1 ebXML Registry Information Model v0.90
2 ebXML Registry Project Team
3 20 April 2001

1 Status of this Document

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

Distribution of this document is unlimited.

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

This version: [http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.61.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.61.pdf)

Latest version: [http://www.ebxml.org/project_teams/registry/private/RegistryInfoModel.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModel.pdf)

Previous version: [http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.60.pdf](http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.60.pdf)
2 ebXML participants

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

Lisa Carnahan, NIST
Joe Dalman, Tie
Philippe DeSmedt, Viquity
Sally Fuger, AIAG
Len Gallagher, NIST
Steve Hanna, Sun Microsystems
Scott Hinkelman, IBM
Michael Kass, NIST
Jong.L Kim, Innodigital
Sangwon Lim, Korea Institute for Electronic Commerce
Bob Miller, GXS
Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan
Dale Moberg, Sterling Commerce
Ron Monzillo, Sun Microsystems
JP Morgenthal, eThink Systems, Inc.
Joel Munter, Intel
Farrukh Najmi, Sun Microsystems
Scott Nieman, Norstan Consulting
Frank Olken, Lawrence Berkeley National Laboratory
Michael Park, eSum Technologies
Bruce Peat, eProcess Solutions
Mike Rowley, Excelon Corporation
Waqar Sadiq, Vitria
Krishna Sankar, Cisco Systems Inc.
Kim Tae Soo, Government of Korea
Nikola Stojanovic, Encoda Systems, Inc.
David Webber, XML Global
Yutaka Yoshida, Sun Microsystems
Prasad Yendluri, webmethods
Peter Z. Zhoo, Knowledge For the new Millennium
# Table of Contents

1. **STATUS OF THIS DOCUMENT** ........................................................................... 1
2. **EBXML PARTICIPANTS** .................................................................................... 2
3. **INTRODUCTION** ............................................................................................... 6
   3.1 **SUMMARY OF CONTENTS OF DOCUMENT** ..................................................... 6
   3.2 **GENERAL CONVENTIONS** ............................................................................. 6
      3.2.1 **Naming Conventions** ............................................................................. 7
   3.3 **AUDIENCE** .................................................................................................... 7
   3.4 **RELATED DOCUMENTS** ............................................................................... 7
4. **DESIGN OBJECTIVES** ...................................................................................... 8
   4.1 **GOALS** ....................................................................................................... 8
5. **SYSTEM OVERVIEW** ....................................................................................... 8
   5.1 **ROLE OF EBXML Registry** .......................................................................... 8
   5.2 **REGISTRY SERVICES** .................................................................................. 8
   5.3 **WHAT THE Registry INFORMATION MODEL DOES** .................................... 8
   5.4 **HOW THE Registry INFORMATION MODEL WORKS** .................................. 9
   5.5 **WHERE THE Registry INFORMATION MODEL MAY BE IMPLEMENTED** .......... 9
   5.6 **CONFORMANCE AS AN ebXML Registry** ................................................... 9
6. **Registry INFORMATION MODEL: HIGH LEVEL PUBLIC VIEW** ................. 9
   6.1 **RegistryEntry** ............................................................................................. 10
   6.2 **Slot** ........................................................................................................... 11
   6.3 **Association** ................................................................................................. 11
   6.4 **ExternalIdentifier** ...................................................................................... 11
   6.5 **ExternalLink** ............................................................................................... 11
   6.6 **ClassificationNode** ..................................................................................... 11
   6.7 **Classification** ............................................................................................. 11
   6.8 **Package** ..................................................................................................... 11
   6.9 **AuditableEvent** .......................................................................................... 12
   6.10 **User** ........................................................................................................ 12
   6.11 **PostalAddress** .......................................................................................... 12
   6.12 **Organization** ............................................................................................. 12
7. **Registry INFORMATION MODEL: DETAIL VIEW** ................................... 12
   7.1 **Interface RegistryObject** ............................................................................. 13
   7.2 **Interface Versionable** ................................................................................ 15
   7.3 **Interface RegistryEntry** ............................................................................... 15
      7.3.1 **Pre-defined RegistryEntry Status Types** ............................................... 17
Table of Figures

Figure 1: Information Model Public View .......................................................... 10
Figure 2: Information Model Inheritance View ................................................. 13
Figure 3: Example of Registry Entry Association .............................................. 28
Figure 4: Example showing a Classification Tree ............................................. 32
Figure 5: Information Model Classification View .............................................. 32
Figure 6: Classification Instance Diagram ....................................................... 32
Figure 7: Context Sensitive Classification ......................................................... 35
Figure 8: Information Model: Security View ..................................................... 38

Table of Tables

Table 1: Sample Classification Schemes ............................................................ 36
3 Introduction

3.1 Summary of Contents of Document
This document specifies the information model for the ebXML Registry.

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

3.2 General Conventions

- UML diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific Implementation or methodology requirements.

