  
222 11179 [3] Registry models  
223 o Align with relevant works in progress within other ebXML working groups

- 224 o Be able to evolve to support future ebXML Registry requirements
- 225 o Be compatible with other ebXML specifications

## 226 **4.2 Caveats and Assumptions**

227 The Registry Information Model specification is a living document. It is the first in  
228 a series of phased deliverables. Later versions of the document will include  
229 additional functionality planned for current and future development.

## 230 **5 System Overview**

### 231 **5.1 Role of ebXML Registry**

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

### 237 **5.2 Registry Services**

238 A set of Registry Services that provide access to registry content to clients of the  
239 registry is defined in the ebXML Registry Services Specification [5]. This  
240 document does not provide details on these services but may occasionally refer  
241 to them.

### 242 **5.3 What the Registry Information Model Does**

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

247 The Registry information model:

- 248 o Defines what types of objects are stored in the Registry
- 249 o Defines how stored objects are organized in the Registry
- 250 o Is the metamodel for the Registry
- 251 o Is based on ebXML metamodels from various working groups

252

### 253 **5.4 How the Registry Information Model Works**

254 Implementers of the ebXML Registry may use the information model to  
255 determine which classes to include in their registry implementation and what  
256 attributes and methods these classes may have. They may also use it to  
257 determine what sort of database schema their registry implementation may need.

258 Note that the information model is meant to be illustrative and does not prescribe  
 259 any specific implementation choices.  
 260

261 **5.5 Where the Registry Information Model May Be Implemented**

262 The Registry Information Model may be implemented within an ebXML registry in  
 263 form of a relational database schema, object database schema or some other  
 264 physical schema. It may also be implemented as interfaces and classes within a  
 265 registry implementation.

266 **6 Registry Information Model: Public View**

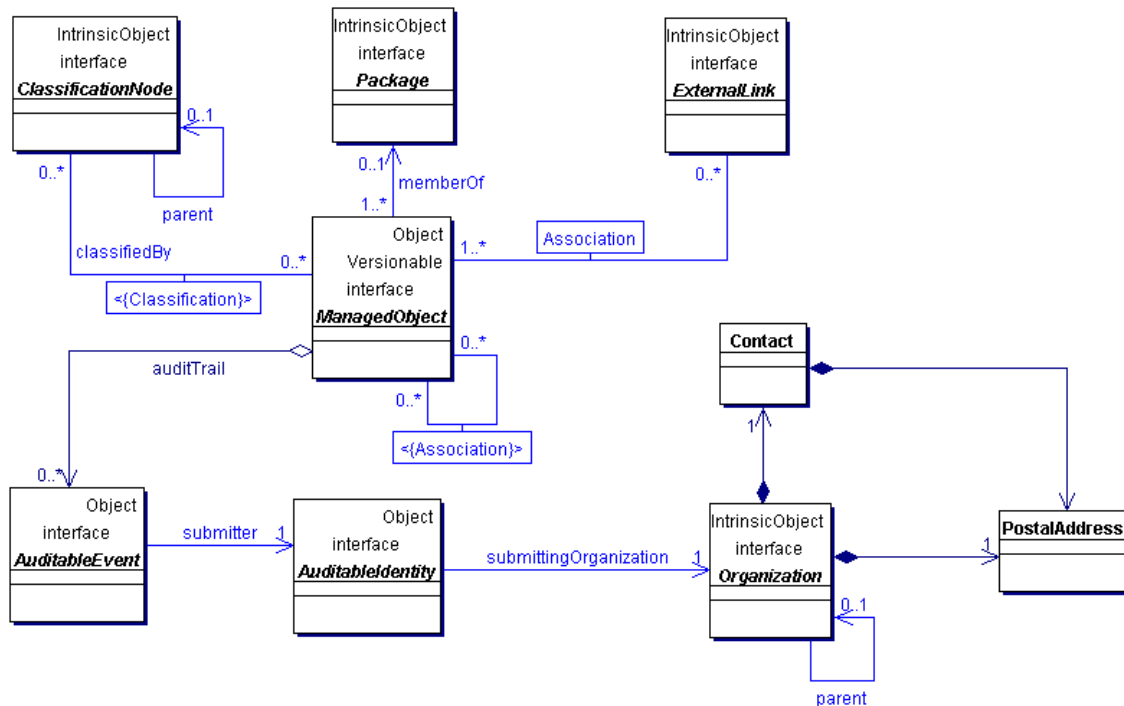
267 This chapter provides a high level public view of the most visible objects in the  
 268 registry.

269

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

273

274 The reader is again reminded that the information model is modeling metadata  
 275 and not actual content.  
 276



277

278

Figure 1: Information Model Public View

## 279 **6.1 ManagedObject**

280 The central object in the information model is a ManagedObject. An instance of  
281 ManagedObject exists for each content instance submitted to registry. Instances  
282 of ManagedObject class provide metadata about a managed object content in  
283 the registry. The actual managed object content (e.g. a DTD) is not contained in  
284 an instance of the ManagedObject class.

## 285 **6.2 Association**

286 Association instances are used to define many-to-many associations between  
287 objects in the information model. Associations are described in detail in chapter  
288 22.

## 289 **6.3 ExternalLink**

290 ExternalLink instances model a named URI to content that may reside outside  
291 the registry. ManagedObject may be associated with any number of  
292 ExternalLinks to annotate a ManagedObject with external links to external  
293 content.

294  
295 Consider the case where a Submitting Organization submits a managed object  
296 content (e.g. a DTD) and wants to associate some external content to that object  
297 (e.g. the Submitting Organization's home page). The ExternalLink enables this  
298 capability. A potential use of the ExternalLink capability may be in a GUI tool that  
299 displays the ExternalLinks to a ManagedObject. The user may click on such links  
300 and navigate to an external web page referenced by the link.

## 301 **6.4 ClassificationNode**

302 ClassificationNode instances are used to define tree structures where each node  
303 in the tree is a ClassificationNode. Classification trees constructed with  
304 ClassificationNodes are used to define classification schemes or ontologies.  
305 ClassificationNode is described in detail in chapter 25.

## 306 **6.5 Classification**

307 Classification instances are used to classify managed object content by  
308 associating their ManagedObject instance with a ClassificationNode within a  
309 classification scheme. Classification is described in detail in chapter 25.

## 310 **6.6 Package**

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

## 314 **6.7 AuditableEvent**

315 AuditableEvent is used to provide an audit trail for ManagedObjects

## 316 **6.8 PostalAddress**

317 PostalAddress is a simple reusable entity class that defines attributes of a postal  
318 Address.

## 319 **6.9 Contact**

320 Contact is a simple reusable entity class that defines attributes of a contact  
321 person.  
322

## 323 **6.10 Organization**

324 Organization instances provide information on organizations such as a  
325 Submitting Organization. Each Organization instance may have a reference to a  
326 parent Organization.

