



Creating A Single Global Electronic Market

1  
2  
3  
4

5 **OASIS/ebXML Registry Information Model v1.1**  
6 **DRAFT**

7 **OASIS/ebXML Registry Technical Committee**

8 **30 August 2001**

9

10 **1 Status of this Document**

11  
12 Distribution of this document is unlimited.

13  
14 ***This version:***

15 <http://www.oasis-open.org/committees/regrep/documents/rimV1-1.pdf>

16  
17 ***Latest version:***

18 <http://www.oasis-open.org/committees/regrep/documents/rimV1-1.pdf>

19  
20  
21

## 21 **2 OASIS/ebXML Registry Technical Committee**

22 This document, in its current form, is a draft working document of the OASIS  
23 ebXML Registry Technical Committee. It build upon version 1.0 which was  
24 approved by the OASIS/ebXML Registry Technical Committee as DRAFT  
25 Specification of the TC. At the time of that approval the following were members  
26 of the OASIS/ebXML Registry Technical Committee.

27  
28 Nagwa Abdelghfour, Sun Microsystems  
29 Nicholas Berry, Boeing  
30 Kathryn Breining, Boeing  
31 Lisa Carnahan, US NIST (TC Chair)  
32 Dan Chang, IBM  
33 Joseph M. Chiusano, LMI  
34 Joe Dalman, Tie Commerce  
35 Suresh Damodaran, Sterling Commerce  
36 Vadim Draluk, BEA  
37 John Evdemon, Vitria Technologies  
38 Anne Fischer, Drummond Group  
39 Sally Fuger, AIAG  
40 Len Gallagher, NIST  
41 Michael Joya, XMLGlobal  
42 Una Kearns, Documentum  
43 Kyu-Chul Lee, Chungnam National University  
44 Megan MacMillan, Gartner Solista  
45 Norbert Mikula, DataChannel  
46 Joel Munter, Intel  
47 Farrukh Najmi, Sun Microsystems  
48 Joel Neu, Vitria Technologies  
49 Sanjay Patil, IONA  
50 Neal Smith, Chevron  
51 Nikola Stojanovic, Encoda Systems Inc.  
52 David Webber, XMLGlobal  
53 Prasad Yendluri, webmethods  
54 Yutaka Yoshida, Sun Microsystems

55  
56  
57

57 **Table of Contents**

58

|    |          |   |           |
|----|----------|---|-----------|
| 59 | <b>1</b> | <b>STATUS OF THIS DOCUMENT</b> .....                            | <b>1</b>  |
| 60 | <b>2</b> | <b>OASIS/EBXML REGISTRY TECHNICAL COMMITTEE</b> .....           | <b>2</b>  |
| 61 |          | <b>INTRODUCTION</b> .....                                       | <b>8</b>  |
| 62 | 2.1      | SUMMARY OF CONTENTS OF DOCUMENT.....                            | 8         |
| 63 | 2.2      | GENERAL CONVENTIONS .....                                       | 8         |
| 64 | 2.2.1    | <i>Naming Conventions</i> .....                                 | 8         |
| 65 | 2.3      | AUDIENCE.....   | 9         |
| 66 | 2.4      | RELATED DOCUMENTS .....   | 9         |
| 67 | <b>3</b> | <b>DESIGN OBJECTIVES</b> .....                                  | <b>9</b>  |
| 68 | 3.1      | GOALS .....   | 9         |
| 69 | <b>4</b> | <b>SYSTEM OVERVIEW</b> .....                                    | <b>10</b> |
| 70 | 4.1      | ROLE OF EBXML <i>REGISTRY</i> .....                             | 10        |
| 71 | 4.2      | <i>REGISTRY SERVICES</i> .....                                  | 10        |
| 72 | 4.3      | WHAT THE REGISTRY INFORMATION MODEL DOES .....                  | 10        |
| 73 | 4.4      | HOW THE REGISTRY INFORMATION MODEL WORKS .....                  | 10        |
| 74 | 4.5      | WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED .....   | 10        |
| 75 | 4.6      | <i>CONFORMANCE TO AN EBXML REGISTRY</i> .....                   | 11        |
| 76 | <b>5</b> | <b>REGISTRY INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW</b> ..... | <b>11</b> |
| 77 | 5.1      | REGISTRYOBJECT .....  | 12        |
| 78 | 5.2      | SLOT.....   | 12        |
| 79 | 5.3      | ASSOCIATION.....  | 12        |
| 80 | 5.4      | EXTERNALIDENTIFIER .....  | 12        |
| 81 | 5.5      | EXTERNALLINK .....  | 12        |
| 82 | 5.6      | CLASSIFICATIONSCHEME.....                                       | 12        |
| 83 | 5.7      | CLASSIFICATIONNODE.....   | 13        |
| 84 | 5.8      | CLASSIFICATION .....  | 13        |
| 85 | 5.9      | PACKAGE.....  | 13        |
| 86 | 5.10     | AUDITABLEEVENT.....   | 13        |
| 87 | 5.11     | USER.....   | 13        |
| 88 | 5.12     | POSTALADDRESS.....  | 13        |
| 89 | 5.13     | ORGANIZATION.....   | 13        |
| 90 | <b>6</b> | <b>REGISTRY INFORMATION MODEL: DETAIL VIEW</b> .....            | <b>14</b> |
| 91 | 6.1      | ATTRIBUTE AND METHODS OF INFORMATION MODEL CLASSES .....        | 14        |
| 92 | 6.2      | DATA TYPES .....  | 15        |
| 93 | 6.3      | CLASS REGISTRYOBJECT .....                                      | 15        |
| 94 | 6.3.1    | <i>Attribute Summary</i> .....                                  | 16        |

|     |          |   |           |
|-----|----------|---|-----------|
| 95  | 6.3.2    | <i>Attribute accesControlPolicy</i> ..... | 16        |
| 96  | 6.3.3    | <i>Attribute description</i> .....        | 16        |
| 97  | 6.3.4    | <i>Attribute id</i> .....                 | 16        |
| 98  | 6.3.5    | <i>Attribute name</i> .....               | 17        |
| 99  | 6.3.6    | <i>Attribute objectType</i> .....         | 17        |
| 100 | 6.3.7    | <i>Method Summary</i> .....               | 19        |
| 101 | 6.4      | CLASS REGISTRYENTRY.....                  | 20        |
| 102 | 6.4.1    | <i>Attribute Summary</i> .....            | 20        |
| 103 | 6.4.2    | <i>Attribute expiration</i> .....         | 20        |
| 104 | 6.4.3    | <i>Attribute majorVersion</i> .....       | 20        |
| 105 | 6.4.4    | <i>Attribute minorVersion</i> .....       | 20        |
| 106 | 6.4.5    | <i>Attribute stability</i> .....          | 21        |
| 107 | 6.4.6    | <i>Attribute status</i> .....             | 21        |
| 108 | 6.4.7    | <i>Attribute userVersion</i> .....        | 22        |
| 109 | 6.5      | CLASS SLOT.....                           | 22        |
| 110 | 6.5.1    | <i>A</i> .....                            | 22        |
| 111 | 6.5.2    | <i>Attribute name</i> .....               | 22        |
| 112 | 6.5.3    | <i>Attribute slotType</i> .....           | 22        |
| 113 | 6.5.4    | <i>Attribute values</i> .....             | 22        |
| 114 | 6.6      | CLASS EXTRINSICOBJECT.....                | 23        |
| 115 | 6.6.1    | <i>Attribute Summary</i> .....            | 23        |
| 116 | 6.6.2    | <i>Attribute contentURI</i> .....         | 23        |
| 117 | 6.6.3    | <i>Attribute isOpaque</i> .....           | 23        |
| 118 | 6.6.4    | <i>Attribute mimeType</i> .....           | 23        |
| 119 | 6.7      | CLASS PACKAGE.....                        | 24        |
| 120 | 6.7.1    | <i>Attribute Summary</i> .....            | 24        |
| 121 | 6.7.2    | <i>Method Summary</i> .....               | 24        |
| 122 | 6.8      | CLASS EXTERNALIDENTIFIER.....             | 24        |
| 123 | 6.8.1    | <i>Attribute Summary</i> .....            | 24        |
| 124 | 6.8.2    | <i>Attribute registryObject</i> .....     | 25        |
| 125 | 6.8.3    | <i>Attribute value</i> .....              | 25        |
| 126 | 6.8.4    | <i>Inherited Attribute id</i> .....       | 25        |
| 127 | 6.8.5    | <i>Inherited Attribute name</i> .....     | 25        |
| 128 | 6.9      | CLASS EXTERNALLINK.....                   | 25        |
| 129 | 6.9.1    | <i>Attribute Summary</i> .....            | 25        |
| 130 | 6.9.2    | <i>Attribute externalURI</i> .....        | 26        |
| 131 | 6.9.3    | <i>Method Summary</i> .....               | 26        |
| 132 | <b>7</b> | <b>REGISTRY AUDIT TRAIL</b> .....         | <b>26</b> |
| 133 | 7.1      | CLASS AUDITABLEEVENT.....                 | 26        |
| 134 | 7.1.1    | <i>Attribute Summary</i> .....            | 27        |
| 135 | 7.1.2    | <i>Attribute eventType</i> .....          | 27        |
| 136 | 7.1.3    | <i>Attribute RegistryObject</i> .....     | 27        |
| 137 | 7.1.4    | <i>Attribute timestamp</i> .....          | 27        |
| 138 | 7.1.5    | <i>Attribute user</i> .....               | 28        |
| 139 | 7.2      | CLASS USER.....                           | 28        |

|     |           |   |           |
|-----|-----------|---|-----------|
| 140 | 7.2.1     | <i>Attribute Summary</i> .....                | 28        |
| 141 | 7.2.2     | <i>Attribute address</i> .....                | 28        |
| 142 | 7.2.3     | <i>Attribute email</i> .....                  | 28        |
| 143 | 7.2.4     | <i>Attribute organization</i> .....           | 28        |
| 144 | 7.2.5     | <i>Attribute personName</i> .....             | 28        |
| 145 | 7.2.6     | <i>Attribute telephoneNumbers</i> .....       | 29        |
| 146 | 7.2.7     | <i>Attribute url</i> .....                    | 29        |
| 147 | 7.3       | <b>CLASS ORGANIZATION</b> .....               | 29        |
| 148 | 7.3.1     | <i>Attribute Summary</i> .....                | 29        |
| 149 | 7.3.2     | <i>Attribute address</i> .....                | 29        |
| 150 | 7.3.3     | <i>Attribute parent</i> .....                 | 29        |
| 151 | 7.3.4     | <i>Attribute primaryContact</i> .....         | 29        |
| 152 | 7.3.5     | <i>Attribute telephoneNumbers</i> .....       | 30        |
| 153 | 7.4       | <b>CLASS POSTALADDRESS</b> .....              | 30        |
| 154 | 7.4.1     | <i>Attribute Summary</i> .....                | 30        |
| 155 | 7.4.2     | <i>Attribute city</i> .....                   | 30        |
| 156 | 7.4.3     | <i>Attribute country</i> .....                | 30        |
| 157 | 7.4.4     | <i>Attribute postalCode</i> .....             | 30        |
| 158 | 7.4.5     | <i>Attribute state</i> .....                  | 30        |
| 159 | 7.4.6     | <i>Attribute street</i> .....                 | 30        |
| 160 | 7.4.7     | <i>Method Summary</i> .....                   | 30        |
| 161 | 7.5       | <b>CLASS TELEPHONENUMBER</b> .....            | 31        |
| 162 | 7.5.1     | <i>Attribute Summary</i> .....                | 31        |
| 163 | 7.5.2     | <i>Attribute areaCode</i> .....               | 31        |
| 164 | 7.5.3     | <i>Attribute countryCode</i> .....            | 31        |
| 165 | 7.5.4     | <i>Attribute extension</i> .....              | 31        |
| 166 | 7.5.5     | <i>Attribute number</i> .....                 | 31        |
| 167 | 7.5.6     | <i>Attribute phoneType</i> .....              | 32        |
| 168 | 7.5.7     | <i>Attribute url</i> .....                    | 32        |
| 169 | 7.6       | <b>CLASS PERSONNAME</b> .....                 | 32        |
| 170 | 7.6.1     | <i>Attribute Summary</i> .....                | 32        |
| 171 | 7.6.2     | <i>Attribute firstName</i> .....              | 32        |
| 172 | 7.6.3     | <i>Attribute lastName</i> .....               | 32        |
| 173 | 7.6.4     | <i>Attribute middleName</i> .....             | 32        |
| 174 | <b>8</b>  | <b>REGISTRYOBJECT NAMING</b> .....            | <b>32</b> |
| 175 | <b>9</b>  | <b>ASSOCIATION OF REGISTRYENTRY</b> .....     | <b>33</b> |
| 176 | 9.1       | <b>CLASS ASSOCIATION</b> .....                | 33        |
| 177 | 9.1.1     | <i>Attribute Summary</i> .....                | 33        |
| 178 | 9.1.2     | <i>Attribute associationType</i> .....        | 34        |
| 179 | 9.1.3     | <i>Attribute sourceObject</i> .....           | 35        |
| 180 | 9.1.4     | <i>Attribute targetObject</i> .....           | 35        |
| 181 | 9.1.5     | <i>Inherited Attribute id</i> .....           | 35        |
| 182 | <b>10</b> | <b>CLASSIFICATION OF REGISTRYOBJECT</b> ..... | <b>35</b> |