- Interfaces are often used in UML diagrams. They are used instead of Classes with attributes to provide an abstract definition without implying any specific Implementation. Specifically, they do not imply that objects in the Registry will be accessed directly via these interfaces. Objects in the Registry are accessed via interfaces described in the ebXML Registry Services Specification. Each get method in every interface has an explicit indication of the attribute name that the get method maps to. For example getName method maps to an attribute named name.

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

- The term “RegistryEntry” is used to refer to an object that provides metadata about content Instance (repository item).

- The term “RegistryObject” is used to refer to the base interface in the information model to avoid the confusion with the common term “object”. However, when the term “object” is used to refer to a class or an interface in the information model, it may also mean RegistryObject because almost all classes are descendants of RegistryObject.

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

Software practitioners MAY use this document in combination with other ebXML specification documents when creating ebXML compliant software.
The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

3.2.1 Naming Conventions

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

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

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

3.3 Audience

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

- Implementers of ebXML Registry Services
- Implementers of ebXML Registry Clients

3.4 Related Documents

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

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

4.1 Goals

The goals of this version of the specification are to:

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

5 System Overview

5.1 Role of ebXML Registry

The Registry provides a stable store where content submitted by a Submitting Organization is made persistent. Such content is used to facilitate ebXML-based Business to Business (B2B) partnerships and transactions. Submitted content may be XML schema and documents, process descriptions, Core Components, context descriptions, UML models, information about parties and even software components.

5.2 Registry Services

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

5.3 What the Registry Information Model Does

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

The Registry information model:

- Defines what types of objects are stored in the Registry
- Defines how stored objects are organized in the Registry
- Is based on ebXML metamodels from various working groups
5.4 How the Registry Information Model Works
Implementers of the ebXML Registry MAY use the information model to
determine which Classes to include in their Registry Implementation and what
attributes and methods these Classes may have. They MAY also use it to
determine what sort of database schema their Registry Implementation may
need.

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

5.5 Where the Registry Information Model May Be Implemented
The Registry Information Model MAY be implemented within an ebXML Registry
in form of a relational database schema, object database schema or some other
physical schema. It MAY also be implemented as interfaces and Classes within a
Registry Implementation.

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

6 Registry Information Model: High Level Public View
This section provides a high level public view of the most visible objects in the
Registry.

Figure 1 shows the high level public view of the objects in the Registry and their
relationships as a UML Class Diagram. It does not show Inheritance, Class
attributes or Class methods.
The reader is again reminded that the information model is not modeling actual
repository items.
6.1 RegistryEntry

The central object in the information model is a RegistryEntry. An Instance of RegistryEntry exists for each content Instance submitted to the Registry. Instances of the RegistryEntry Class provide metadata about a repository item in the Registry. The actual repository item (e.g. a DTD) is not contained in an Instance of the RegistryEntry Class. Note that most Classes in the information model are specialized sub-classes of RegistryEntry. Each RegistryEntry is related to exactly one repository item, however, in the future revision of this document, it may be related to multiple repository items.
6.2 Slot

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

6.3 Association

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

6.4 ExternalIdentifier

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

6.5 ExternalLink

ExternalLink Instances are RegistryEntries that model a named URI to content that is not managed by the Registry. Unlike managed content, such external content may change or be deleted at any time without the knowledge of the Registry. RegistryEntry may be associated with any number of ExternalLinks.

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

6.6 ClassificationNode

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

6.7 Classification

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

6.8 Package

Package Instances are RegistryEntries that group logically related RegistryEntries together. One use of a Package is to allow operations to be performed on an entire Package of objects. For example all objects belonging to a Package may be deleted in a single request.
6.9 AuditableEvent
AuditableEvent Instances are Objects that are used to provide an audit trail for RegistryEntries. AuditableEvent is described in detail in section 8.

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

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

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

7 Registry Information Model: Detail View
This section covers the information model Classes in more detail than the Public View. The detail view introduces some additional Classes within the model that were not described in the public view of the information model.

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

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

The reader is again reminded that the information model is not modeling actual repository items.
7.1 Interface RegistryObject

All Known Subinterfaces:

Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, RegistryEntry, Organization, Package, Submission

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getAccessControlPolicy()</strong></td>
<td>Gets the AccessControlPolicy object associated with this RegistryObject. An AccessControlPolicy defines the Security Model associated with the RegistryObject in terms of “who is permitted to do what” with that RegistryObject. Maps to attribute named accessControlPolicy.</td>
</tr>
<tr>
<td><strong>getDescription()</strong></td>
<td>Gets the context independent textual description for this RegistryObject. Maps to attribute named description.</td>
</tr>
<tr>
<td><strong>getName()</strong></td>
<td>Gets user friendly context independent name of object in Repository. Maps to attribute named name.</td>
</tr>
<tr>
<td><strong>getID()</strong></td>
<td>Gets the universally unique ID, as defined by [UUID], for this RegistryObject. Maps to attribute named id.</td>
</tr>
<tr>
<td><strong>setDescription(String description)</strong></td>
<td>Sets the context independent textual description for this RegistryObject.</td>
</tr>
<tr>
<td><strong>setName(String name)</strong></td>
<td>Sets user friendly context independent name of object in Repository.</td>
</tr>
<tr>
<td><strong>setID(String id)</strong></td>
<td>Sets the universally unique ID, as defined by [UUID], for this RegistryObject.</td>
</tr>
</tbody>
</table>
7.2 Interface Versionable