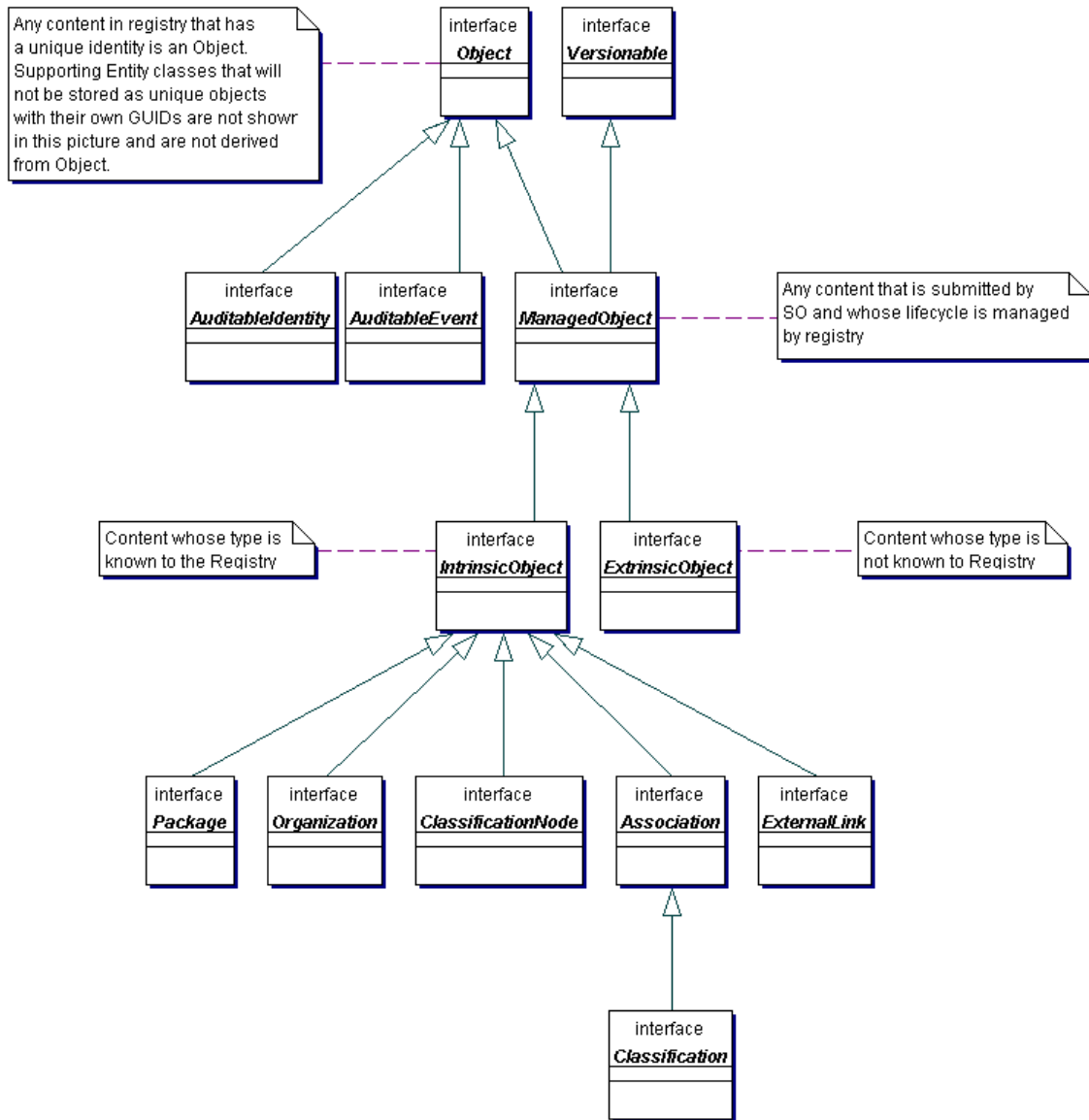
## 327 **7 Registry Information Model: Detail View**

328 This chapter covers the information model classes in more detail than the Public  
329 View. The detail view introduces some additional classes within the model that  
330 were not described in the public view of the information model.

331  
332 Figure 4 shows the inheritance or “is a” relationships between the classes in the  
333 information model. Note that it does not show the relationships since they have  
334 already been shown in Figure 1. Class attributes and class methods are also not  
335 shown in Figure 2. Detailed description of methods and attributes of most  
336 interfaces and classes will be displayed in tabular form following the description  
337 of each class in the model.

338  
339 The interface Association will be covered in detail separately in chapter 10. The  
340 interfaces Classification and ClassificationNode will be covered in detail  
341 separately in chapter 11.

342  
343 The reader is again reminded that the information model is modeling metadata  
344 and not actual content.



345

346

Figure 2

347

Figure 3

348

349

Figure 4: Information Model Inheritance View

350

### 351 **7.1 Interface *Object***

#### 352 **All Known Subinterfaces:**

- 353 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),
- 354 [ExtrinsicObject](#), [IntrinsicObject](#), [ManagedObject](#), [Organization](#), [Package](#),
- 355 [Submission](#)

356  
 357 Object provides a common base interface for almost all objects in the information  
 358 model. Information model classes whose instances have a unique identity and an  
 359 independent life cycle are descendents of the Object class.

360  
 361 Note that Contact and Address are not descendants of the Object class because  
 362 their instances do not have an independent existence and unique identity. They  
 363 are always a part of some other class's instance (e.g. Organization has an  
 364 Address).

365

Method Summary	
String	<a href="#">getDescription()</a> Gets the context independent textual description for this object.
String	<a href="#">getGUID()</a> Gets the globally unique ID for this object.
void	<a href="#">setDescription(String description)</a> Sets the context independent textual description for this object.
void	<a href="#">setGUID(String GUID)</a> Sets the globally unique ID for this object.

366

367 **7.2 Interface *Versionable***

368 **All Known Subinterfaces:**

369 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),  
 370 [ExtrinsicObject](#), [IntrinsicObject](#), [ManagedObject](#), [Organization](#), [Package](#)

371

372 The Versionable interface defines the behavior common to classes that are  
 373 capable of creating versions of their instances. At present all ManagedObject  
 374 classes are required to implement the Versionable interface.

375

Method Summary	
int	<a href="#">getMajorVersion()</a> Gets the major revision number for this version of the Versionable object.
int	<a href="#">getMinorVersion()</a> Gets the minor revision number for this version of the Versionable object.

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.

376

377 **7.3 Interface *ManagedObject***

378 **All Superinterfaces:**

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

380 **All Known Subinterfaces:**

381 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#),  
382 [ExtrinsicObject](#), [IntrinsicObject](#), [Organization](#), [Package](#)

383

384 The ManagedObject class models a common base class for all submitted content  
385 whose life cycle is managed by the registry. Content submitted to the registry is  
386 further specialized by ExtrinsicObject and IntrinsicObject sub-classes of  
387 ManagedObject.