|     |           |   |           |
|-----|-----------|---|-----------|
| 183 | 10.1      | CLASS CLASSIFICATIONSCHEME.....                             | 37        |
| 184 | 10.2      | CLASS CLASSIFICATIONNODE.....                               | 38        |
| 185 | 10.2.1    | Attribute Summary.....                                      | 38        |
| 186 | 10.2.2    | Attribute parent.....                                       | 38        |
| 187 | 10.2.3    | Attribute code.....   | 38        |
| 188 | 10.2.4    | Method Summary.....   | 38        |
| 189 | 10.3      | CLASS CLASSIFICATION.....                                   | 39        |
| 190 | 10.3.1    | Attribute Summary.....                                      | 39        |
| 191 | 10.3.2    | Attribute classificationNode .....                          | 39        |
| 192 | 10.3.3    | Attribute classifiedObject .....                            | 39        |
| 193 | 10.3.4    | Inherited Attribute id .....                                | 39        |
| 194 | 10.3.5    | Context Sensitive Classification .....                      | 40        |
| 195 | 10.4      | EXAMPLE OF CLASSIFICATION SCHEMES.....                      | 41        |
| 196 | 10.5      | STANDARDIZED TAXONOMY SUPPORT .....                         | 41        |
| 197 | 10.5.1    | Full-featured Taxonomy Based Classification .....           | 42        |
| 198 | 10.5.2    | Light Weight Taxonomy Based Classification .....            | 42        |
| 199 | <b>11</b> | <b>INFORMATION MODEL: SECURITY VIEW .....</b>               | <b>43</b> |
| 200 | 11.1      | CLASS ACCESSCONTROLPOLICY.....                              | 44        |
| 201 | 11.2      | CLASS PERMISSION .....                                      | 45        |
| 202 | 11.3      | CLASS PRIVILEGE .....                                       | 45        |
| 203 | 11.4      | CLASS PRIVILEGEATTRIBUTE.....                               | 46        |
| 204 | 11.5      | CLASS ROLE .....  | 46        |
| 205 | 11.6      | CLASS GROUP .....   | 46        |
| 206 | 11.7      | CLASS IDENTITY .....  | 46        |
| 207 | 11.8      | CLASS PRINCIPAL.....  | 47        |
| 208 | <b>12</b> | <b>REFERENCES .....</b>                                     | <b>48</b> |
| 209 | <b>13</b> | <b>DISCLAIMER .....</b>                                     | <b>48</b> |
| 210 | <b>14</b> | <b>CONTACT INFORMATION.....</b>                             | <b>49</b> |
| 211 |           | <b>COPYRIGHT STATEMENT.....</b>                             | <b>50</b> |
| 212 |           | <b>Table of Figures</b>                                     |           |
| 213 |           | Figure 2: Information Model High Level Public View.....     | 11        |
| 214 |           | Figure 3: Information Model <i>Inheritance</i> View.....    | 14        |
| 215 |           | Figure 4: Example of RegistryEntry Association.....         | 33        |
| 216 |           | Figure 5: Example showing a <i>Classification</i> Tree..... | 36        |
| 217 |           | Figure 6: Information Model <i>Classification</i> View..... | 37        |
| 218 |           | Figure 7: Classification <i>Instance</i> Diagram .....      | 37        |
| 219 |           | Figure 8: Context Sensitive <i>Classification</i> .....     | 40        |
| 220 |           | Figure 9: Information Model: Security View.....             | 44        |

|     |  |    |
|-----|--|----|
| 221 | <b>Table of Tables</b>                             |    |
| 222 | Table 1: Sample <i>Classification</i> Schemes..... | 41 |
| 223 |  |    |

## 223 Introduction

### 224 2.1 Summary of Contents of Document

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

226

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

### 230 2.2 General Conventions

231

232 ○ *UML* diagrams are used as a way to concisely describe concepts. They  
233 are not intended to convey any specific *Implementation* or methodology  
234 requirements.

235 ○ The term “*repository item*” is used to refer to an object that has been  
236 submitted to a Registry for storage and safekeeping (e.g. an XML  
237 document or a DTD). Every repository item is described by a  
238 RegistryObject instance.

239 ○ The term “RegistryObject” is used to refer to an object that provides  
240 metadata about a repository item. “RegistryObject” is also the name of the  
241 most base class in the information model.

242

243 ○ The information model does not deal with the actual content of the  
244 repository. All *Elements* of the information model represent metadata  
245 about the content and not the content itself.

246

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

249

250 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,  
251 SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in  
252 this document, are to be interpreted as described in RFC 2119 [Bra97].  
253

#### 254 2.2.1 Naming Conventions

255

256 In order to enforce a consistent capitalization and naming convention in this  
257 document, “Upper Camel Case” (*UCC*) and “Lower Camel Case” (*LCC*)  
258 Capitalization styles are used in the following conventions

259

- 260 ○ Element name is in *UCC* convention
- 261 ○ (example: <UpperCamelCaseElement/>).
- 262 ○ Attribute name is in *LCC* convention



- 263 ○ (example: <UpperCamelCaseElement
- 264     lowerCamelCaseAttribute="Whatever"/>).
- 265 ○ *Class*, *Interface* names use UCC convention
- 266 ○ (examples: *ClassificationNode*, *Versionable*).
- 267 ○ Method name uses LCC convention
- 268 ○ (example: *getName()*, *setName()*)
- 269
- 270 Also, *Capitalized Italics* words are defined in the ebXML Glossary [ebGLOSS].

## 271 **2.3 Audience**

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

- 274 ○ Implementers of ebXML *Registry Services*
- 275 ○ Implementers of ebXML *Registry Clients*

## 276 **2.4 Related Documents**

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

- 279
- 280 a) ebXML Registry Services Specification [ebRS] - defines the actual
- 281 *Registry Services* based on this information model
- 282 b) ebXML Collaboration-Protocol Profile and Agreement Specification
- 283 [ebCPP] - defines how profiles can be defined for a *Party* and how two
- 284 *Parties'* profiles may be used to define a *Party* agreement
- 285 c) ebXML Business Process Specification Schema [ebBPSS]
- 286 d) ebXML Technical Architecture Specification [ebTA]
- 287

## 288 **3 Design Objectives**

### 289 **3.1 Goals**

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

- 291 ○ Communicate what information is in the *Registry* and how that information
- 292     is organized
- 293 ○ Leverage as much as possible the work done in the *OASIS* [OAS] and the
- 294 *ISO 11179* [ISO] Registry models
- 295 ○ Align with relevant works within other ebXML working groups
- 296 ○ Be able to evolve to support future ebXML *Registry* requirements
- 297 ○ Be compatible with other ebXML specifications
- 298

## 299 **4 System Overview**

### 300 **4.1 Role of ebXML Registry**

301

302 The *Registry* provides a stable store where information submitted by a  
303 *Submitting Organization* is made persistent. Such information is used to facilitate  
304 ebXML-based *Business to Business* (B2B) partnerships and transactions.

305 Submitted content may be *XML* schema and documents, process descriptions,  
306 *Core Components*, context descriptions, *UML* models, information about parties  
307 and even software components.

### 308 **4.2 Registry Services**

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

### 313 **4.3 What the Registry Information Model Does**

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

318 The Registry information model:

- 319 ○ Defines what types of objects are stored in the *Registry*
- 320 ○ Defines how stored objects are organized in the *Registry*
- 321 ○ Is based on ebXML metamodels from various working groups

322

### 323 **4.4 How the Registry Information Model Works**

324 Implementers of the ebXML *Registry* MAY use the information model to  
325 determine which *Classes* to include in their *Registry Implementation* and what  
326 attributes and methods these *Classes* may have. They MAY also use it to  
327 determine what sort of database schema their *Registry Implementation* may  
328 need.

329 [Note]The information model is meant to be  
330 illustrative and does not prescribe any  
331 specific *Implementation* choices.  
332

### 333 **4.5 Where the Registry Information Model May Be Implemented**

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

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

338 **4.6 Conformance to an ebXML Registry**

339

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

343 **5 Registry Information Model: High Level Public View**

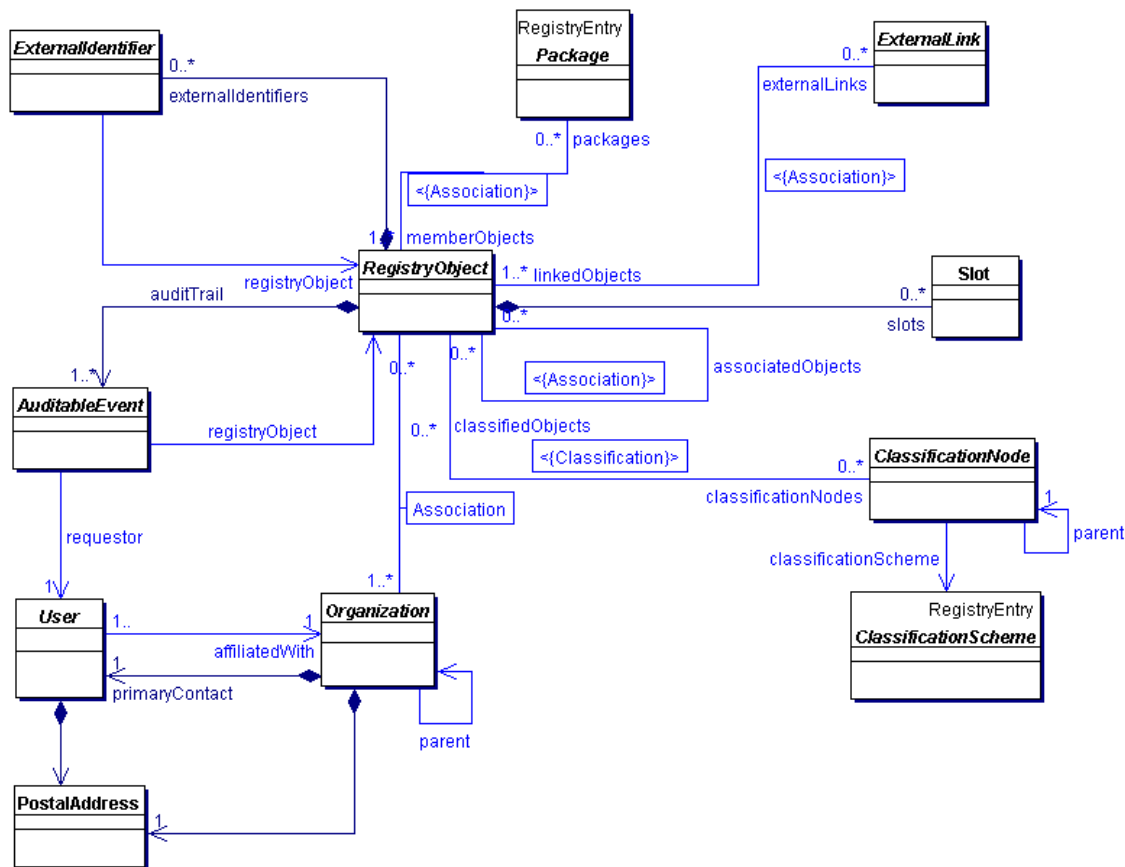
344 This section provides a high level public view of the most visible objects in the  
 345 *Registry*.