All Known Subinterfaces:

Association, Classification, ClassificationNode, ExternalLink,
ExtrinsicObject, IntrinsicObject, RegistryEntry, Organization, Package

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

Method Summary

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int getMajorVersion()</td>
<td>Gets the major revision number for this version of the Versionable object. Maps to attribute named majorVersion.</td>
</tr>
<tr>
<td>int getMinorVersion()</td>
<td>Gets the minor revision number for this version of the Versionable object. Maps to attribute named minorVersion.</td>
</tr>
<tr>
<td>void setMajorVersion(int majorVersion)</td>
<td>Gets the major revision number for this version of the Versionable object.</td>
</tr>
<tr>
<td>void setMinorVersion(int minorVersion)</td>
<td>Sets the minor revision number for this version of the Versionable object.</td>
</tr>
</tbody>
</table>
```

7.3 Interface RegistryEntry

All Superinterfaces:

RegistryObject, Versionable

All Known Subinterfaces:

Association, Classification, ClassificationNode, ExternalLink,
ExtrinsicObject, IntrinsicObject, Organization, Package

RegistryEntry is a common base Class for all metadata describing submitted content whose life cycle is managed by the Registry. Metadata describing content submitted to the Registry is further specialized by the ExtrinsicObject and IntrinsicObject subclasses of RegistryEntry.
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Collection getAssociatedObjects()</code></td>
<td>Returns the collection of RegistryObjects associated with this RegistryObject. Maps to attribute named <code>associatedObjects</code>.</td>
</tr>
<tr>
<td><code>Collection getAuditTrail()</code></td>
<td>Returns the complete audit trail of all requests that effected a state change in this RegistryObject as an ordered Collection of AuditableEvent objects. Maps to attribute named <code>auditTrail</code>.</td>
</tr>
<tr>
<td><code>Collection getClassificationNodes()</code></td>
<td>Returns the collection of ClassificationNodes associated with this RegistryObject. Maps to attribute named <code>classificationNodes</code>.</td>
</tr>
<tr>
<td><code>Collection getExternalLinks()</code></td>
<td>Returns the collection of ExternalLinks associated with this RegistryObject. Maps to attribute named <code>externalLinks</code>.</td>
</tr>
<tr>
<td><code>Collection getExternalIdentifiers()</code></td>
<td>Returns the collection of ExternalIdentifiers associated with this RegistryObject. Maps to attribute named <code>externalIdentifiers</code>.</td>
</tr>
<tr>
<td><code>String getObjectType()</code></td>
<td>Gets the pre-defined object type associated with this RegistryEntry. This SHOULD be the name of a object type as described in 7.3.2. Maps to attribute named <code>objectType</code>.</td>
</tr>
<tr>
<td><code>Collection getOrganizations()</code></td>
<td>Returns the collection of Organizations associated with this RegistryObject. Maps to attribute named <code>organizations</code>.</td>
</tr>
<tr>
<td><code>Collection getPackages()</code></td>
<td>Returns the collection of Packages associated with this RegistryObject. Maps to attribute named <code>packages</code>.</td>
</tr>
<tr>
<td><code>String getStatus()</code></td>
<td>Gets the life cycle status of the RegistryEntry within the Registry. This SHOULD be the name of a RegistryEntry status type as described in 7.3.1. Maps to attribute named <code>status</code>.</td>
</tr>
<tr>
<td><code>String getUserVersion()</code></td>
<td>Gets the userVersion attribute of the RegistryEntry within the Registry. The userVersion is the version for the RegistryEntry as assigned by the user.</td>
</tr>
<tr>
<td><code>void setUserVersion(String UserVersion)</code></td>
<td>Sets the userVersion attribute of the RegistryEntry within the Registry.</td>
</tr>
<tr>
<td><code>String getStability()</code></td>
<td></td>
</tr>
</tbody>
</table>
Gets the stability indicator for the RegistryEntry within the Registry. The stability indicator is provided by the submitter as a guarantee of the level of stability for the content. This SHOULD be the name of a stability type as described in 7.3.3. Maps to attribute named \texttt{stability}.

```java
Date getExpirationDate()
```

 Gets expirationDate attribute of the RegistryEntry within the Registry. This attribute defines a time limit upon the stability guarantee provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes \texttt{STABILITY\_DYNAMIC} implying that content can change at any time and in any manner. A null value implies that there is no expiration on stability attribute. Maps to attribute named \texttt{expirationDate}.