388

<b>Method Summary</b>	
Collection	<a href="#">getAssociations</a> ()
Collection	<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 ChangeLog objects.
Collection	<a href="#">getClassifications</a> () Returns the Classifications associated with this object.
Collection	<a href="#">getExternalLinks</a> () Returns the ExternalLinks associated with this object.
String	<a href="#">getName</a> () Gets user friendly context independent name of object in repository.
<a href="#">Package</a>	<a href="#">getPackage</a> () Returns the Package associated with this object.
int	<a href="#">getStatus</a> () Gets the life cycle status of the ManagedObject within the registry.
void	<a href="#">setName</a> (String name) Sets user friendly context independent name of object in

	repository.
void	<a href="#">setStatus</a> (int status) Sets the life cycle status of the ManagedObject within the registry.

389

<b>Methods inherited from interface Object</b>	
<a href="#">getDescription</a> , <a href="#">getGUID</a> , <a href="#">setDescription</a> , <a href="#">setGUID</a>	

390

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

391

## 392 7.4 Interface *ExtrinsicObject* ExtrinsicObject

393 All Superinterfaces:

394 [ManagedObject](#), [Object](#), [Versionable](#)

395

396 ExtrinsicObject models all submitted content whose type is not intrinsically known  
397 to the registry.

398

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

402

403 Examples of content modeled by ExtrinsicObject include party profiles, business  
404 process descriptions, schemas, etc.

405

<b>Method Summary</b>	
String	<a href="#">getMimeType</a> () Gets the mime type associated with the ManagedObject.
int	<a href="#">getObjectType</a> () Gets the pre-defined object type that best describes the ManagedObject.
URI	<a href="#">getURI</a> () Gets the URI to the actual object in repository
boolean	<a href="#">isOpaque</a> () Determines whether the ExtrinsicObject is opaque (not readable) by the registry. In some situations, a Submitting Organization may submit content that is encrypted and not even readable by the registry. This attribute allows the registry to know whether this is the case.

void	<a href="#">setMimeType</a> (String mimeType) Sets the mime type associated with the ManagedObject.
void	<a href="#">setObjectType</a> (int type) Sets the pre-defined object type that best describes the ManagedObject.
void	<a href="#">setOpaque</a> (boolean isOpaque) Sets whether the ExtrinsicObject is opaque (not readable) by the registry.
void	<a href="#">setURI</a> (URI uri) Sets the URI to the actual object in repository.

406

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

409 **7.4.1 Pre-Defined Extrinsic Object Types**

410 The following table lists pre-defined types of ExtrinsicObjects.

411

<b>Field Summary</b>	
static int	<a href="#">OBJECT_TYPE_PARTY_AGREEMENT</a> These are XML documents that represent a technical agreement between two parties on how they plan to communicate with each other in a specific B2B protocol.
static int	<a href="#">OBJECT_TYPE_PARTY_PROFILE</a> These are XML documents that provide information about a party interested in participating in B2B interaction.
static int	<a href="#">OBJECT_TYPE_PROCESS</a> A process description document.
static int	<a href="#">OBJECT_TYPE_ROLE</a> An XML description of a Role in a Collaboration Party Profile
static int	<a href="#">OBJECT_TYPE_SERVICE_INTERFACE</a> An XML description of a service interface
static int	<a href="#">OBJECT_TYPE_SOFTWARE_COMPONENT</a> A software component (e.g. EJB, class library etc.)
static int	<a href="#">OBJECT_TYPE_TRANSPORT</a> An XML description of a transport configuration Collaboration Party Profile
static int	<a href="#">OBJECT_TYPE_UML_MODEL</a> A UML Model
static int	<a href="#">OBJECT_TYPE_UNKNOWN</a> An unspecified objectType.
static int	<a href="#">OBJECT_TYPE_XML_SCHEMA</a>

	A schema (DTD, XML Schema, Relax, etc.) for an XML document
--	---

412

413 **7.5 Interface *IntrinsicObject* IntrinsicObject**

414 **All Superinterfaces:**

415 [ManagedObject](#), [Object](#), [Versionable](#)

416 **All Known Subinterfaces:**

417 [Association](#), [Classification](#), [ClassificationNode](#), [ExternalLink](#), [Organization](#),  
418 [Package](#)

419  
420 IntrinsicObject provides a common base class for modeling all submitted content  
421 whose type is known to the Registry. In fact, these types are defined by the  
422 ebXML registry specifications.

423  
424 This interface currently does not define any attributes or methods. Note that  
425 methods inherited from the base interfaces of this interface are not shown.  
426

427 **7.6 Interface *ExternalLink* ExternalLink**

428 **All Superinterfaces:**

429 [IntrinsicObject](#), [ManagedObject](#), [Object](#), [Versionable](#)

430  
431 ExternalLink instances model a named URI to content that may reside outside  
432 the registry.

433  
434 Consider the case where a Submitting Organization submits a managed object  
435 content (e.g. a DTD) and wants to associate some external content to that object  
436 (e.g. the Submitting Organization's home page). The ExternalLink enables this  
437 capability.

438  
439  
440

Method Summary	
URI	<a href="#">getURI()</a> Gets URI to the external content.
void	<a href="#">setURI</a> (URI uri) Sets URI to the external content.

441  
442 Note that methods inherited from the base interfaces of this interface are not  
443 shown.  
444

445 **8 Registry Audit Trail**

446 This chapter describes the information model elements that support the audit trail  
 447 capability of the registry. There are several classes in this chapter that are entity  
 448 classes that are used as wrappers to model a set of related attributes. These  
 449 entity classes do not have any associated behavior. They are analogous to the  
 450 “struct” construct in the C programming language.

451  
 452 The `getAuditTrail()` method of a `ManagedObject` returns an ordered `Collection` of  
 453 `AuditableEvents`. These `AuditableEvent` constitute the audit trail for the  
 454 `ManagedObject`. `AuditableEvents` include a timestamp for the event. Each  
 455 `AuditableEvent` has an `AuditableIdentity` that identifies the specific user that  
 456 performed an action that resulted in an `AuditableEvent`. Each `AuditableIdentity`  
 457 has an `Organization` which is usually the submitting `Organization`.

458 **8.1 Interface AuditableEvent**

459 **All Superinterfaces:**

460 [Object](#)

---

461  
 462 `AuditableEvent` instances provide a long-term record of events that effect a  
 463 change of state in a `ManagedObject`. A `ManagedObject` is associated with an  
 464 ordered `Collection` of `AuditableEvent` instances that provide a complete audit trail  
 465 for that `Object`.