346

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

350 The reader is again reminded that the information model is not modeling actual  
 351 repository items.

352



353

354 **Figure 1: Information Model High Level Public View**

## 355 **5.1 RegistryObject**

356 The RegistryObject class is an abstract base class used by most classes in the  
357 model. It provides minimal metadata for registry objects. It also provides methods  
358 for accessing related objects that provide additional dynamic metadata for the  
359 registry object.

## 360 **5.2 Slot**

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

## 367 **5.3 Association**

368 Association instances are RegistryObject instances that are used to define many-  
369 to-many associations between objects in the information model. Associations are  
370 described in detail in section 9.

## 371 **5.4 ExternalIdentifier**

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

## 375 **5.5 ExternalLink**

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

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

## 387 **5.6 ClassificationScheme**

388 A ClassificationScheme instance is a RegistryObject instance that represents a  
389 structured way to classify or categorize RegistryObject instances. A very  
390 common example of a classification scheme in science is the *Classification of*  
391 *living things* where living things are categorized in under a tree like structure.  
392 Another example is the Dewey Decimal system used in libraries to categorize

393 books and other publications. ClassificationScheme is described in detail in  
394 section 10.

### 395 **5.7 ClassificationNode**

396 ClassificationNode instances are RegistryObject instances that are used to  
397 define tree structures under a ClassificationScheme, where each node in the tree  
398 is a ClassificationNode and the root is the ClassificationScheme. *Classification*  
399 trees constructed with ClassificationNodes are used to define *Classification*  
400 schemes or ontologies. ClassificationNode is described in detail in section 10.

### 401 **5.8 Classification**

402 Classification instances are RegistryObjects that are used to classify other  
403 RegistryObject instances with a ClassificationNode within a  
404 ClassificationScheme. Classification is described in detail in section 10.

### 405 **5.9 Package**

406 Package instances are RegistryEntry instances that group logically related  
407 RegistryObject instances together. One use of a Package is to allow operations  
408 to be performed on an entire *Package* of objects. For example all objects  
409 belonging to a Package may be deleted in a single request.

### 410 **5.10 AuditableEvent**

411 AuditableEvent instances are RegistryObject instances that are used to provide  
412 an audit trail for RegistryObject instances. AuditableEvent is described in detail in  
413 section 7.

### 414 **5.11 User**

415 User instances are RegistryObject instances that are used to provide information  
416 about registered users within the *Registry*. User objects are used in audit trail for  
417 RegistryObject instances. User is described in detail in section 7.

### 418 **5.12 PostalAddress**

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

421

### 422 **5.13 Organization**

423 Organization instances are RegistryObject instances that provide information on  
424 organizations such as a *Submitting Organization*. Each Organization instance  
425 may have a reference to a parent Organization.

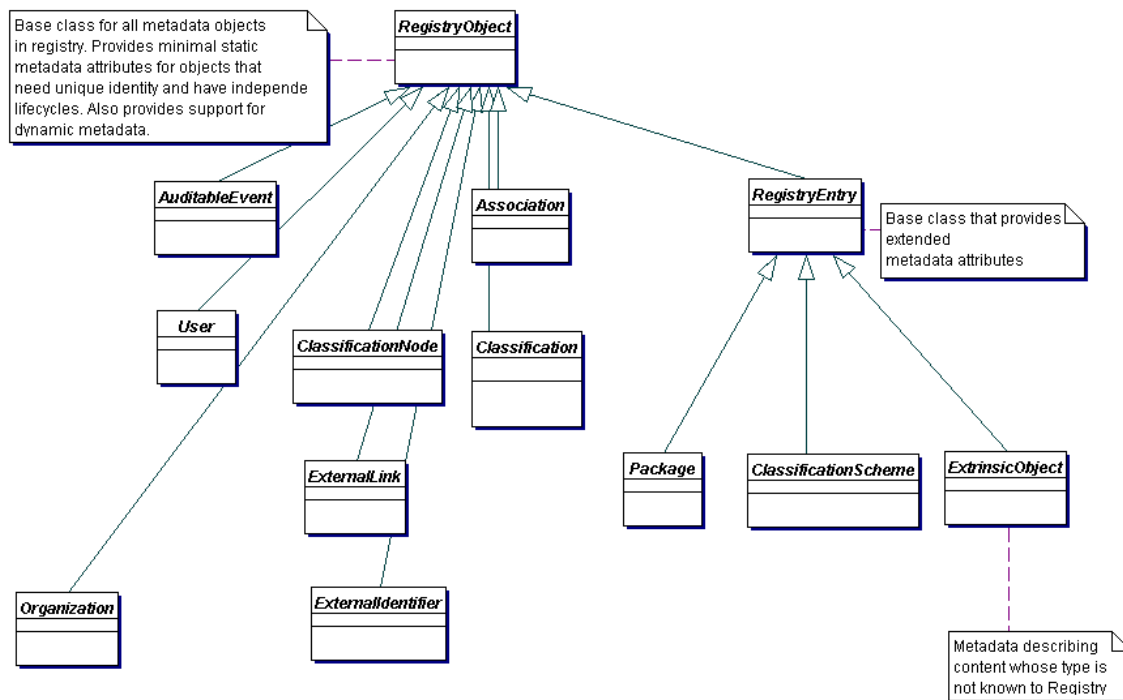
## 426 6 Registry Information Model: Detail View

427 This section covers the information model *Classes* in more detail than the Public  
 428 View. The detail view introduces some additional *Classes* within the model that  
 429 were not described in the public view of the information model.

430  
 431 Figure 2 shows the *Inheritance* or “is a” relationships between the *Classes* in the  
 432 information model. Note that it does not show the other types of relationships,  
 433 such as “has a” relationships, since they have already been shown in a previous  
 434 figure. *Class* attributes and *class* methods are also not shown. Detailed  
 435 description of methods and attributes of most interfaces and *Classes* will be  
 436 displayed in tabular form following the description of each *Class* in the model.

437  
 438 The class Association will be covered in detail separately in section 9. The  
 439 classes Classification and ClassificationNode will be covered in detail separately  
 440 in section 10.

441  
 442 The reader is again reminded that the information model is not modeling actual  
 443 repository items.



444

445

Figure 2: Information Model *Inheritance* View

### 446 6.1 Attribute and Methods of Information Model Classes

447 Information model classes are defined primarily in terms of the attributes they  
 448 carry. These attributes provide state information on instances of these classes.  
 449 Implementations of a registry often map class attributes to attributes in an XML  
 450 store or columns in a relational store.

OASIS/ebXML Registry Information Model

451  
 452 Information model classes may also have methods defined for them. These  
 453 methods provide additional behavior for the class they are defined within.  
 454 Methods are currently used in mapping to SQL stored procedures in the SQL  
 455 query capability defined in [ebRS].  
 456  
 457 Since the model supports inheritance between classes, it is usually the case that  
 458 a class in the model inherits attributes and methods from its base classes, in  
 459 addition to defining its own specialized attributes and methods.

## 460 6.2 Data Types

461 This following table lists the various data types used by the attributes within  
 462 information model classes:  
 463

| Data Type    | Primitive Type | Description  | Length         |
|--------------|----------------|--|----------------|
| Boolean      |                | Used for a true or false value   |                |
| String4      | String         | Used for 4 character long strings                                      | 4 characters   |
| String8      | String         | Used for 8 character long strings                                      | 8 characters   |
| ShortName    | String         | A short text string  | 64 characters  |
| LongName     | String         | A long text string   | 128 characters |
| FreeFormText | String         | A very long text string for free-form text                             | 256 characters |
| UUID         | String         | DCE 128 Bit Universally unique Ids used for referencing another object | 64 characters  |
| URI          | String         | Used for URL and URN values  | 256 characters |
| Integer      |                | Used for integer values  | 4 bytes        |
| Timestamp    |                | Used for a time stamp value such as Date                               |                |

464

## 465 6.3 Class RegistryObject

466 **Direct Known Subclasses:**

467 [Association](#), [AuditableEvent](#), [Classification](#), [ClassificationNode](#),  
 468 [ExternalIdentifier](#), [ExternalLink](#), [Organization](#), [RegistryEntry](#), [User](#)

469

470 RegistryObject provides a common base class for almost all objects in the  
 471 information model. Information model *Classes* whose instances have a unique  
 472 identity and an independent life cycle are descendants of the RegistryObject  
 473 *Class*.

474

475 Note that Slot and PostalAddress are not descendants of the RegistryObject  
 476 *Class* because their instances do not have an independent existence and unique

477 identity. They are always a part of some other *Class's Instance* (e.g. Organization  
478 has a PostalAddress).

### 479 **6.3.1 Attribute Summary**

480 The following is the first of many tables that summarize the attributes of a class.

481 The columns in the table are described as follows:

| Column       | Description   |
|--------------|---|
| Attribute    | The name of the attribute   |
| Data Type    | The data type for the attribute   |
| Required     | Specifies whether the attribute is required to be specified   |
| Default      | Specifies the default value in case the attribute is omitted  |
| Specified By | Indicates whether the attribute is specified by the client or specified by the registry. In some cases it may be both |
| Mutable      | Specifies whether an attribute may be changed once it has been set to a certain value                                 |

482

483

| Attribute           | Data Type    | Required | Default Value | Specified By       | Mutable |
|---------------------|--------------|----------|---------------|--------------------|---------|
| accessControlPolicy | UUID         | No       |               | Registry           | No      |
| description         | FreeFormText | No       |               | Client             | Yes     |
| id                  | UUID         | Yes      |               | Client or registry | No      |
| name                | LongName     | No       |               | Client             | Yes     |
| objectType          | LongName     | Yes      |               | Registry           | No      |

484

### 485 **6.3.2 Attribute accessControlPolicy**

486 Each RegistryObject instance has an AccessControlPolicy instance associated  
487 with it. An AccessControlPolicy instance defines the *Security Model* associated  
488 with the RegistryObject in terms of "who is permitted to do what" with that  
489 RegistryObject.

### 490 **6.3.3 Attribute description**

491 Each RegistryObject instance may have textual description in a human readable  
492 and user-friendly manner.

### 493 **6.3.4 Attribute id**

494 Each RegistryObject instance must have a universally unique ID. Registry  
495 objects use the id of other RegistryObject instances for the purpose of  
496 referencing those objects.

497



498 Note that some classes in the information model do not have a need for a unique  
499 id. Such classes do not inherit from RegistryObject class. Examples include  
500 Entity classes such as TelephoneNumber, PostalAddress and PersonName.

501

502 The id attribute of various derived classes of RegistryObject fall into two  
503 categories:

504

- 505 1. UUID based Id
- 506 2. Attribute based Id

507

#### 508 **6.3.4.1 UUID based Id**

509 Most classes derived from RegistryObject have an id that is a Universally Unique  
510 ID as defined by [UUID]. Such UUID based Id attributes may be specified by the  
511 client. If the UUID based ID is not specified, it must be generated by the registry  
512 when a new RegistryObject instance is first submitted to the registry.

#### 513 **6.3.4.2 Attribute based Id**

514 A few classes derived from RegistryObject have an Id that is not a UUID but is  
515 instead composed of multiple attributes of that object. This is very similar to the  
516 concept of a multi-column primary key in relational databases, or multi-attribute  
517 key instances in XML Schema.

518

519 Examples of classes that use attribute based Id are Classification, Association  
520 and ExternalIdentifier. The reason these objects do not use UUIDs and instead  
521 use attribute based Id is that they do not have an independent lifecycle separate  
522 from their primary RegistryObject.

523

524 Attribute based Ids are not UUIDs and therefore are not constrained by the 64 bit  
525 limit of the UUID data type. Instead they can be of arbitrary length.