```java
void setExpirationDate(Date ExpirationDate)
```

 Sets expirationDate attribute of the RegistryEntry within the Registry.

```java
Collection getSlots()
```

 Gets the collection of slots that have been dynamically added to this RegistryObject. Maps to attribute named \texttt{slots}.

```java
void addSlots(Collection newSlots)
```

 Adds one or more slots to this RegistryObject. Slot names MUST be locally unique within this RegistryObject. Any existing slots are not effected.

```java
void removeSlots(Collection slotNames)
```

 Removes one or more slots from this RegistryObject. Slots to be removed are identified by their name.

### 420

**Methods inherited from interface RegistryObject**

- \texttt{getAccessControlPolicy}, \texttt{getDescription}, \texttt{getName}, \texttt{getID}, \texttt{setDescription}, \texttt{setName}, \texttt{setID}

### 421

**Methods inherited from interface Versionable**

- \texttt{getMajorVersion}, \texttt{getMinorVersion}, \texttt{setMajorVersion}, \texttt{setMinorVersion}

### 7.3.1 Pre-defined RegistryEntry Status Types

The following table lists pre-defined choices for RegistryEntry status attribute. These pre-defined status types are defined as a \textit{Classification} scheme. While the scheme may easily be extended, a Registry MUST support the status types listed below.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submitted</td>
<td>Status of a RegistryEntry that catalogues content that has been submitted to the Registry.</td>
</tr>
<tr>
<td>Approved</td>
<td>Status of a RegistryEntry that catalogues content that has been submitted to the Registry and has been subsequently approved.</td>
</tr>
<tr>
<td>Deprecated</td>
<td>Status of a RegistryEntry that catalogues content that has been submitted to the Registry and has been subsequently deprecated.</td>
</tr>
<tr>
<td>Withdrawn</td>
<td>Status of a RegistryEntry that catalogues content that has been withdrawn from the Registry.</td>
</tr>
</tbody>
</table>

### 7.3.2 Pre-defined Object Types

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

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

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown</td>
<td>An ExtrinsicObject that catalogues content whose type is unspecified or unknown.</td>
</tr>
<tr>
<td>CPA</td>
<td>An ExtrinsicObject of this type catalogues an XML document <strong>Collaboration Protocol Agreement (CPA)</strong> representing a technical agreement between two parties on how they plan to communicate with each other using a specific protocol.</td>
</tr>
<tr>
<td>CPP</td>
<td>An ExtrinsicObject of this type catalogues an document called <strong>Collaboration Protocol Profile (CPP)</strong> that provides information about a Party participating in a Business transaction.</td>
</tr>
<tr>
<td>Process</td>
<td>An ExtrinsicObject of this type catalogues a process description document.</td>
</tr>
<tr>
<td>Role</td>
<td>An ExtrinsicObject of this type catalogues an XML description of a <strong>Role</strong> in a <strong>Collaboration Protocol Profile (CPP)</strong>.</td>
</tr>
<tr>
<td>ServiceInterface</td>
<td>An ExtrinsicObject of this type catalogues an XML description of a service interface as defined by [ebCPP].</td>
</tr>
</tbody>
</table>
SoftwareComponent
An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or Class library).

Transport
An ExtrinsicObject of this type catalogues an XML description of a transport configuration as defined by [ebCPP].

UMLModel
An ExtrinsicObject of this type catalogues a UML model.

XMLSchema
An ExtrinsicObject of this type catalogues an XML schema (DTD, XML Schema, RELAX grammar, etc.).

Package
A Package object

ExternalLink
An ExternalLink object

ExternalIdentifier
An ExternalIdentifier object

Association
An Association object

Classification
A Classification object

ClassificationNode
A ClassificationNode object

AuditableEvent
An AuditableEvent object

User
A User object

Organization
An Organization object

---

7.3.3 Pre-defined RegistryEntry Stability Enumerations

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Stability of a RegistryEntry that indicates that the content is dynamic and may be changed arbitrarily by submitter at any time.</td>
</tr>
<tr>
<td>DynamicCompatible</td>
<td>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.</td>
</tr>
<tr>
<td>Static</td>
<td>Stability of a RegistryEntry that indicates that the content is static and will not be changed by submitter.</td>
</tr>
</tbody>
</table>

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

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

### Method Summary

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

### 7.5 Interface ExtrinsicObject

**All Superinterfaces:**

- `RegistryEntry`
- `RegistryObject`
- `Versionable`

ExtrinsicObjects provide metadata that describes submitted content whose type is not intrinsically known to the Registry and therefore MUST be described by means of additional attributes (e.g., mime type).
Examples of content described by ExtrinsicObject include *Collaboration Protocol Profiles (CPP), Business Process descriptions,* and schemas.

### Method Summary

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

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

### 7.6 Interface IntrinsicObject

**All Superinterfaces:**
```
RegistryEntry, RegistryObject, Versionable
```

**All Known Subinterfaces:**
```
Association, Classification, ClassificationNode, ExternalLink, Organization, Package