466  
 467 `AuditableEvents` are usually a result of a client-initiated request. `AuditableEvent`  
 468 instances are generated by the registry service to log such events.

469  
 470 Often such events effect a change in the life cycle of a `ManagedObject`. For  
 471 example a client request could `Create`, `Update`, `Deprecate` or `Delete` a  
 472 `ManagedObject`. No `AuditableEvent` is created for requests that do not alter the  
 473 state of a `ManagedObject`. Specifically, read-only requests do not generate an  
 474 `AuditableEvent`. No `AuditableEvent` is generated for a `ManagedObject` when it is  
 475 classified, assigned to a `Package` or associated with another `Object`.

476 **See Also:**

478

Field Summary	
static int	<a href="#">EVENT_TYPE_CREATED</a> An event that created a <code>ManagedObject</code>
static int	<a href="#">EVENT_TYPE_DELETED</a> An event that deleted a <code>ManagedObject</code>
static int	<a href="#">EVENT_TYPE_DEPRECATED</a> An event that deprecated a <code>ManagedObject</code>

static int	<a href="#">EVENT_TYPE_UPDATED</a> An event that updated the state of a ManagedObject
static int	<a href="#">EVENT_TYPE_VERSIONED</a> An event that versioned a ManagedObject

479

Method Summary	
<a href="#">AuditableIdentity</a>	<a href="#">getAuditableIdentity()</a> Gets the AuditableIdentity that sent the request that effected this event.
int	<a href="#">getEventType()</a> The type of this event
Timestamp	<a href="#">getTimestamp()</a> Gets the Timestamp for when this event occurred.

480

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

## 483 8.2 Interface AuditableIdentity

484 All Superinterfaces:

485 [Object](#)

486

487 AuditableIdentity instances are used in an AuditableEvents to keep track of the identity  
488 of the requestor that sent the request that generated the AuditableEvent.

489

Method Summary	
String	<a href="#">getName()</a> Get the name of this AuditableIdentity.
<a href="#">Organization</a>	<a href="#">getOrganization()</a> Gets the Submitting Organization that sent the request that effected this change.

490

## 491 8.3 Interface Organization

492 All Superinterfaces:

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

494

495 Organization instances provide information on organizations such as a  
496 Submitting Organization. Each Organization instance may have a reference to a

497 parent Organization. In addition it may have a contact attribute defining the  
 498 primary contact within the organization. An Organization also has an address  
 499 attribute.

500 **See Also:**

501

<b>Method Summary</b>	
<a href="#">PostalAddress</a>	<a href="#">getAddress()</a> Gets the Address for this Organization.
<a href="#">Contact</a>	<a href="#">getContact()</a> Gets the primary Contact for this Organization.
String	<a href="#">getFax()</a> Gets the FAX number for this Organization.
<a href="#">Organization</a>	<a href="#">getParent()</a> Gets the parent Organization for this Organization.
String	<a href="#">getTelephone()</a> Gets the main telephone number for this Organization.

502

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

505

## 506 8.4 Class Contact

507

508

509 Contact is a simple reusable entity class that defines attributes of a contact  
 510 person.

511

<b>Field Summary</b>	
<a href="#">PostalAddress</a>	<a href="#">address</a> The postal address for this Contact.
String	<a href="#">email</a> The email address for this Contact.
<a href="#">TelephoneNumber</a>	<a href="#">fax</a> The FAX number for this Contact.
<a href="#">TelephoneNumber</a>	<a href="#">mobilePhone</a> The mobile telephone number for this Contact.
<a href="#">PersonName</a>	<a href="#">name</a> Name of contact person
<a href="#">TelephoneNumber</a>	<a href="#">pager</a> The pager telephone number for this Contact.
<a href="#">TelephoneNumber</a>	<a href="#">telephone</a> The default (land line) telephone number for this

	Contact.
URL	<a href="#">url</a> The URL to the web page for this contact.

512

513 **8.5 Class PostalAddress**

514

515

516 Address is a simple reusable entity class that defines attributes of a postal  
517 Address.

518

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

519

520 **8.6 Class TelephoneNumber**

521

522

523

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

525

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

526

527 **8.7 Class PersonName**

528

529 A simple entity class for a person's name.

530

531

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

532

533 **9 Managed Object Naming**

534 A *managed object content* has a name that may or may not be unique within the  
535 Registry.

536

537 In addition a *managed object content* may have any number of context sensitive  
538 alternate names that are valid only in the context of a particular classification  
539 scheme. Alternate contextual naming will be addressed in a later version of the  
540 Registry Information Model.

541

542 **10 Association of Managed Objects**

543 A managed object content may be associated with 0 or more objects. The  
544 information model defines an Association class. An instance of the Association  
545 class represents an association between a ManagedObject and another Object.  
546 An example of such an association is between a PartyProfile and a  
547 PartyAgreement associated with that PartyProfile as shown in Figure 5.

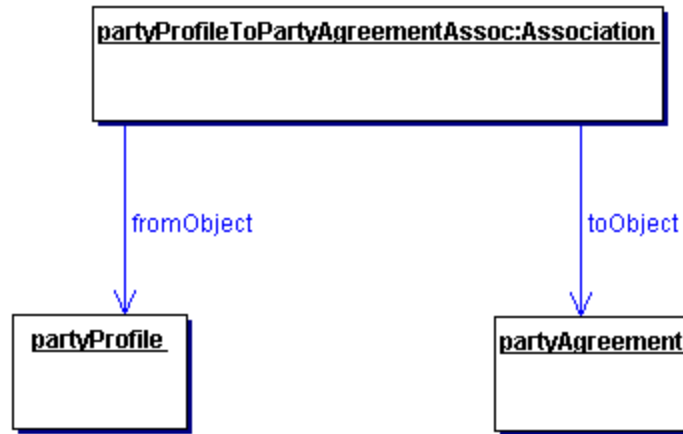


Figure 5: Example of Managed Object Association

548  
549  
550

### 10.1 Interface Association

**All Superinterfaces:**

[IntrinsicObject](#), [ManagedObject](#), [Object](#), [Versionable](#)

**All Known Subinterfaces:**

[Classification](#)

551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562

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

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

Field Summary	
static int	<a href="#">ASSOCIATION_TYPE_CLASSIFIED_BY</a> Defines that the source object is classified by the target object.
static int	<a href="#">ASSOCIATION_TYPE_CONTAINED_BY</a> Defines that source object is contained by the target object.
static int	<a href="#">ASSOCIATION_TYPE_CONTAINS</a> Defines that source object contains the target object.
static int	<a href="#">ASSOCIATION_TYPE_EXTENDS</a> Defines that source object inherits from or specializes the target object.
static int	<a href="#">ASSOCIATION_TYPE_IMPLEMENTES</a> Defines that source object implements the behaviour defined by the target object.
static int	<a href="#">ASSOCIATION_TYPE_INSTANCE_OF</a>

	Defines that source object is an instance of target object
static int	<a href="#">ASSOCIATION_TYPE_RELATED_TO</a> Defines that source object is an instance of target object.
static int	<a href="#">ASSOCIATION_TYPE_SUPERSEDED_BY</a> Defines that the source object is superseded by the target object.
static int	<a href="#">ASSOCIATION_TYPE_SUPERSEDES</a> Defines that the source object supersedes the target object.
static int	<a href="#">ASSOCIATION_TYPE_USED_BY</a> Defines that the source object is used by the target object in some manner.
static int	<a href="#">ASSOCIATION_TYPE_USES</a> Defines that the source object uses the target object in some manner.