#### 526 **6.3.5 Attribute name**

527 Each RegistryObject instance may have human readable name. The name does  
528 not need to be unique with respect other RegistryObject instances.

#### 529 **6.3.6 Attribute objectType**

530 Each RegistryObject instance has an objectType. The objectType for almost all  
531 objects in the information model is the name of their class. For example the  
532 objectType for a Classification is "Classification". The only exception to this rule  
533 is that the objectType for an ExtrinsicObject instance is user defined and  
534 indicates the type of repository item associated with the ExtrinsicObject.

##### 535 **6.3.6.1 Pre-defined Object Types**

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

540  
541  
542  
543  
544

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

| <b>Name</b>               | <b>description</b>  |
|---------------------------|---|
| <b>Unknown</b>            | An ExtrinsicObject that catalogues content whose type is unspecified or unknown.  |
| <b>CPA</b>                | An ExtrinsicObject of this type catalogues an <i>XML</i> document <i>Collaboration Protocol Agreement (CPA)</i> representing a technical agreement between two parties on how they plan to communicate with each other using a specific protocol. |
| <b>CPP</b>                | An ExtrinsicObject of this type catalogues an document called <i>Collaboration Protocol Profile (CPP)</i> that provides information about a <i>Party</i> participating in a <i>Business</i> transaction.  |
| <b>Process</b>            | An ExtrinsicObject of this type catalogues a process description document.  |
| <b>Role</b>               | An ExtrinsicObject of this type catalogues an <i>XML</i> description of a <i>Role</i> in a <i>Collaboration Protocol Profile (CPP)</i> .  |
| <b>ServiceInterface</b>   | An ExtrinsicObject of this type catalogues an <i>XML</i> description of a service interface as defined by [ebCPP].  |
| <b>SoftwareComponent</b>  | An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or <i>Class</i> library).   |
| <b>Transport</b>          | An ExtrinsicObject of this type catalogues an <i>XML</i> description of a transport configuration as defined by [ebCPP].  |
| <b>UMLModel</b>           | An ExtrinsicObject of this type catalogues a <i>UML</i> model.  |
| <b>XMLSchema</b>          | An ExtrinsicObject of this type catalogues an <i>XML</i> schema ( <i>DTD</i> , <i>XML</i> Schema, RELAX grammar, etc.).   |
| <b>Package</b>            | A Package object  |
| <b>ExternalLink</b>       | An ExternalLink object  |
| <b>ExternalIdentifier</b> | An ExternalIdentifier object  |
| <b>Association</b>        | An Association object   |
| <b>Classification</b>     | A Classification object   |
| <b>ClassificationNode</b> | A ClassificationNode object   |
| <b>AuditableEvent</b>     | An AuditableEvent object  |
| <b>User</b>               | A User object   |
| <b>Organization</b>       | An Organization object  |

545

546 **6.3.7 Method Summary**

547 In addition to its attributes, the RegistryObject class also defines the following  
 548 methods. These methods are used to navigate relationship links from a  
 549 RegistryObject instance to other objects.  
 550

| <b>Method Summary for RegistryObject</b> |   |
|--|---|
| Collection <sup>1</sup>                  | <a href="#">getAssociatedObjects()</a><br>Gets the collection of RegistryObject instances associated with this object.  |
| Collection                               | <a href="#">getAssociations()</a><br>Gets all Associations where this object is the source of the Association.  |
| Collection                               | <a href="#">getAuditTrail()</a><br>Gets the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.  |
| Collection                               | <a href="#">getClassificationNodes()</a><br>Gets the ClassificationNodes that classify this object.   |
| Collection                               | <a href="#">getClassifications()</a><br>Gets the Classification that classify this object.  |
| Collection                               | <a href="#">getExternalIdentifiers()</a><br>Gets the collection of ExternalIdentifiers associated with this object.   |
| Collection                               | <a href="#">getExternalLinks()</a><br>Gets the ExternalLinks associated with this object.   |
| Collection                               | <a href="#">getOrganizations(String type)</a><br>Gets the Organizations associated with this object. If a non-null type is specified it is used as a filter to match only specified type of organizations as indicated by the associationType attribute in the Association instance linking the object to the Organization. |
| Collection                               | <a href="#">getPackages()</a><br>Gets the Packages that this object is a member of.   |
| Collection                               | <a href="#">getSlots()</a><br>Gets the Slots associated with this object.   |

551  
552

---

<sup>1</sup> A Collection represents a collection of multiple RegistryObject instances  
 OASIS/ebXML Registry Information Model

## 553 **6.4 Class RegistryEntry**

### 554 **Super Classes:**

555 [RegistryObject](#)

556

### 557 **Direct Known Subclasses:**

558 [ClassificationScheme](#), [ExtrinsicObject](#), [Package](#)

559

560 RegistryEntry is a common base *Class* for classes in the information model that  
561 require additional metadata beyond the minimal metadata provided by  
562 RegistryObject class. The additional metadata is described by the attributes of  
563 the RegistryEntry class below.

### 564 **6.4.1 Attribute Summary**

565

| Attribute           | Data Type | Required | Default Value | Specified By | Mutable |
|---------------------|-----------|----------|---------------|--------------|---------|
| expiration          | Timestamp | No       |               | Client       | Yes     |
| majorVersion        | Integer   | Yes      | 1             | Registry     | Yes     |
| minorVersion        | Integer   | Yes      | 0             | Registry     | Yes     |
| stability           | LongName  | No       |               | Client       | Yes     |
| status <sup>2</sup> | LongName  | Yes      |               | Registry     | Yes     |
| userVersion         | ShortName | No       |               | Client       | Yes     |

566

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

### 569 **6.4.2 Attribute expiration**

570 Each RegistrEntry instance may have an expirationDate. This attribute defines a  
571 time limit upon the stability guarantee provided by the stability attribute. Once the  
572 expirationDate has been reached the stability attribute in effect becomes  
573 STABILITY\_DYNAMIC implying that content can change at any time and in any  
574 manner. A null value implies that there is no expiration on stability attribute.

### 575 **6.4.3 Attribute majorVersion**

576 Each RegistrEntry instance must have a major revision number for the current  
577 version of the RegistryEntry instance. This number is assigned by the registry  
578 when the object is created. This number may be updated by the registry when an  
579 object is updated.

### 580 **6.4.4 Attribute minorVersion**

581 Each RegistrEntry instance must have a minor revision number for the current  
582 version of the RegistryEntry instance. This number is assigned by the registry

---

<sup>2</sup> Was Integer in RIM 1.0 for some reason.

583 when the object is created. This number may be updated by the registry when an  
584 object is updated.

#### 585 **6.4.5 Attribute stability**

586 Each RegistrEntry instance may have a stability indicator. The stability indicator  
587 is provided by the submitter as a guarantee of the level of stability for the content.

##### 588 **6.4.5.1 Pre-defined RegistryEntry Stability Enumerations**

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

593

| 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.                                      |

594

#### 595 **6.4.6 Attribute status**

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

##### 598 **6.4.6.1 Pre-defined RegistryObject Status Types**

599 The following table lists pre-defined choices for RegistryObject status attribute.  
600 These pre-defined status types are defined as a *Classification* scheme. While the  
601 scheme may easily be extended, a *Registry* MUST support the status types listed  
602 below.

603

| Name       | Description   |
|------------|---|
| Submitted  | Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> .                                     |
| Approved   | Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> and has been subsequently approved.   |
| Deprecated | Status of a RegistryObject that catalogues content that has been submitted to the <i>Registry</i> and has been subsequently deprecated. |

|           |   |
|-----------|---|
| Withdrawn | Status of a RegistryObject that catalogues content that has been withdrawn from the <i>Registry</i> . |
|-----------|---|

604

#### 605 **6.4.7 Attribute userVersion**

606 Each RegistrEntry instance may have a userVersion. The userVersion is similar  
 607 to the majorVersion-minorVersion tuple. They both provide an indication of the  
 608 version of the object. The majorVersion-minorVersion tuple is provided by the  
 609 registry while userVersion provides a user specified version for the object.

### 613 **6.5 Class Slot**

614 Slot instances provide a dynamic way to add arbitrary attributes to  
 615 RegistryObject instances. This ability to add attributes dynamically to  
 616 RegistryObject instances enables extensibility within the information model.  
 617

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

#### 620 **6.5.1 Attribute Summary**

621

| Attribute | Data Type               | Required | Default Value | Specified By | Mutable |
|-----------|-------------------------|----------|---------------|--------------|---------|
| name      | LongName                | Yes      |               | Client       | No      |
| slotType  | LongName                | Yes      |               | Client       | No      |
| values    | Collection of ShortName | Yes      |               | Client       | No      |

622

#### 623 **6.5.2 Attribute name**

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

#### 627 **6.5.3 Attribute slotType**

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

#### 630 **6.5.4 Attribute values**

631 A Slot instance must have a Collection of values. Since a Slot represent an  
 632 extensible attribute whose value may be a collection, therefore a Slot is allowed  
 633 to have a collection of values rather than a single value.

## 634 **6.6 Class ExtrinsicObject**

### 635 **Super Classes:**

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

637

638

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

642

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

646

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

### 649 **6.6.1 Attribute Summary**

650

| Attribute   | Data Type | Required | Default Value | Specified By | Mutable |
|-------------|-----------|----------|---------------|--------------|---------|
| contentURI  | URI       | Yes      |               | Registry     | No      |
| isOpaque    | Boolean   | No       | false         | Client       | No      |
| contentType | LongName  | Yes      |               | Client       | No      |

651

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

### 654 **6.6.2 Attribute contentURI**

655 Each ExtrinsicObject instance must have a contentURI attribute. The contentURI  
656 is a URI to the repository item that is catalogued by this ExtrinsicObject instance.  
657 The contentURI is assigned by the Registry and must be resolvable by the  
658 registry.

### 659 **6.6.3 Attribute isOpaque**

660 Each ExtrinsicObject instance may have an isOpaque attribute defined. This  
661 attribute determines whether the content catalogued by this ExtrinsicObject is  
662 opaque to (not readable by) the *Registry*. In some situations, a *Submitting*  
663 *Organization* may submit content that is encrypted and not even readable by the  
664 *Registry*.

### 665 **6.6.4 Attribute mimeType**

666 Each ExtrinsicObject instance may have a mimeType attribute defined. The  
667 mimeType provides information on the type of repository item catalogued by the  
668 ExtrinsicObject instance.

## 669 **6.7 Class Package**

### 670 **Super Classes:**

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

672

673 Packages allow for grouping of logically related RegistryObject instances even if  
674 individual member objects belong to different Submitting Organizations.

### 675 **6.7.1 Attribute Summary**

676

677 The Package class defines no new attributes other than those that are inherited  
678 from RegistryEntry and RegistryObject base classes. The inherited attributes are  
679 not shown here.

### 680 **6.7.2 Method Summary**

681 In addition to its attributes, the Package class also defines the following methods.

682

| Method Summary of Package |   |
|---------------------------|---|
| Collection                | <a href="#">getMemberObjects()</a><br>Get the collection of RegistryObject instances that are members of this Package. Maps to attribute named memberObjects. |

683

## 684 **6.8 Class ExternalIdentifier**

### 685 **Super Classes:**

686 [RegistryObject](#)

687

688 ExternalIdentifier instances provide the additional identifier information to  
689 RegistryObject such as DUNS number, Social Security Number, or an alias  
690 name of the organization. The attribute *name* inherited from RegistryObject is  
691 used to contain the identification scheme (Social Security Number, etc), and the  
692 attribute *value* contains the actual information. Each RegistryObject may contain  
693 0 or more ExternalIdentifier instances.

### 694 **6.8.1 Attribute Summary**

695

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

696

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



## 698 **6.8.2 Attribute registryObject**

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

## 701 **6.8.3 Attribute value**

702 Each ExternalIdentifier instance must have a value attribute which provides the  
703 identifier value for this ExternalIdentifier (e.g. social security number).