```

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

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

All Superinterfaces:

- IntrinsicObject
- RegistryEntry
- RegistryObject
- Versionable

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

Method Summary

Collection getMemberObjects()

Get the collection of RegistryEntries that are members of this Package. Maps to attribute named memberObjects.

7.8 Interface ExternalIdentifier

All Superinterfaces:

- IntrinsicObject
- RegistryEntry
- RegistryObject
- Versionable

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

See Also:

Method Summary

String getValue()

Gets the value of this ExternalIdentifier. Maps to attribute named value.

Void setValue(String value)

Sets the value of this ExternalIdentifier.

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

7.9 Interface ExternalLink

All Superinterfaces:

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

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection getLinkedObjects()</td>
<td>Gets the collection of RegistryObjects that use this external link. Maps to attribute named linkedObjects.</td>
</tr>
<tr>
<td>URI getExternalURI()</td>
<td>Gets URI to the external content. Maps to attribute named externalURI.</td>
</tr>
<tr>
<td>void setExternalURI(URI uri)</td>
<td>Sets URI to the external content.</td>
</tr>
</tbody>
</table>

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

### 8 Registry Audit Trail

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

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

### 8.1 Interface AuditableEvent

**All Superinterfaces:**

<table>
<thead>
<tr>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryObject</td>
</tr>
</tbody>
</table>

AuditableEvent *Instances* provide a long-term record of *Events* that effect a change of state in a RegistryEntry. A RegistryEntry is associated with an ordered Collection of AuditableEvent *Instances* that provide a complete audit trail for that RegistryObject.
AuditableEvents are usually a result of a client-initiated request. AuditableEvent Instances are generated by the Registry Service to log such Events.

Often such Events effect a change in the life cycle of a RegistryEntry. For example a client request could Create, Update, Deprecate or Delete a RegistryEntry. No AuditableEvent is created for requests that do not alter the state of a RegistryEntry. Specifically, read-only requests do not generate an AuditableEvent. No AuditableEvent is generated for a RegistryEntry when it is classified, assigned to a Package or associated with another RegistryObject.

8.1.1 Pre-defined Auditable Event Types

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>An Event that created a RegistryEntry.</td>
</tr>
<tr>
<td>Deleted</td>
<td>An Event that deleted a RegistryEntry.</td>
</tr>
<tr>
<td>Deprecated</td>
<td>An Event that deprecated a RegistryEntry.</td>
</tr>
<tr>
<td>Updated</td>
<td>An Event that updated the state of a RegistryEntry.</td>
</tr>
<tr>
<td>Versioned</td>
<td>An Event that versioned a RegistryEntry.</td>
</tr>
</tbody>
</table>

Method Summary

- getUser(): Gets the User that sent the request that generated this Event. Maps to attribute named user.
- getEventType(): The type of this Event as defined by the name attribute of an event type as defined in section 8.1.1. Maps to attribute named eventType.
- getRegistryEntry(): Gets the RegistryEntry associated with this AuditableEvent. Maps to attribute named registryEntry.
- getTimestamp(): Gets the timestamp for when this Event occurred. Maps to attribute named timestamp.
Note that methods inherited from the base interfaces of this interface are not shown.

### 8.2 Interface User

All Superinterfaces:
- `RegistryObject`

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

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getOrganization()</code></td>
<td>Gets the <em>Submitting Organization</em> that sent the request that effected this change. Maps to attribute named <code>organization</code>.</td>
</tr>
<tr>
<td><code>getAddress()</code></td>
<td>Gets the postal address for this user. Maps to attribute named <code>address</code>.</td>
</tr>
<tr>
<td><code>getEmail()</code></td>
<td>Gets the email address for this user. Maps to attribute named <code>email</code>.</td>
</tr>
<tr>
<td><code>getFax()</code></td>
<td>The FAX number for this user. Maps to attribute named <code>fax</code>.</td>
</tr>
<tr>
<td><code>getMobilePhone()</code></td>
<td>The mobile telephone number for this user. Maps to attribute named <code>mobilePhone</code>.</td>
</tr>
<tr>
<td><code>getName()</code></td>
<td>Name of contact person. Maps to attribute named <code>name</code>.</td>
</tr>
<tr>
<td><code>getPager()</code></td>
<td>The pager telephone number for this user. Maps to attribute named <code>pager</code>.</td>
</tr>
<tr>
<td><code>getTelephone()</code></td>
<td>The default (land line) telephone number for this user. Maps to attribute named <code>telephone</code>.</td>
</tr>
<tr>
<td><code>getUrl()</code></td>
<td>The <em>URL</em> to the web page for this contact. Maps to attribute named <code>url</code>.</td>
</tr>
</tbody>
</table>
8.3 Interface Organization

All Superinterfaces:

| IntrinsicObject | RegistryEntry | RegistryObject | Versionable |

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

Method Summary

```java
PostalAddress getAddress()
Gets the PostalAddress for this Organization. Maps to attribute named address.