563

<b>Method Summary</b>	
int	<a href="#">getAssociationType()</a> Gets the predefined association type for this Association.
<a href="#">Object</a>	<a href="#">getSourceObject()</a> Gets the Object that is the source of this Association.
String	<a href="#">getSourceRole()</a> Gets the name of the role played by the source Object in this Association.
<a href="#">Object</a>	<a href="#">getTargetObject()</a> Gets the Object that is the target of this Association.
String	<a href="#">getTargetRole()</a> Gets the name of the role played by the target Object in this Association.
boolean	<a href="#">isBidirectional()</a> Determine whether this Association is bi-directional.
void	<a href="#">setAssociationType(int associationType)</a> Sets the predefined association type for this Association.
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.

Association.
--------------

564

## 567 **11 Classification of Managed Objects**

568 This section describes the how the information model supports classification of  
569 managed object contents. It is a simplified version of the OASIS classification  
570 model [2].

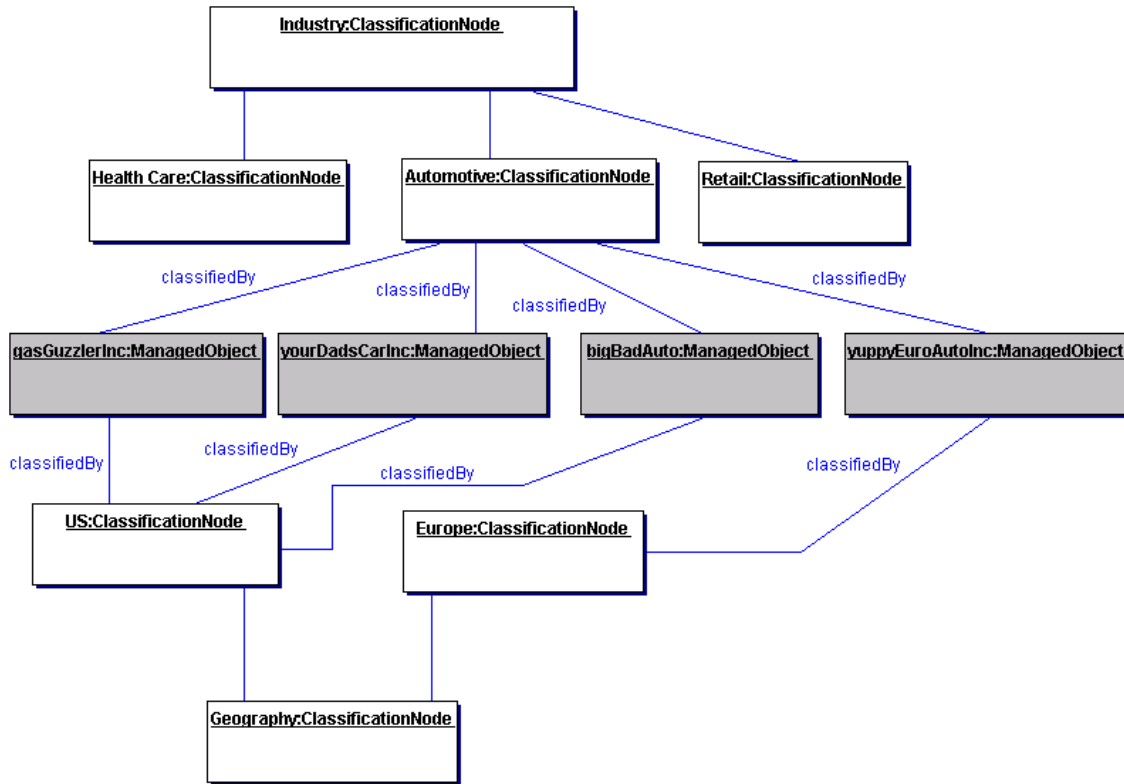
571 A managed object content may be classified in many ways. For example the  
572 same Party managed object content may be classified by the industry it is in, by  
573 the products it sells and by the geographical location it is in.

574  
575 A general classification scheme can be viewed as a classification tree. In the  
576 example shown below in Figure 4, ManagedObjects representing PartyProfiles  
577 are shown as shaded boxes. Each PartyProfile represents an automobile  
578 manufacturer. Each PartyProfile is classified by the ClassificationNode named  
579 Automotive under the root ClassificationNode named Industry. Furthermore, the  
580 US Automobile manufacturers are classified by the US ClassificationNode under  
581 the Geography ClassificationNode. Similarly, a European automobile  
582 manufacturer is classified by the Europe ClassificationNode under the  
583 Geography ClassificationNode.

584  
585 The example shows how a ManagedObject may be classified by multiple  
586 classification schemes. A classification scheme is defined by a  
587 ClassificationNode that is the root of a classification tree (e.g. Industry,  
588 Geography).

---

<sup>1</sup> Superset of OASIS. Some examples are borrowed from OASIS.



589  
590

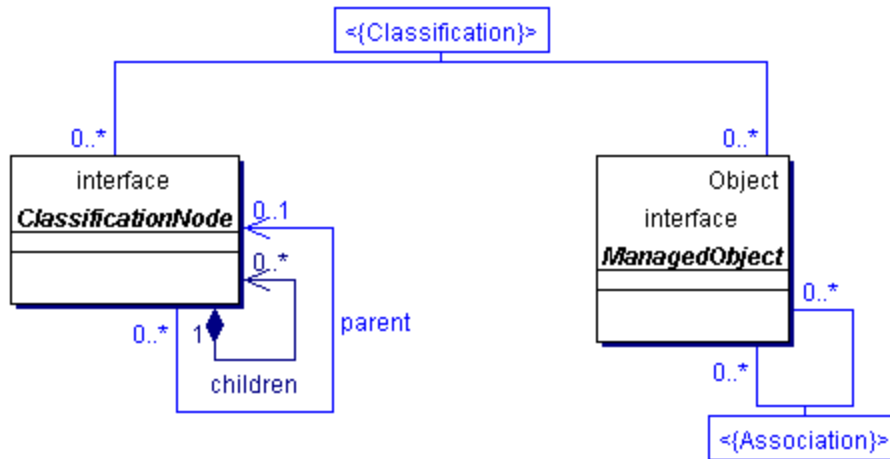
**Figure 6: Example showing a Classification Tree**

591  
592  
593  
594  
595  
596  
597

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

598  
599  
600

In order to support a general classification scheme that can support single level as well as multi-level classifications, the information model defines the classes and relationships shown in Figure 7.