## 704 **6.8.4 Inherited Attribute id**

705 The id attribute for an ExternalIdentifier is an attribute based id composed of the  
706 value of the registryObject and the name attributes in that order, where each  
707 attribute value is separated by a ‘.’.

708

709 The pattern is as follows:

710 urn:uuid:<RegistryObject id>:<name>

711

712 An example is as follows:

713

714 urn:uuid:a2345678-1234-1234-123456789012:Social Security Number

## 715 **6.8.5 Inherited Attribute name**

716 An ExternalIdentifier instance for a RegistryObject instance must have a unique  
717 name among all other ExternalIdentifier instances for that RegistryObject  
718 instance.

## 719 **6.9 Class ExternalLink**

720 **Super Classes:**

721 [RegistryObject](#)

722

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

### 727 **6.9.1 Attribute Summary**

728

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

729

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

## 731 **6.9.2 Attribute externalURI**

732 Each ExternalLink instance must have an externalURI attribute defined. The  
733 externalURI attribute provides a URI to the external resource pointed to by this  
734 ExternalLink instance.

## 735 **6.9.3 Method Summary**

736 In addition to its attributes, the ExternalLink class also defines the following  
737 methods.

738

| Method Summary of ExternalLink |  |
|--------------------------------|--|
| Collection                     | <a href="#">getLinkedObjects()</a><br>Gets the collection of RegistryObjects that are linked by this ExternalLink to content outside the registry. |

739

740 Note that methods inherited from the base classes of this class are not shown.

## 741 **7 Registry Audit Trail**

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

747

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

### 754 **7.1 Class AuditableEvent**

755 **Super Classes:**

756 [RegistryObject](#)

757

758 AuditableEvent instances provide a long-term record of *Events* that effect a  
759 change of state in a RegistryObject. A RegistryObject is associated with an  
760 ordered Collection of AuditableEvent instances that provide a complete audit trail  
761 for that RegistryObject.

762

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

765

766 Often such *Events* effect a change in the life cycle of a RegistryObject. For  
767 example a client request could Create, Update, Deprecate or Delete a

768 RegistryObject. No AuditableEvent is created for requests that do not alter the  
 769 state of a RegistryObject. Specifically, read-only requests do not generate an  
 770 AuditableEvent. No AuditableEvent is generated for a RegistryObject when it is  
 771 classified, assigned to a Package or associated with another RegistryObject.

### 772 7.1.1 Attribute Summary

773

| Attribute      | Data Type | Required | Default Value | Specified By | Mutable |
|----------------|-----------|----------|---------------|--------------|---------|
| eventType      | LongName  | Yes      |               | Registry     | No      |
| registryObject | UUID      | Yes      |               | Registry     | No      |
| timestamp      | Timestamp | Yes      |               | Registry     | No      |
| user           | UUID      | Yes      |               | Registry     | No      |

774

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

### 776 7.1.2 Attribute eventType

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

#### 779 7.1.2.1 Pre-defined Auditable Event Types

780 The following table lists pre-defined auditable event types. These pre-defined  
 781 event types are defined as a pre-defined *ClassificationScheme* with name  
 782 "EventType". While this scheme may easily be extended, a *Registry* MUST  
 783 support the event types listed below.

784

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

### 785 7.1.3 Attribute RegistryObject

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

### 788 7.1.4 Attribute timestamp

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

791 **7.1.5 Attribute user**

792 Each AuditableEvent must have a timestamp attribute that identifies the User that  
 793 sent the request that generated this event affecting the RegistryObject instance.  
 794

795 **7.2 Class User**796 **Super Classes:**

797 [RegistryObject](#)

798

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

801 **7.2.1 Attribute Summary**

802

| Attribute        | Data Type                     | Required | Default Value | Specified By | Mutable |
|------------------|-------------------------------|----------|---------------|--------------|---------|
| address          | PostalAddress                 | Yes      |               | Client       | Yes     |
| email            | LongName                      | Yes      |               | Client       | Yes     |
| organization     | UUID                          | Yes      |               | Client       | No      |
| personName       | PersonName                    | Yes      |               | Client       | No      |
| telephoneNumbers | Collection of TelephoneNumber | Yes      |               | Client       | Yes     |
| url              | URI                           | No       |               | Client       | Yes     |

803

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

805

806 **7.2.2 Attribute address**

807 Each User instance must have an address attribute that provides the postal  
 808 address for that user.

809 **7.2.3 Attribute email**

810 Each User instance must have an email attribute that provides the email address  
 811 for that user.

812 **7.2.4 Attribute organization**

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

815 **7.2.5 Attribute personName**

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

## 818 **7.2.6 Attribute telephoneNumbers**

819 Each User instance must have a telephoneNumbers attribute that contains the  
820 Collection of TelephoneNumber instances for each telephone number defined for  
821 that user.

## 822 **7.2.7 Attribute url**

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

## 825 **7.3 Class Organization**

### 826 **Super Classes:**

827 [RegistryObject](#)

828

829 Organization instances provide information on organizations such as a  
830 *Submitting Organization*. Each Organization *Instance* may have a reference to a  
831 parent Organization.

### 832 **7.3.1 Attribute Summary**

833

| Attribute        | Data Type                     | Required | Default Value | Specified By | Mutable |
|------------------|-------------------------------|----------|---------------|--------------|---------|
| address          | PostalAddress                 | Yes      |               | Client       | Yes     |
| parent           | UUID                          | No       |               | Client       | Yes     |
| primaryContact   | UUID                          | Yes      |               | Client       | No      |
| telephoneNumbers | Collection of TelephoneNumber | Yes      |               | Client       | Yes     |

834

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

### 836 **7.3.2 Attribute address**

837 Each Organization instance must have an address attribute that provides the  
838 postal address for that organization.

### 839 **7.3.3 Attribute parent**

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

### 842 **7.3.4 Attribute primaryContact**

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

845 **7.3.5 Attribute telephoneNumbers**

846 Each Organization instance must have a telephoneNumbers attribute that  
847 contains the Collection of TelephoneNumber instances for each telephone  
848 number defined for that organization.

849 **7.4 Class PostalAddress**

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

852 **7.4.1 Attribute Summary**

853

| Attribute  | Data Type | Required | Default Value | Specified By | Mutable |
|------------|-----------|----------|---------------|--------------|---------|
| City       | ShortName | No       |               | Client       | Yes     |
| country    | ShortName | No       |               | Client       | Yes     |
| postalCode | ShortName | No       |               | Client       | No      |
| state      | ShortName | No       |               | Client       | Yes     |
| street     | ShortName | No       |               | Client       | Yes     |

854

855 **7.4.2 Attribute city**

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

857 **7.4.3 Attribute country**

858 Each PostalAddress may have a country attribute identifying the country for that  
859 address.

860 **7.4.4 Attribute postalCode**

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

863 **7.4.5 Attribute state**

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

866 **7.4.6 Attribute street**

867 Each PostalAddress may have a street attribute identifying the street address for  
868 that address.

869 **7.4.7 Method Summary**

870 In addition to its attributes, the PostalAddress class also defines the following  
871 methods.

872

| Method Summary of ExternalLink |  |
|--------------------------------|--|
| Collection                     | <a href="#">getSlots()</a><br>Gets the collection of Slots for this object. Each PostalAddress may have multiple Slot instances where a Slot is a dynamically defined attribute. The use of Slots allows the client to extend PostalAddress class by defining additional dynamic attributes using slots to handle locale specific needs. |

873

## 874 **7.5 Class TelephoneNumber**

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

### 876 **7.5.1 Attribute Summary**

877

| Attribute   | Data Type | Required | Default Value | Specified By | Mutable |
|-------------|-----------|----------|---------------|--------------|---------|
| areaCode    | String4   | No       |               | Client       | Yes     |
| countryCode | String4   | No       |               | Client       | Yes     |
| extension   | String8   | No       |               | Client       | Yes     |
| number      | String8   | No       |               | Client       | Yes     |
| phoneType   | LongName  | No       |               | Client       | Yes     |
| url         | URI       | No       |               | Client       | Yes     |

878

### 879 **7.5.2 Attribute areaCode**

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

### 882 **7.5.3 Attribute countryCode**

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

### 885 **7.5.4 Attribute extension**

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

### 888 **7.5.5 Attribute number**

889 Each TelephoneNumber instance may have an number attribute that provides  
 890 the local number (without area code, country code and extension) for that  
 891 telephone number.

### 892 **7.5.6 Attribute phoneType**

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

### 895 **7.5.7 Attribute url**

896 Each TelephoneNumber instance may have a url attribute that provides the url, if  
897 any, associated with that telephone number. It is anticipated that it will be  
898 possible to dial telephone numbers via URLs sometime in the future. Do we need  
899 this or should we remove it??

## 900 **7.6 Class PersonName**

901 \_\_\_\_\_  
902 A simple *Entity Class* for a person's name.

### 903 **7.6.1 Attribute Summary**

904

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

905

### 906 **7.6.2 Attribute firstName**

907 Each PersonName may have a firstName attribute that is the first name of the  
908 person.

### 909 **7.6.3 Attribute lastName**

910 Each PersonName may have a lastName attribute that is the last name of the  
911 person.

### 912 **7.6.4 Attribute middleName**

913 Each PersonName may have a middleName attribute that is the middle name of  
914 the person.

## 915 **8 RegistryObject Naming**

916 A RegistryObject has a name that may or may not be unique within the *Registry*.

917

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

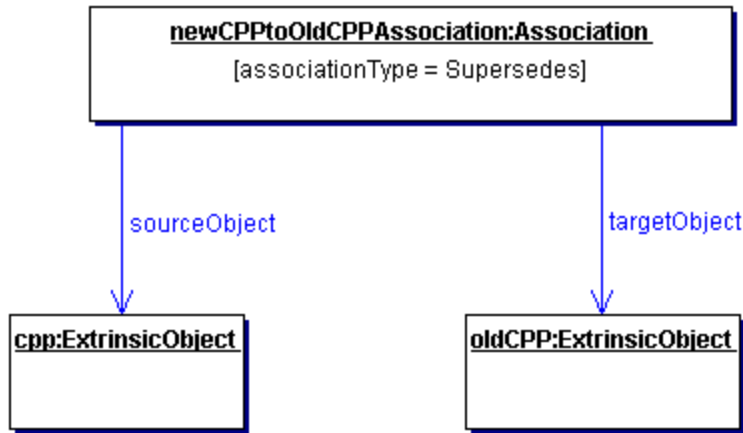
920 Alternate contextual naming will be addressed in a later version of the Registry  
921 Information Model.



922 <FSN: We should remove this chapter??>

923 **9 Association of RegistryEntry**

924 A RegistryObject may be associated with 0 or more RegistryObjects. The  
 925 information model defines an Association Class. An Instance of the Association  
 926 Class represents an association between a source RegistryObject and another  
 927 target RegistryObject. An example of such an association is between an  
 928 ExtrinsicObject instance that catalogues a new Collaboration Protocol Profile  
 929 (CPP) and another ExtrinsicObject instance that catalogues an older  
 930 Collaboration Protocol Profile where the newer CPP supersedes the older CPP  
 931 as shown in Figure 3.



932  
 933 **Figure 3: Example of RegistryEntry Association**

934  
 935 **9.1 Class Association**

936 **Super Classes:**

937 [RegistryObject](#)

938  
 939  
 940 Association instances are used to define many-to-many associations between  
 941 RegistryObjects in the information model.

942  
 943 An Instance of the Association Class represents an association between two  
 944 RegistryObjects.

945 **9.1.1 Attribute Summary**

946

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

|              |      |     |  |        |    |
|--------------|------|-----|--|--------|----|
| targetObject | UUID | Yes |  | Client | No |
|--------------|------|-----|--|--------|----|

947

948

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

949

### 9.1.2 Attribute associationType

950

951

952

Each Association must have an associationType attribute that identifies the type of that association. This MUST be the name attribute of an association type as defined by 1.1.1.