User getPrimaryContact()
Gets the primary Contact for this Organization. The primary contact is a reference to a User object. Maps to attribute named primaryContact.

TelephoneNumber getFax()
Gets the FAX number for this Organization. Maps to attribute named fax.

Organization getParent()
Gets the parent Organization for this Organization. Maps to attribute named parent.

TelephoneNumber getTelephone()
Gets the main telephone number for this Organization. Maps to attribute named telephone.
```

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

8.4 Class PostalAddress

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

Field Summary

```java
String city
The city.

String country
The country.
```
String **postalCode**

The postal or zip code.

String **state**

The state or province.

String **street**

The street.

### 8.5 Class TelephoneNumber

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

#### Field Summary

- **String areaCode**
  - Area code.
- **String countryCode**
  - Country code.
- **String extension**
  - Internal extension if any.
- **String number**
  - The telephone number suffix not including the country or area code.
- **String url**
  - A URL that can dial this number electronically.

### 8.6 Class PersonName

A simple *Entity Class* for a person’s name.

#### Field Summary

- **String firstName**
  - The first name for this person.
- **String lastName**
  - The last name (surname) for this person.
- **String middleName**
  - The middle name for this person.
9 RegistryEntry Naming

A RegistryEntry has a name that may or may not be unique within the Registry. In addition a RegistryEntry may have any number of context sensitive alternate names that are valid only in the context of a particular Classification scheme. Alternate contextual naming will be addressed in a later version of the Registry Information Model.

10 Association of RegistryEntry

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

![Figure 3: Example of RegistryEntry Association](image)

10.1 Interface Association

All Superinterfaces:

- IntrinsicObject
- RegistryEntry
- RegistryObject
- Versionable

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

An Instance of the Association Class represents an association between two RegistryObjects.
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String <code>getAssociationType()</code></td>
<td>Gets the association type for this Association. This MUST be the name attribute of an association type as defined by 10.1.1. Maps to attribute named <code>associationType</code>.</td>
</tr>
<tr>
<td>Object <code>getSourceObject()</code></td>
<td>Gets the RegistryObject that is the source of this Association. Maps to attribute named <code>sourceObject</code>.</td>
</tr>
<tr>
<td>String <code>getSourceRole()</code></td>
<td>Gets the name of the Role played by the source RegistryObject in this Association. Maps to attribute named <code>sourceRole</code>.</td>
</tr>
<tr>
<td>Object <code>getTargetObject()</code></td>
<td>Gets the RegistryObject that is the target of this Association. Maps to attribute named <code>targetObject</code>.</td>
</tr>
<tr>
<td>String <code>getTargetRole()</code></td>
<td>Gets the name of the Role played by the target RegistryObject in this Association. Maps to attribute named <code>targetRole</code>.</td>
</tr>
<tr>
<td>boolean <code>isBidirectional()</code></td>
<td>Determine whether this Association is bi-directional. Maps to attribute named <code>bidirectional</code>.</td>
</tr>
<tr>
<td>void <code>setBidirectional(boolean bidirectional)</code></td>
<td>Set whether this Association is bi-directional.</td>
</tr>
<tr>
<td>void <code>setSourceRole(String sourceRole)</code></td>
<td>Sets the name of the Role played by the source RegistryObject in this Association.</td>
</tr>
<tr>
<td>void <code>setTargetRole(String targetRole)</code></td>
<td>Sets the name of the Role played by the destination RegistryObject in this Association.</td>
</tr>
</tbody>
</table>

### 10.1.1 Pre-defined Association Types

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.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
</tr>
</thead>
</table>

---

Copyright © ebXML 2000 & 2001. All Rights Reserved.
<table>
<thead>
<tr>
<th>RelatedTo</th>
<th>Defines that source RegistryObject is related to target RegistryObject.</th>
</tr>
</thead>
<tbody>
<tr>
<td>HasMember</td>
<td>Defines that the source Package object has the target RegistryEntry object as a member. Reserved for use in Packaging of RegistryEntries.</td>
</tr>
<tr>
<td>ExternallyLinks</td>
<td>Defines that the source ExternalLink object externally links the target RegistryEntry object. Reserved for use in associating ExternalLinks with RegistryEntries.</td>
</tr>
<tr>
<td>ExternallyIdentifies</td>
<td>Defines that the source ExternalIdentifier object identifies the target RegistryEntry object. Reserved for use in associating ExternalIdentifiers with RegistryEntries.</td>
</tr>
<tr>
<td>ContainedBy</td>
<td>Defines that source RegistryObject is contained by the target RegistryObject.</td>
</tr>
<tr>
<td>Contains</td>
<td>Defines that source RegistryObject contains the target RegistryObject.</td>
</tr>
<tr>
<td>Extends</td>
<td>Defines that source RegistryObject inherits from or specializes the target RegistryObject.</td>
</tr>
<tr>
<td>Implements</td>
<td>Defines that source RegistryObject implements the functionality defined by the target RegistryObject.</td>
</tr>
<tr>
<td>InstanceOf</td>
<td>Defines that source RegistryObject is an Instance of target RegistryObject.</td>
</tr>
<tr>
<td>SupersededBy</td>
<td>Defines that the source RegistryObject is superseded by the target RegistryObject.</td>
</tr>
<tr>
<td>Supersedes</td>
<td>Defines that the source RegistryObject supersedes the target RegistryObject.</td>