601

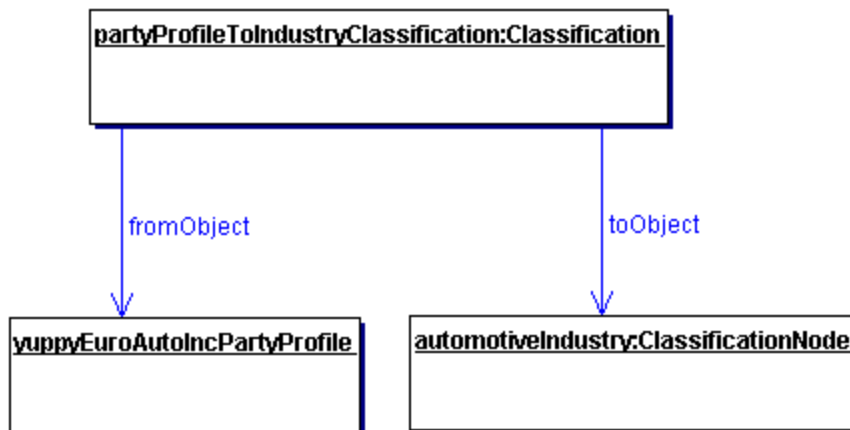
602

**Figure 7: Information Model Classification View**

603

A Classification is a specialized form of an Association. Figure 8 shows an example of a Party object that is classified by the Industry that it belongs to.

604



605

606

**Figure 8: Classification Instance Diagram**

607

## 11.1 Interface *ClassificationNode* ClassificationNode

608

**All Superinterfaces:**

609

[IntrinsicObject](#), [ManagedObject](#), [Object](#), [Versionable](#)

610

611

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

612

613

614

**See Also:**

615

[Classification](#)

616

## Method Summary

<a href="#">ClassificationNode</a>	<a href="#">getParent</a> ( ) Gets the parent ClassificationNode for this ClassificationNode.
void	<a href="#">setParent</a> ( <a href="#">ClassificationNode</a> parent) Sets the parent ClassificationNode for this ClassificationNode.

617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627

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

In Figure 6, several instances of ClassificationNode are defined (all light colored boxes). A ClassificationNode has zero or 1 ClassificationNode 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.

## 11.2 Interface *Classification*

### All Superinterfaces:

[Association](#), [IntrinsicObject](#), [ManagedObject](#), [Object](#), [Versionable](#)

631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649

---

Classification instances are used to classify managed object content by associating their ManagedObject instance with a ClassificationNode instance within a classification scheme.

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

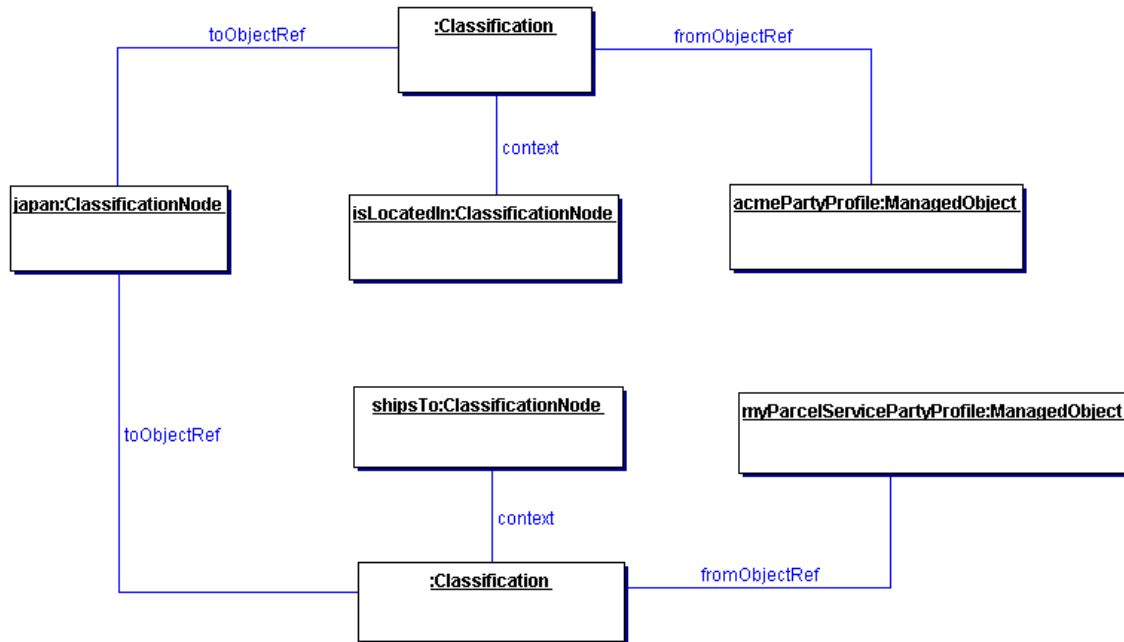
Classification is a specialized form of Association from a ManagedObject to a specific ClassificationNode in the classification tree. The information model defines a Classification class as a sub-class of Association class to allow for future specialization as well as to make classification notion be obvious in the model. This also allows for more efficient implementation by providing a different extent for Classifications than from Associations.

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

### 11.2.1 Context Sensitive Classification

Consider the case depicted in Figure 9 where a PartyProfile for ACME Inc. is classified by the Japan ClassificationNode under the Geography classification

653 scheme. In the absence of the context for this classification its meaning is  
 654 ambiguous. Does it mean that ACME is located in Japan, or does it mean that  
 655 ACME ships products to Japan, or does it have some other meaning? To  
 656 address this ambiguity a Classification may optionally be associated with another  
 657 ClassificationNode (in this example named isLocatedIn) that provides the missing  
 658 context for the Classification. Another PartyProfile for MyParcelService may be  
 659 classified by the Japan ClassificationNode where this Classification is associated  
 660 with a different ClassificationNode (e.g. named shipsTo) to indicate a different  
 661 context than the one used by ACME Inc.