953

#### 9.1.2.1 Pre-defined Association Types

954

955

956

957

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

| name                 | description  |
|----------------------|--|
| RelatedTo            | Defines that source RegistryObject is related to target RegistryObject.  |
| HasMember            | Defines that the source Package object has the target RegistryObject object as a member. Reserved for use in Packaging of RegistryEntries.                               |
| ExternallyLinks      | Defines that the source ExternalLink object externally links the target RegistryObject object. Reserved for use in associating ExternalLinks with RegistryEntries.       |
| ExternallyIdentifies | Defines that the source ExternalIdentifier object identifies the target RegistryObject object. Reserved for use in associating ExternalIdentifiers with RegistryEntries. |
| ContainedBy          | Defines that source RegistryObject is contained by the target RegistryObject.  |
| Contains             | Defines that source RegistryObject contains the target RegistryObject.   |
| Extends              | Defines that source RegistryObject inherits from or specializes the target RegistryObject.   |
| Implements           | Defines that source RegistryObject implements the functionality defined by the target RegistryObject.  |
| InstanceOf           | Defines that source RegistryObject is an <i>Instance</i> of target RegistryObject.   |
| SupersededBy         | Defines that the source RegistryObject is superseded by the target RegistryObject.   |
| Supersedes           | Defines that the source RegistryObject supersedes the target RegistryObject.   |
| UsedBy               | Defines that the source RegistryObject is used by the target RegistryObject in some manner.  |

|                   |   |
|-------------------|---|
| <b>Uses</b>       | Defines that the source RegistryObject uses the target RegistryObject in some manner.           |
| <b>ReplacedBy</b> | Defines that the source RegistryObject is replaced by the target RegistryObject in some manner. |
| <b>Replaces</b>   | Defines that the source RegistryObject replaces the target RegistryObject in some manner.       |

958

959 **9.1.3 Attribute sourceObject**

960 Each Association must have a sourceObject attribute that references the  
961 RegistryObject instance that is the source or owner of that association.

962 **9.1.4 Attribute targetObject**

963 Each Association must have an targetObject attribute that references the  
964 RegistryObject instance that is the target of that association.

965 **9.1.5 Inherited Attribute id**

966 The id attribute for an Association is an attribute based id composed of the value  
967 of the sourceObject, targetObject and associationType attributes in that order,  
968 where each attribute value is separated by a ‘.’.

969

970 The pattern is as follows:

971 urn:uuid:< sourceObject id>:< targetObject id>:<associationType>

972

973 An example is as follows:

974

975 urn:uuid:a2345678-1234-1234-123456789012: a2345678-1234-1234-

976

123456789013:Implements

978 **10 Classification of RegistryObject**

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

981

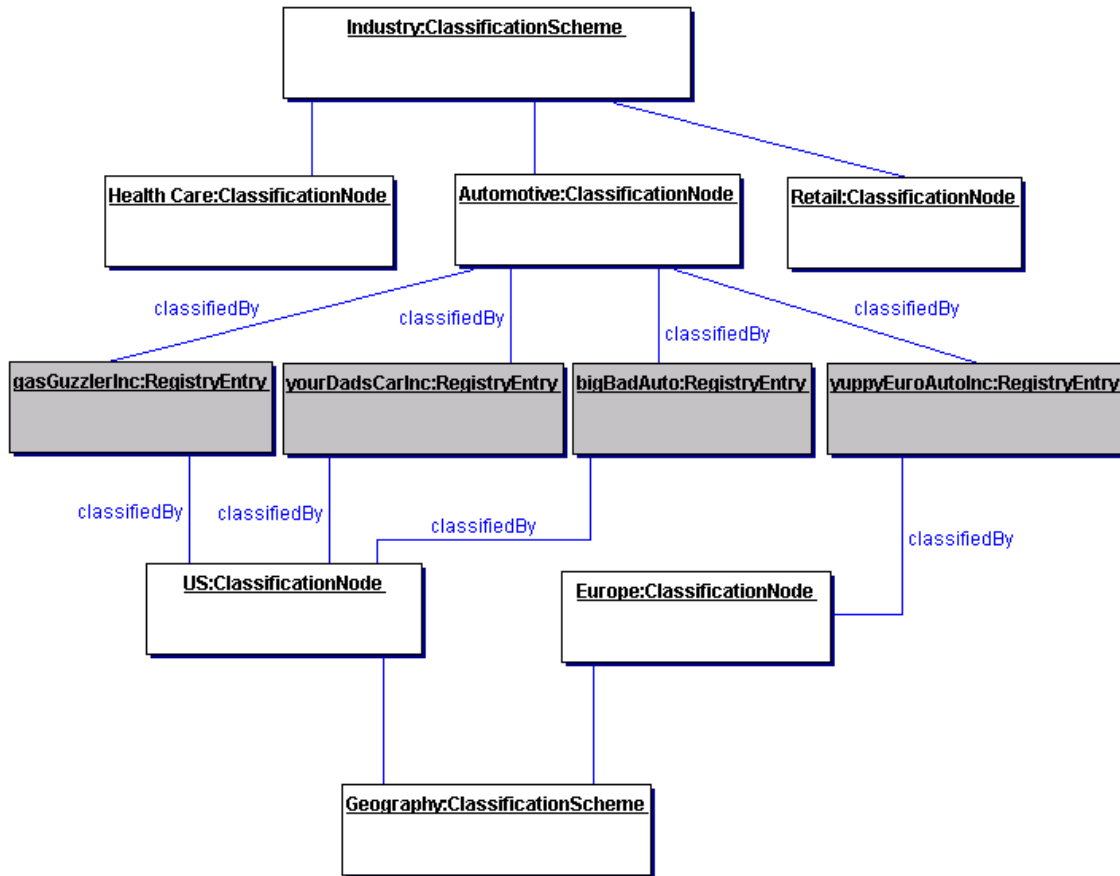
982 A RegistryObject may be classified in many ways. For example the  
983 RegistryObject for the same *Collaboration Protocol Profile (CPP)* may be  
984 classified by its industry, by the products it sells and by its geographical location.

985

986 A general *Classification* scheme can be viewed as a *Classification* tree. In the  
987 example shown in Figure 4, RegistryObject instances representing *Collaboration*  
988 *Protocol Profiles* are shown as shaded boxes. Each *Collaboration Protocol*  
989 *Profile* represents an automobile manufacturer. Each *Collaboration Protocol*  
990 *Profile* is classified by the ClassificationNode named “Automotive” under the  
991 ClassificationScheme instance with name “Industry”. Furthermore, the US  
992 Automobile manufacturers are classified by the US ClassificationNode under the  
OASIS/ebXML Registry Information Model

993 ClassificationScheme with name "Geography". Similarly, a European automobile  
 994 manufacturer is classified by the "Europe" ClassificationNode under the  
 995 ClassificationScheme with name "Geography".

997 The example shows how a RegistryObject may be classified by multiple  
 998 ClassificationNode instances under multiple ClassificationScheme instances (e.g.  
 999 Industry, Geography).

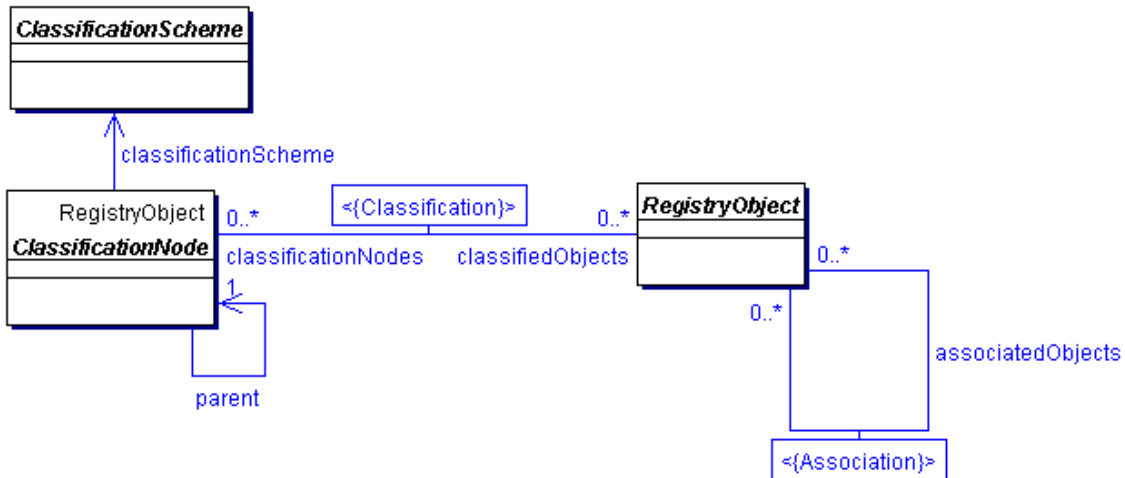


1000  
 1001

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

1002 [Note]It is important to point out that the dark  
 1003 nodes (gasGuzzlerInc, yourDadsCarInc etc.) are  
 1004 not part of the Classification tree. The leaf  
 1005 nodes of the Classification tree are Health  
 1006 Care, Automotive, Retail, US and Europe. The  
 1007 dark nodes are associated with the  
 1008 Classification tree via a Classification  
 1009 Instance that is not shown in the picture  
 1010

1011 In order to support a general Classification scheme that can support single level  
 1012 as well as multi-level Classifications, the information model defines the Classes  
 1013 and relationships shown in Figure 5.

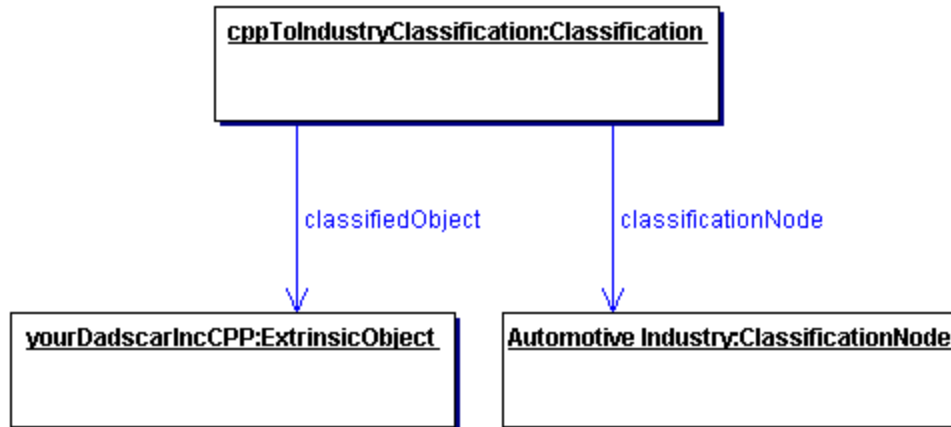


1014

1015

**Figure 5: Information Model Classification View**

1016 A Classification is somewhat like a specialized form of an Association. Figure 6  
 1017 shows an example of an ExtrinsicObject Instance for a Collaboration Protocol  
 1018 Profile (CPP) object that is classified by a ClassificationNode representing the  
 1019 Industry that it belongs to.



1020

1021

**Figure 6: Classification Instance Diagram**

1022 **10.1 Class ClassificationScheme**

1023 A ClassificationScheme instance defines the root of a classification hierarchy  
 1024 where the nodes of the tree are composed of ClassificationNodes instances.  
 1025 Currently the ClassificationScheme class does not define any attributes other  
 1026 than the attributes inherited from RegistryObject and RegistryEntry base classes.

1027  
 1028 <FSN: Do we need any attributes or methods for this class?? Also do we need to  
 1029 change the name of GetRootClassificationNodesRequest/Response to  
 1030 GetClassificationSchemeRequest/Response in the RS 1.1??>

1031 **10.2 Class ClassificationNode**1032 **Base classes:**1033 [RegistryObject](#)

1034

1035 ClassificationNode instances are used to define tree structures where each node  
 1036 in the tree is a ClassificationNode. Such *Classification* trees are constructed with  
 1037 ClassificationNode instances under a ClassificationScheme instance, and are  
 1038 used to define *Classification* schemes or ontologies.

1039 **10.2.1 Attribute Summary**

1040

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

1041

1042 Note that attributes inherited from the base classes of this class are not shown.

1043 **10.2.2 Attribute parent**

1044 Each ClassificationNode must have a parent attribute. The parent attribute either  
 1045 references a parent ClassificationNode or a ClassificationScheme instance in  
 1046 case of first level ClassificationNode instances.