</tr>
<tr>
<td>UsedBy</td>
<td>Defines that the source RegistryObject is used by the target RegistryObject in some manner.</td>
</tr>
<tr>
<td>Uses</td>
<td>Defines that the source RegistryObject uses the target RegistryObject in some manner.</td>
</tr>
<tr>
<td>ReplacedBy</td>
<td>Defines that the source RegistryObject is replaced by the target RegistryObject in some manner.</td>
</tr>
<tr>
<td>Replaces</td>
<td>Defines that the source RegistryObject replaces the target RegistryObject in some manner.</td>
</tr>
</tbody>
</table>

[Note] In some association types, such as Extends and Implements, although the association is between RegistryObjects, the actual relationship specified by that type is between repository items pointed by RegistryObjects.
11 Classification of RegistryEntry

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

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

A general Classification scheme can be viewed as a Classification tree. In the example shown in Figure 4, RegistryEntries representing Collaboration Protocol Profiles are shown as shaded boxes. Each Collaboration Protocol Profile represents an automobile manufacturer. Each Collaboration Protocol Profile is classified by the ClassificationNode named Automotive under the root ClassificationNode named Industry. Furthermore, the US Automobile manufacturers are classified by the US ClassificationNode under the Geography ClassificationNode. Similarly, a European automobile manufacturer is classified by the Europe ClassificationNode under the Geography ClassificationNode.

The example shows how a RegistryEntry may be classified by multiple Classification schemes. A Classification scheme is defined by a ClassificationNode that is the root of a Classification tree (e.g. Industry, Geography).
Figure 4: Example showing a Classification Tree

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

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

Figure 5: Information Model Classification View

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

Figure 6: Classification Instance Diagram
## 11.1 Interface ClassificationNode

### All Superinterfaces:
- `IntrinsicObject`, `RegistryEntry`, `RegistryObject`, `Versionable`

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

**See Also:**
- `classification`

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `Collection getClassifiedObjects()` | Get the collection of RegistryEntries classified by this ClassificationNode. Maps to attribute named `classifiedObjects`.
| `ClassificationNode getParent()` | Gets the parent ClassificationNode for this ClassificationNode. Maps to attribute named `parent`.
| `String getPath()` | Gets the path from the root ancestor of this ClassificationNode. The path conforms to the [XPATH] expression syntax (e.g. “/Geography/Asia/Japan”). Maps to attribute named `path`.
| `void setParent(ClassificationNode parent)` | Sets the parent ClassificationNode for this ClassificationNode.
| `String getCode()` | Gets the code for this ClassificationNode. See section 11.4 for details. Maps to attribute named `code`.
| `void setCode(String code)` | Sets the code for this ClassificationNode. See section 11.4 for details.

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

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

#### All Superinterfaces:
- `IntrinsicObject`, `RegistryEntry`, `RegistryObject`, `Versionable`

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

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

### Method Summary

- **`RegistryObject getClassifiedObject()`**
  - Gets the `RegistryObject` that is classified by this Classification. Maps to attribute named `classifiedObject`.

- **`RegistryObject getClassificationNode()`**
  - Gets the `ClassificationNode` that classifies the `RegistryObject` in this Classification. Maps to attribute named `classificationNode`.

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

#### 11.2.1 Context Sensitive Classification

Consider the case depicted in Figure 7 where a `Collaboration Protocol Profile` for ACME Inc. is classified by the Japan `ClassificationNode` under the Geography `Classification` scheme. In the absence of the context for this `Classification` its meaning is ambiguous. Does it mean that ACME is located in Japan, or does it mean that ACME ships products to Japan, or does it have some other meaning? To address this ambiguity a Classification may optionally be associated with another `ClassificationNode` (in this example named `isLocatedIn`) that provides the missing context for the Classification. Another `Collaboration Protocol Profile` for MyParcelService may be classified by the Japan `ClassificationNode` where this Classification is associated with a different `ClassificationNode` (e.g. named `shipsTo`) to indicate a different context than the one used by ACME Inc.
Thus, in order to support the possibility of Classification within multiple contexts, a Classification is itself classified by any number of Classifications that bind the first Classification to ClassificationNodes that provide the missing contexts.

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

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

11.3 Example of Classification Schemes

The following table lists some examples of possible Classification schemes enabled by the information model. These schemes are based on a subset of contextual concepts identified by the ebXML Business Process and Core Components Project Teams. This list is meant to be illustrative not prescriptive.
<table>
<thead>
<tr>
<th>Classification Scheme (Context)</th>
<th>Usage Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry</td>
<td>Find all Parties in Automotive industry</td>
</tr>
<tr>