662  
 663

**Figure 9: Context Sensitive Classification**

664 A more complex case is where a Classification may be associated with multiple  
 665 contexts. In order to support the possibility of Classification within multiple  
 666 contexts, a Classification is itself classified by another Classification that binds  
 667 the first Classification to a Classification node that provides the missing context.  
 668

669 In summary, the generalized support for classification schemes in the information  
 670 model allows:

- 671 o A ManagedObject to be classified by defining a Classification that associates
- 672 it with a ClassificationNode in a classification tree
- 673 o A ManagedObject to be classified along multiple facets by having multiple
- 674 classifications that associate it with multiple ClassificationNodes.
- 675 o A classification defined for a ManagedObject to be qualified by the context in
- 676 which it is being classified

677 **11.3 Example of Classification Schemes [9]**

678 The following table lists some examples of possible classification schemes  
 679 enabled by the information model. These schemes are based on a subset of  
 680 contextual concepts identified by BP and CC in [9]. This list is meant to be  
 681 illustrative not prescriptive.

682  
 683

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 a region
Temporal	Find Supplier that can ship with 24 hours
Role	Find All Suppliers
Package	Find all DTDs in the RosettaNet DTD package

684

**Table 1: Pre-defined Classification Schemes**

685 **12 Querying of Managed Objects**

686 This chapter describes how the information model supports the querying of  
 687 managed object contents based on the attributes, content, associations and  
 688 classifications of managed object contents. Details of the access protocol  
 689 between clients and the Registry for the purpose of object querying are described  
 690 in [5]. This chapter defines at a high level the query mechanism without defining  
 691 the actual query protocol and messages exchanged as part of that protocol.

692 **12.1 Object Query Use Cases**

693 It is recognized that there are several different use cases defining how a client  
 694 may want to query and search the Registry for managed object contents.

695 **12.1.1 Browse and Drill Down Query**

696 This is expected to be the primary use case for querying the Registry. In this  
 697 scenario a user browses the registry content using a GUI tool referred to as the  
 698 Registry Browser. The user expects to initially browse the content based on the  
 699 pre-defined classification schemes defined in section 11.3. The user may also  
 700 use additional classification schemes that may have been defined for objects  
 701 selected by the pre-defined classification scheme chosen. The user will select a  
 702 managed object content and drill down to view the details of the object.  
 703 Such browse and drill down query support is defined in section 12.2.

### 704 **12.1.2 Ad Hoc Queries Based on Object Metadata And Content**

705 This is an advanced form of use case for querying the Registry. In this scenario a  
706 client program may search for managed object contents based on the metadata  
707 defined as attributes in its corresponding ManagedObject as well as the  
708 managed object content itself.

709

### 710 **12.1.3 Keyword Search Query**

711 In this scenario a user may search for managed object contents by specifying  
712 keywords that may be used to identify the managed object contents.

## 713 **12.2 Classification Based Query Support**

714 **CLASSIFICATION BASED QUERY SUPPORT IS REQUIRED FOR AN EBXML**  
715 **REGISTRY IMPLEMENTATION. AN IMPLEMENTATION MUST SUPPORT**  
716 **THE FOLLOWING CAPABILITIES FOR CLASSIFICATION BASED QUERY**  
717 **SUPPORT. DETAILS OF THE ACTUAL PROTOCOL AND MESSAGES**  
718 **EXCHANGED ARE DESCRIBED IN [5].**

- 719 1. Receive a query which requests a list of root ClassificationNodes  
720 (classification schemes) defined and return the ClassificationNodes that  
721 have no parent ClassificationNode.
- 722 2. Receive a query in which a ClassificationNode is identified and return the  
723 classification tree under that node in the tree.
- 724 3. Receive a query in which a ClassificationNode is identified and return the  
725 managed object contents classified by that ClassificationNode.

## 726 **13 Information Model: Security View**

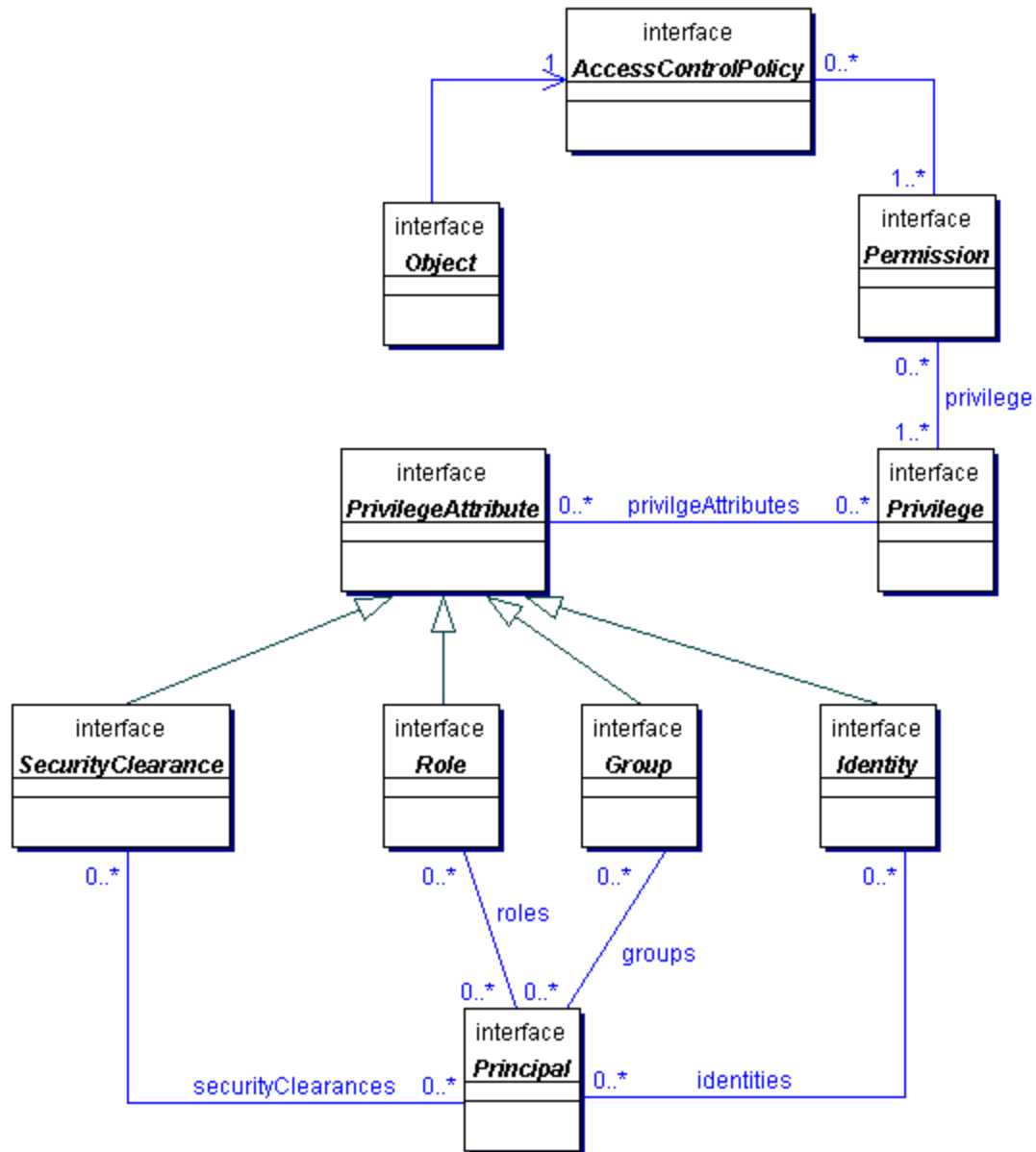
727 This chapter describes the aspects of the information model that relate to the  
728 security features of the registry.