1047 **10.2.3 Attribute code**

1048 Each ClassificationNode may have a code attribute. The code attribute contains  
 1049 a code within a standard coding scheme as described in section 10.5.

1050 **10.2.4 Method Summary**

1051 In addition to its attributes, the Package class also defines the following methods.

1052

| Method Summary of ClassificationNode |   |
|--------------------------------------|---|
| ClassificationScheme                 | <a href="#">getClassificationScheme</a> ()<br>Get the ClassificationScheme that this this ClassificationNode belongs to.  |
| Collection                           | <a href="#">getClassifiedObjects</a> ()<br>Get the collection of RegistryObjects classified by this ClassificationNode.   |
| String                               | <a href="#">getPath</a> ()<br>Gets the path from the ClassificationScheme of this ClassificationNode. The path conforms to the [XPATH] expression syntax (e.g "/Geography/Asia/Japan"). |

1053

1054 Note that methods inherited from the base classes of this class are not shown.

1055

1056 In Figure 4, several instances of ClassificationNode are defined (all light colored  
1057 boxes). A ClassificationNode has exactly one parent and zero or more

1058 ClassificationNodes for its immediate children. The parent of a

1059 ClassificationNodes may be another ClassificationNodes or a

1060 ClassificationScheme in case of first level ClassificationNodes.

## 1061 10.3 Class Classification

1062 **Base Classes:**

1063 [RegistryObject](#)

1064

1065 Classification instances are used to classify repository item by associating their  
1066 RegistryObject *Instance* with a ClassificationNode *Instance* within a *Classification*  
1067 scheme.

1068

1069 In Figure 4, Classification instances are not explicitly shown but are implied as  
1070 associations between the RegistryObject instances (shaded leaf node) and the  
1071 associated ClassificationNode.

### 1072 10.3.1 Attribute Summary

1073

| Attribute          | Data Type | Required | Default Value | Specified By | Mutable |
|--------------------|-----------|----------|---------------|--------------|---------|
| classificationNode | UUID      | Yes      |               | Client       | No      |
| classifiedObject   | UUID      | Yes      |               | Client       | No      |

1074 Note that attributes inherited from the base classes of this class are not shown.

### 1075 10.3.2 Attribute classificationNode

1076 Each Classification instance must have a classificationNode attribute that

1077 references the ClassificationNode instance that is used to classify a

1078 RegistryObject specified by the classifiedObject attribute. This is similar to the

1079 targetObject attribute in an Association instance.

### 1080 10.3.3 Attribute classifiedObject

1081 Each Classification instance must have a classifiedObject attribute that

1082 references the RegistryObject instance that is classified by this Classification.

1083 This is similar to the sourceObject attribute in an Association instance.

### 1084 10.3.4 Inherited Attribute id

1085 The id attribute for a Classification is an attribute based id composed of the

1086 value of the classifiedObject and the classificationNode attributes in that order,

1087 where each attribute value is separated by a ':'.

1088

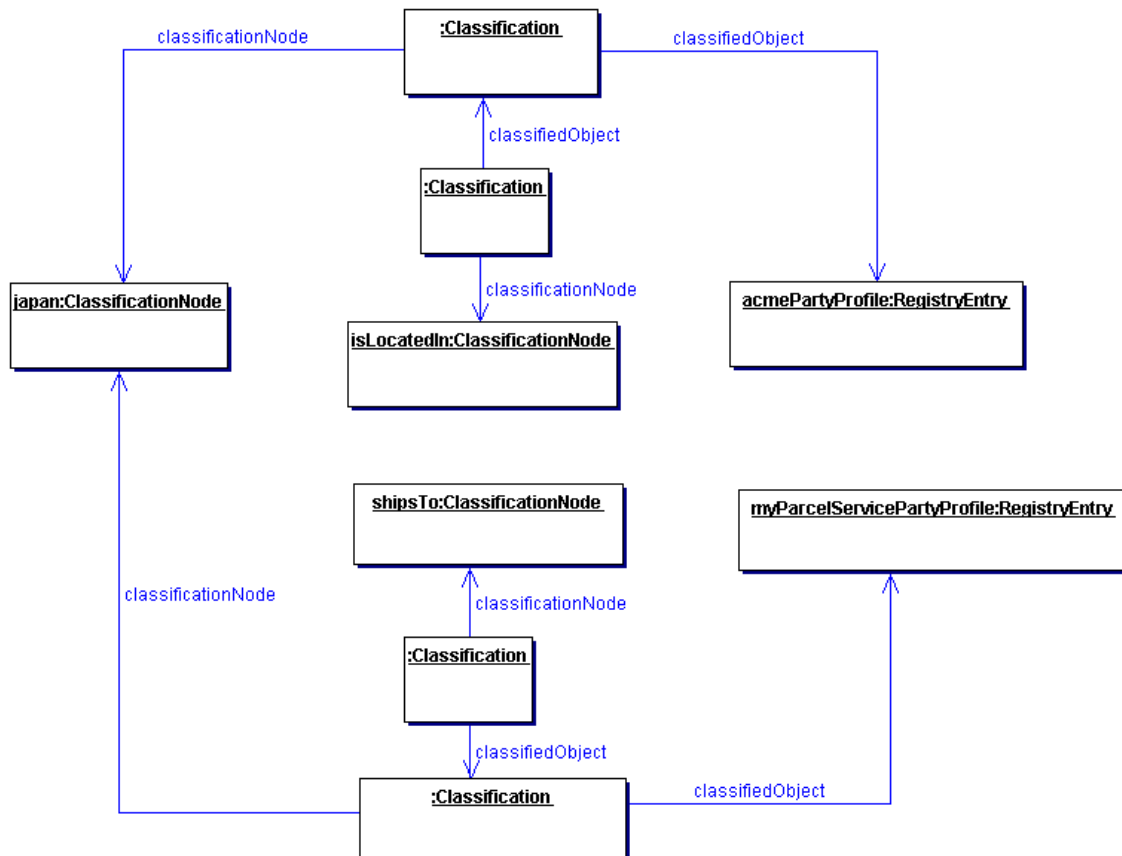
1089 The pattern is as follows:  
 1090 urn:uuid:<classifiedObject id>:< classificationNode id>

1091  
 1092 An example is as follows:

1093 urn:uuid:a2345678-1234-1234-123456789012: a2345678-1234-1234-123456789013  
 1094

1095 **10.3.5 Context Sensitive Classification**

1096 Consider the case depicted in Figure 7 where a *Collaboration Protocol Profile* for  
 1097 ACME Inc. is classified by the Japan ClassificationNode under the Geography  
 1098 *Classification* scheme. In the absence of the context for this *Classification* its  
 1099 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it  
 1100 mean that ACME ships products to Japan, or does it have some other meaning?  
 1101 To address this ambiguity a Classification may optionally be associated with  
 1102 another ClassificationNode (in this example named isLocatedIn) that provides the  
 1103 missing context for the Classification. Another *Collaboration Protocol Profile* for  
 1104 MyParcelService may be classified by the Japan ClassificationNode where this  
 1105 Classification is associated with a different ClassificationNode (e.g. named  
 1106 shipsTo) to indicate a different context than the one used by ACME Inc.



1107  
 1108

Figure 7: Context Sensitive Classification



1109 Thus, in order to support the possibility of Classification within multiple contexts,  
 1110 a Classification is itself classified by any number of Classifications that bind the  
 1111 first Classification to ClassificationNodes that provide the missing contexts.

1112

1113 In summary, the generalized support for *Classification* schemes in the  
 1114 information model allows:

- 1115 ○ A RegistryObject to be classified by defining a Classification that
- 1116 associates it with a ClassificationNode in a *Classification* tree.
- 1117 ○ A RegistryObject to be classified along multiple facets by having multiple
- 1118 *Classifications* that associate it with multiple ClassificationNodes.
- 1119 ○ A *Classification* defined for a RegistryObject to be qualified by the
- 1120 contexts in which it is being classified.

## 1121 10.4 Example of Classification Schemes

1122 The following table lists some examples of possible *Classification* schemes  
 1123 enabled by the information model. These schemes are based on a subset of  
 1124 contextual concepts identified by the ebXML Business Process and Core  
 1125 Components Project Teams. This list is meant to be illustrative not prescriptive.

1126

1127

| <b>Classification Scheme</b> | <b>Usage Example</b>  | <b>Standard Classification Schemes</b> |
|------------------------------|---|--|
| Industry                     | Find all Parties in Automotive industry                         | NAICS                                  |
| Process                      | Find a ServiceInterface that implements a Process               |  |
| Product / Services           | Find a <i>Business</i> that sells a product or offers a service | UNSPSC                                 |
| Locale                       | Find a Supplier located in Japan                                | ISO 3166                               |
| Temporal                     | Find Supplier that can ship with 24 hours                       |  |
| Role                         | Find All Suppliers that have a <i>Role</i> of "Seller"          |  |

1128

**Table 1: Sample Classification Schemes**

## 1129 10.5 Standardized Taxonomy Support

1130 Standardized taxonomies also referred to as ontologies, classification schemes,  
 1131 or coding schemes exist in various industries to provide a structured coded  
 1132 vocabulary. The ebXML *Registry* does not define support for specific taxonomies.  
 1133 Instead it provides a general capability to link RegistryEntries to codes defined by  
 1134 various taxonomies.

1135

1136 The information model provides two alternatives for using standardized  
 1137 taxonomies for *Classification* of RegistryEntries.

**1138 10.5.1 Full-featured Taxonomy Based Classification**

1139 The information model provides a full-featured taxonomy based *Classification*  
1140 alternative based on *Classification*, *ClassificationScheme* and *ClassificationNode*  
1141 instances. This alternative requires that a standard taxonomy be imported into  
1142 the *Registry* as a *Classification* tree consisting of *ClassificationNode* instances  
1143 rooted under a *ClassificationScheme* instance. This specification does not  
1144 prescribe the transformation tools necessary to convert standard taxonomies into  
1145 ebXML *Registry Classification* trees. However, the transformation **MUST** ensure  
1146 that:

- 1147     o The name attribute of the *ClassificationScheme* instance is the *name* of  
1148     the standard taxonomy (e.g. NAICS, ICD-9, SNOMED).
- 1149     o All codes in the standard taxonomy are preserved in the *code* attribute of  
1150     a *ClassificationNode*.
- 1151     o The intended structure of the standard taxonomy is preserved in the  
1152     *ClassificationNode* tree, thus allowing polymorphic browse and drill down  
1153     discovery. This means that when searching for entries classified by Asia, a  
1154     client will find entries classified by descendants of Asia (e.g. Japan and  
1155     Korea).

**1156 10.5.2 Light Weight Taxonomy Based Classification**

1157 <FSN: This section will be reworked based on the classification sub-team  
1158 proposal??>

1159  
1160 The information model also provides a lightweight alternative for classifying  
1161 *RegistryObject* instances by codes defined by standard taxonomies, where the  
1162 submitter does not wish to import an entire taxonomy as a native *Classification*  
1163 scheme.

1164  
1165 In this alternative the submitter adds one or more taxonomy related Slots to the  
1166 *RegistryObject*. Each Slot's name identifies a standardized taxonomy while the  
1167 Slot's value is the code within the specified taxonomy. Such taxonomy related  
1168 Slots **MUST** be defined with a *slotType* of *Classification*.

1169  
1170 For example if a *RegistryObject* has a Slot with name "NAICS", a *slotType* of  
1171 "Classification" and a value "51113" it implies that the *RegistryObject* is classified  
1172 by the code for "Book Publishers" in the NAICS taxonomy. Note that in this  
1173 example, there is no need to import the entire NAICS taxonomy, nor is there any  
1174 need to create instances of *ClassificationNode* or *Classification*.

1175  
1176 The following points are noteworthy in this light weight *Classification* alternative:  
1177     o Validation of the name and the value of the "Classification" is responsibility  
1178     of the SO and not of the ebXML *Registry* itself.  
1179     o Discovery is based on exact match on slot name and slot value rather  
1180     than the flexible "browse and drill down discovery" available to the heavy  
1181     weight *Classification* alternative.