<td>Process</td>
<td>Find a ServiceInterface that implements a Process</td>
</tr>
<tr>
<td>Product</td>
<td>Find a Business that sells a product</td>
</tr>
<tr>
<td>Locale</td>
<td>Find a Supplier located in Japan</td>
</tr>
<tr>
<td>Temporal</td>
<td>Find Supplier that can ship with 24 hours</td>
</tr>
<tr>
<td>Role</td>
<td>Find All Suppliers that have a Role of “Seller”</td>
</tr>
</tbody>
</table>

Table 1: Sample Classification Schemes

11.4 Standardized Taxonomy Support

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

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

11.4.1 Full-featured Taxonomy Based Classification

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

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

11.4.2 Light Weight Taxonomy Based Classification

The information model also provides a lightweight alternative for classifying RegistryEntry Instances by codes defined by standard taxonomies, where the submitter does not wish to import an entire taxonomy as a native Classification scheme.
In this alternative the submitter adds one or more taxonomy related Slots to the RegistryEntry for a submitted repository item. Each Slot’s name identifies a standardized taxonomy while the Slot’s value is the code within the specified taxonomy. Such taxonomy related Slots MUST be defined with a slotType of Classification.

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

The following points are noteworthy in this light weight Classification alternative:

- Validation of the name and the value of the Classification" is responsibility of the SO and not of the ebXML Registry itself.
- Discovery is based on exact match on slot name and slot value rather than the flexible “browse and drill down discovery” available to the heavy weight Classification alternative.

12 Information Model: Security View

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

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

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

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

12.2 Interface Permission

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

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

See Also:
privilege, AccessControlPolicy

Method Summary

String <getMethodName()> Gets the method name that is accessible to a Principal with specified Privilege by this Permission. Maps to attribute named methodName.

Collection <getPrivileges()> Gets the Privileges associated with this Permission. Maps to attribute named privileges.

12.3 Interface Privilege

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

A requesting Principal MUST have all of the PrivilegeAttributes specified in a Privilege in order to gain access to a method in a protected RegistryObject. Permissions defined in the RegistryObject's AccessControlPolicy define the Privileges that can authorize access to specific methods.

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

See Also:
PrivilegeAttribute, Permission
Method Summary

<table>
<thead>
<tr>
<th>Collection</th>
<th>getPrivilegeAttributes()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the PrivilegeAttributes associated with this Privilege. Maps to attribute named privilegeAttributes.</td>
<td></td>
</tr>
</tbody>
</table>

12.4 Interface PrivilegeAttribute

All Known Subinterfaces:

- Group
- Identity
- Role

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

See Also:

- Principal
- Privilege

12.5 Interface Role

All Superinterfaces:

- PrivilegeAttribute

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

12.6 Interface Group

All Superinterfaces:

- PrivilegeAttribute

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

All Superinterfaces:

PrivilegeAttribute

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

12.8 Interface Principal

Principal is a completely generic term used by the security community to include both people and software systems. The Principal object is an entity that has a set of PrivilegeAttributes. These PrivilegeAttributes include at least one identity, and optionally a set of role memberships, group memberships or security clearances.

A principal is used to authenticate a requestor and to authorize the requested action based on the PrivilegeAttributes associated with the Principal.

See Also:
PrivilegeAttributes, Privilege, Permission

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getGroups()</strong></td>
</tr>
<tr>
<td><strong>getIdentities()</strong></td>
</tr>
<tr>
<td><strong>getRoles()</strong></td>
</tr>
</tbody>
</table>
13 References


[UUID] DCE 128 bit Universal Unique Identifier http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
[XPATH] XML Path Language (XPath) Version 1.0 http://www.w3.org/TR/xpath

14 Disclaimer

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

Team Leader
Name:    Scott Nieman
Company:    Norstan Consulting
Street:    5101 Shady Oak Road
City, State, Postal Code:  Minnetonka, MN 55343
Country:    USA
Phone:     952.352.5889
Email:    Scott.Nieman@Norstan

Vice Team Lead
Name:    Yutaka Yoshida
Company:    Sun Microsystems
Street:    901 San Antonio Road, MS UMPK17-102
City, State, Postal Code:  Palo Alto, CA 94303
Country:    USA
Phone:    650.786.5488
Email:    Yutaka.Yoshida@eng.sun.com

Editor
Name:    Farrukh S. Najmi
Company:    Sun Microsystems
Street:    1 Network Dr., MS BUR02-302
City, State, Postal Code:  Burlington, MA, 01803-0902
Country:    USA
Phone:    781.442.0703
Email:    najmi@east.sun.com
Copyright Statement

Copyright © ebXML 2000 & 2001 All Rights Reserved.

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

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

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

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