729

730 shows the view of the objects in the Registry from a security perspective. It is  
731 showing object relationships as a UML class diagram. It is not showing class  
732 attributes or class methods that will be described in subsequent sections.

733

734



735  
736

Figure 10: Information Model: Security View

737 **13.1 Interface AccessControlPolicy**

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

741  
742

**Method Summary**

Collection	<a href="#">getPermissions()</a> Gets the Permissions defined for this AccessControlPolicy
------------	---

743

744 **13.2 Interface Permission**

745

746 The Permission object is used for authorization and access control to Objects in  
747 the registry. The Permissions for an Object are defined in an  
748 AccessControlPolicy object.

749

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

752

**See Also:**

753

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

754

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

755

756 **13.3 Interface Privilege**

757

758 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute  
759 can be a SecurityClearance, a Group, a Role, or an Identity.

760

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

765

766 This mechanism enables the flexibility to have object access control policies that  
767 are based on any combination of Roles, Identities, Groups or a  
768 SecurityClearances.

769

**See Also:**

770

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

771

Method Summary	
Collection	<a href="#">getPrivilegeAttributes()</a>

	Gets the PrivilegeAttributes associated with this Privilege.
--	--

772

773 **13.4 Interface PrivilegeAttribute**

774 **All Known Subinterfaces:**

775 [Group](#), [Identity](#), [Role](#), [SecurityClearance](#)

776

---

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

781 **See Also:**

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

783 **13.5 Interface SecurityClearance**

784 **All Superinterfaces:**

785 [PrivilegeAttribute](#)

786

---

787 A SecurityClearance is PrivilegeAttribute that describes a security clearance.

788 **13.6 Interface Role**

789 **All Superinterfaces:**

790 [PrivilegeAttribute](#)

791

---

792 A security Role PrivilegeAttribute. For example a hospital may have Roles such  
 793 as Nurse, Doctor, Administrator etc. Roles are used to grant Privileges to  
 794 Principals. For example a Doctor may be allowed to write a prescription but a  
 795 Nurse may not.

796 **13.7 Interface Group**

797 **All Superinterfaces:**

798 [PrivilegeAttribute](#)

799

---

800 A security Group PrivilegeAttribute. A Group is an aggregation of users that may  
 801 have different roles. For example a hospital may have a Group defined for  
 802 Nurses and Doctors that are participating in a specific clinical trial (e.g  
 803 AspirinTrial). Groups are used to grant Privileges to Principals. For example the  
 804 members of the AspirinTrial group may be allowed to write a prescription for  
 805 Aspirin (even though Nurses as a rule may not be allowed to write prescriptions).

806 **13.8 Interface Identity**

807 **All Superinterfaces:**

808 [PrivilegeAttribute](#)

809

810 A security Identity PrivilegeAttribute. This is typically used to identify a person.  
811 Identity attribute may be in form of a digital certificate.

812 **13.9 Interface Principal**

813

814 Principal is a completely generic term used by the security community to include  
815 both people and software systems. The Principal object is an entity, which has a  
816 set of PrivilegeAttributes. These PrivilegeAttributes include at least one identity,  
817 and optionally a set of role memberships, group memberships or security  
818 clearances. A principal is used to authenticate a requestor and to authorize the  
819 requested action based on the PrivilegeAttributes associated with the Principal.

820 **See Also:**

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

822

Method Summary	
Collection	<a href="#">getGroups()</a> Gets the Groups associated with this Principal.
Collection	<a href="#">getIdentities()</a> Gets the Identities associated with this Principal.
Collection	<a href="#">getRoles()</a> Gets the Roles associated with this Principal.
Collection	<a href="#">getSecurityClearances()</a> Gets the SecurityClearances associated with this Principal.

823

827

## 827 References

- 828 [1] ebXML Glossary,  
829 [2] OASIS Information Model  
830 <http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html>  
831 [3] ISO 11179 Information Model  
832 [http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba78525662100](http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument)  
833 [5419d7/b83fc7816a6064c68525690e0065f913?OpenDocument](http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument)  
834 [4] Registry and Repository: Business Domain Model  
835 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>  
836 [5] ebXML Registry Services Specification  
837 [6] ebXML Business Process Metamodel Specification Schema  
838 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>  
839 [7] ebXML Core Components Metamodel (unable to get these so far)  
840 [8] Trading-Partner Specification  
841 [http://www.ebxml.org/project\\_teams/trade\\_partner/private/](http://www.ebxml.org/project_teams/trade_partner/private/)  
842 [9] Context table informal document from Core Components  
843 [http://www.ebxml.org/project\\_teams/core\\_components/ContextTable.doc](http://www.ebxml.org/project_teams/core_components/ContextTable.doc)  
844  
845

846 **14 Disclaimer**

847 The views and specification expressed in this document are those of the authors  
848 and are not necessarily those of their employers. The authors and their  
849 employers specifically disclaim responsibility for any problems arising from  
850 correct or incorrect implementation or use of this design.  
851

851 **15 Contact Information**

852

853 Team Leader

854 Name: Scott Nieman  
 855 Company: Norstan Consulting  
 856 Street: 5101 Shady Oak Road  
 857 City, State, Postal Code: Minnetonka, MN 55343  
 858 Country: USA  
 859 Phone: 952.352.5889  
 860 Email: Scott.Nieman@Norstan

861

862 Vice Team Lead

863 Name: Yutaka Yoshida  
 864 Company: Sun Microsystems  
 865 Street: 901 San Antonio Road, MS UMPK17-102  
 866 City, State, Postal Code: Palo Alto, CA 94303  
 867 Country: USA  
 868 Phone: 650.786.5488  
 869 Email: Yutaka.Yoshida@eng.sun.com

870

871 Editor

872 Name: Farrukh S. Najmi  
 873 Company: Sun Microsystems  
 874 Street: 1 Network Dr., MS BUR02-302  
 875 City, State, Postal Code: Burlington, MA, 01803-0902  
 876 Country: USA  
 877 Phone: 781.442.0703  
 878 Email: najmi@east.sun.com

879

880

**880 Copyright Statement**

881 Copyright © ebXML 2000. All Rights Reserved.

882

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

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

894

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

897

898 This document and the information contained herein is provided on an  
899 "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR  
900 IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE  
901 USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR  
902 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A  
903 PARTICULAR PURPOSE.