1182

1183 **11 Information Model: Security View**

1184 This section describes the aspects of the information model that relate to the  
1185 security features of the *Registry*.

1186

1187 <FSN: Thos chapter will be updated based on output from the security sub-  
1188 team?? It is therefore not updated in format to the new format for RIM 1.1 >

1189

1190 Figure 8 shows the view of the objects in the *Registry* from a security  
1191 perspective. It shows object relationships as a *UML Class* diagram. It does not  
1192 show *Class* attributes or *Class* methods that will be described in subsequent  
1193 sections. It is meant to be illustrative not prescriptive.

1194

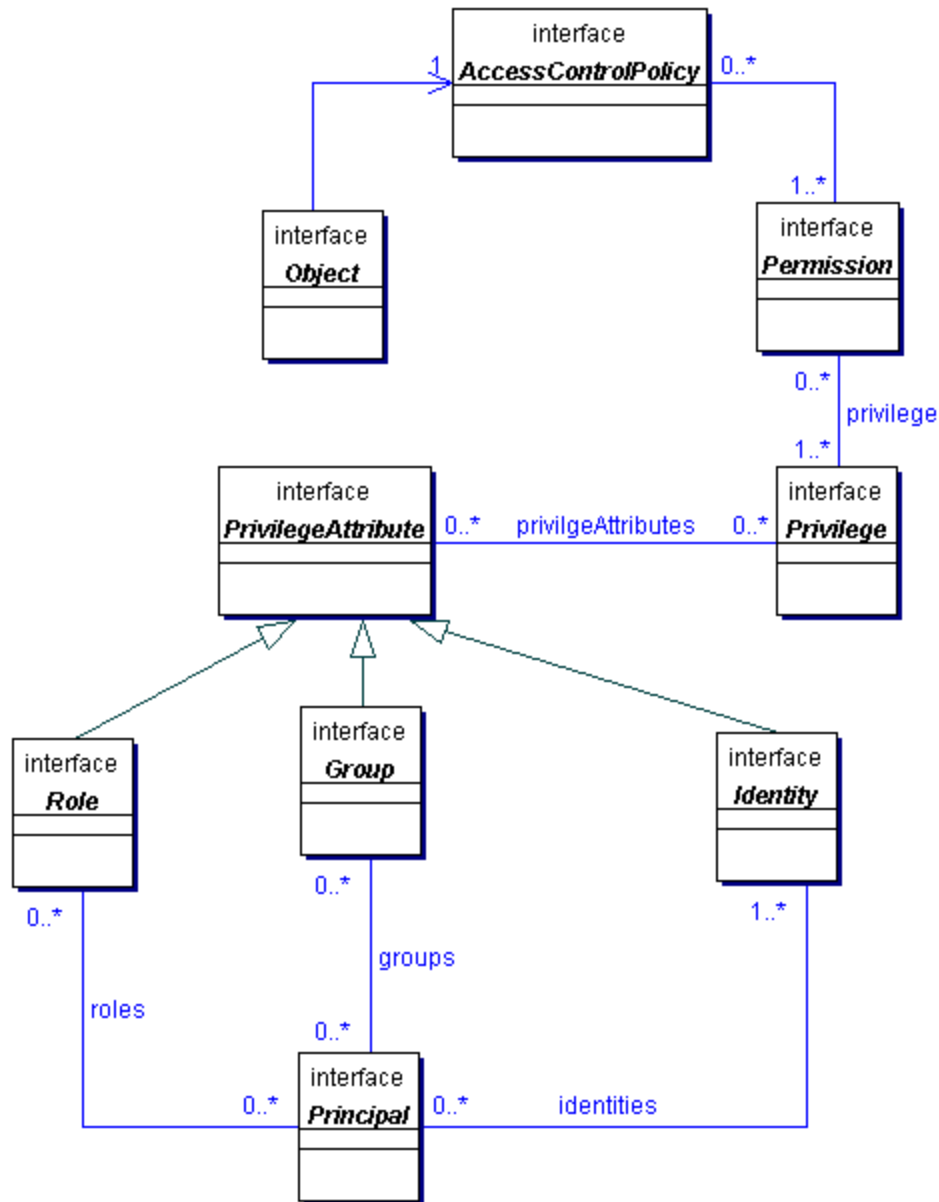


Figure 8: Information Model: Security View

1195  
1196

1197 **11.1 Class AccessControlPolicy**

1198 Every RegistryObject is associated with exactly one AccessControlPolicy which  
 1199 defines the policy rules that govern access to operations or methods performed  
 1200 on that RegistryObject. Such policy rules are defined as a collection of  
 1201 Permissions.

1202  
1203  
1204  
1205

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

1206

## 1207 11.2 Class Permission

1208

1209 The Permission object is used for authorization and access control to  
1210 RegistryObjects in the *Registry*. The Permissions for a RegistryObject are  
1211 defined in an AccessControlPolicy object.

1212

1213 A Permission object authorizes access to a method in a RegistryObject if the  
1214 requesting Principal has any of the Privileges defined in the Permission.

1215 **See Also:**1216 [Privilege](#), [AccessControlPolicy](#)

1217

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

1218

## 1219 11.3 Class Privilege

1220

1221 A Privilege object contains zero or more PrivilegeAttributes. A PrivilegeAttribute  
1222 can be a Group, a Role, or an Identity.

1223

1224 A requesting Principal MUST have all of the PrivilegeAttributes specified in a  
1225 Privilege in order to gain access to a method in a protected RegistryObject.

1226 Permissions defined in the RegistryObject's AccessControlPolicy define the  
1227 Privileges that can authorize access to specific methods.

1228

1229 This mechanism enables the flexibility to have object access control policies that  
1230 are based on any combination of Roles, Identities or Groups.

1231 **See Also:**1232 [PrivilegeAttribute](#), [Permission](#)

1233

1234

1235

| Method Summary of Privilege |  |
|-----------------------------|--|
| Collection                  | <a href="#">getPrivilegeAttributes()</a>   |
|                             | Gets the PrivilegeAttributes associated with this Privilege.<br>Maps to attribute named <code>privilegeAttributes</code> . |

1236

## 1237 **11.4 Class PrivilegeAttribute**

1238 **All Known Subclasses:**

1239 [Group](#), [Identity](#), [Role](#)

1240

1241 PrivilegeAttribute is a common base *Class* for all types of security attributes that  
1242 are used to grant specific access control privileges to a Principal. A Principal may  
1243 have several different types of PrivilegeAttributes. Specific combination of  
1244 PrivilegeAttributes may be defined as a Privilege object.

1245 **See Also:**

1246 [Principal](#), [Privilege](#)

## 1247 **11.5 Class Role**

1248 **All Superclasses:**

1249 [PrivilegeAttribute](#)

1250

1251 A security Role PrivilegeAttribute. For example a hospital may have *Roles* such  
1252 as Nurse, Doctor, Administrator etc. Roles are used to grant Privileges to  
1253 Principals. For example a Doctor *Role* may be allowed to write a prescription but  
1254 a Nurse *Role* may not.

## 1255 **11.6 Class Group**

1256 **All Superclasses:**

1257 [PrivilegeAttribute](#)

1258

1259 A security Group PrivilegeAttribute. A Group is an aggregation of users that may  
1260 have different Roles. For example a hospital may have a Group defined for  
1261 Nurses and Doctors that are participating in a specific clinical trial (e.g.  
1262 AspirinTrial group). Groups are used to grant Privileges to Principals. For  
1263 example the members of the AspirinTrial group may be allowed to write a  
1264 prescription for Aspirin (even though Nurse Role as a rule may not be allowed to  
1265 write prescriptions).

## 1266 **11.7 Class Identity**

1267 **All Superclasses:**

1268 [PrivilegeAttribute](#)

1269

1270 A security Identity PrivilegeAttribute. This is typically used to identify a person, an  
 1271 organization, or software service. Identity attribute may be in the form of a digital  
 1272 certificate.

## 1273 **11.8 Class Principal**

1274  
 1275 Principal is a completely generic term used by the security community to include  
 1276 both people and software systems. The Principal object is an entity that has a set  
 1277 of PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and  
 1278 optionally a set of role memberships, group memberships or security clearances.  
 1279 A principal is used to authenticate a requestor and to authorize the requested  
 1280 action based on the PrivilegeAttributes associated with the Principal.

1281 **See Also:**

1282 PrivilegeAttributes, [Privilege](#), [Permission](#)

1283

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

1284

1285

## 1285 **12 References**

- 1286 [ebGLOSS] ebXML Glossary,  
1287 [http://www.ebxml.org/documents/199909/terms\\_of\\_reference.htm](http://www.ebxml.org/documents/199909/terms_of_reference.htm)
- 1288 [ebTA] ebXML Technical Architecture Specification  
1289 [http://www.ebxml.org/specdrafts/ebXML\\_TA\\_v1.0.4.pdf](http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.4.pdf)
- 1290 [OAS] OASIS Information Model  
1291 <http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf>
- 1292 [ISO] ISO 11179 Information Model  
1293 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument>
- 1294
- 1295 [BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use  
1296 in RFCs to Indicate Requirement Levels  
1297 <http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html>
- 1298 [ebRS] ebXML Registry Services Specification  
1299 [http://www.ebxml.org/specdrafts/ebXML\\_RS\\_v1.0.pdf](http://www.ebxml.org/specdrafts/ebXML_RS_v1.0.pdf)
- 1300 [ebBPSS] ebXML Business Process Specification Schema  
1301 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 1302 [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification  
1303 <http://www.ebxml.org/specdrafts/>
- 1304
- 1305 [UUID] DCE 128 bit Universal Unique Identifier  
1306 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20)  
1307 <http://www.opengroup.org/publications/catalog/c706.htm>  
1308 <http://www.w3.org/TR/REC-xml>
- 1309 [XPATH] XML Path Language (XPath) Version 1.0  
1310 <http://www.w3.org/TR/xpath>  
1311

## 1312 **13 Disclaimer**

- 1313 The views and specification expressed in this document are those of the authors  
1314 and are not necessarily those of their employers. The authors and their  
1315 employers specifically disclaim responsibility for any problems arising from  
1316 correct or incorrect implementation or use of this design.  
1317



1317 **14 Contact Information**

1318

1319 Team Leader

1320 Name: Lisa Carnahan  
1321 Company: NIST  
1322 Street: 100 Bureau Drive STOP 8970  
1323 City, State, Postal Code: Gaithersburg, MD 20899-8970  
1324 Country: USA  
1325 Phone: 301-975-3362  
1326 Email: lisa.carnahan@nist.gov

1327

1328 Editor

1329 Name: Sally Fuger  
1330 Company: <Need Sally's contact info??>  
1331 Street:  
1332 City, State, Postal Code:  
1333 Country: USA  
1334 Phone:  
1335 Email: sfuger@aiag.org

1336

1337 Technical Editor

1338 Name: Farrukh S. Najmi  
1339 Company: Sun Microsystems  
1340 Street: 1 Network Dr., MS BUR02-302  
1341 City, State, Postal Code: Burlington, MA, 01803-0902  
1342 Country: USA  
1343 Phone: 781.442.0703  
1344 Email: najmi@east.sun.com

1345

1346

1346 **Copyright Statement**

1347 Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved

1348

1349 This document and translations of it MAY be copied and furnished to others, and  
1350 derivative works that comment on or otherwise explain it or assist in its  
1351 implementation MAY be prepared, copied, published and distributed, in whole or  
1352 in part, without restriction of any kind, provided that the above copyright notice  
1353 and this paragraph are included on all such copies and derivative works.

1354 However, this document itself MAY not be modified in any way, such as by  
1355 removing the copyright notice or references to ebXML, UN/CEFACT, or OASIS,  
1356 except as required to translate it into languages other than English.

1357

1358 The limited permissions granted above are perpetual and will not be revoked by  
1359 ebXML or its successors or assigns.

1360

1361 This document and the information contained herein is provided on an "AS IS"  
1362 basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,  
1363 INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE  
1364 INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED  
1365 WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR  
1366 PURPOSE.

1367