ebXML Registry Information Model

ebXML Registry Project Team

Working Draft 3/19/2001

This version: Version 0.60

1 Status of this Document

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

Distribution of this document is unlimited.

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

This version:
http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.60.pdf

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

Previous version:
http://www.ebxml.org/project_teams/registry/private/RegistryInfoModelv0.59.pdf

Copyright © ebXML 2000 & 2001. All Rights Reserved.
2 ebXML participants

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

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 Figures

- Figure 1: Information Model Public View.......................................................... 9
- Figure 2: Information Model Inheritance View............................................... 12
- Figure 3: Example of Registry Entry Association........................................... 27
- Figure 4: Example showing a Classification Tree........................................... 30
- Figure 5: Information Model Classification View......................................... 31
- Figure 6: Classification Instance Diagram.................................................... 31
- Figure 7: Context Sensitive Classification..................................................... 34
- Figure 8: Information Model: Security View............................................... 37

Table of Tables

- Table 1: Sample Classification Schemes.......................................................... 35
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* [RS], 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 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].

3.3 Audience

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

**ebXML Registry Information Model**

Copyright © ebXML 2000 & 2001. All Rights Reserved.
3.4 Related Documents
The following specifications provide some background and related information to the reader:

- **ebXML Registry Business Domain Model [BDM]** - defines requirements for ebXML Registry Services
- **ebXML Registry Services Specification [RS]** - defines the actual Registry services based on this information model
- **Collaboration Protocol Agreement Specification [CPA]** (under development) - defines how profiles can be defined for a party and how two parties’ profiles may be used to define a party agreement
- **ebXML Business Process Specification Schema [BPM]**

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 in progress within other ebXML working groups
- Be able to evolve to support future ebXML Registry requirements
- Be compatible with other ebXML specifications

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

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 [RS]. 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: Public View

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

Figure 1 shows the 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.

Figure 1: Information Model Public View
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 chapter 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 chapter 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 chapter 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 chapter 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 chapter 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 chapter 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 chapter 10. The interfaces Classification and ClassificationNode will be covered in detail separately in chapter 11.

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

Figure 2: Information Model Inheritance View

7.1 Interface Object

All Known Subinterfaces:

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

Object provides a common base interface for almost all objects in the information model. Information model classes whose instances have a unique identity and an independent life cycle are descendants of the Object class.
Note that Slot and PostalAddress are not descendants of the Object class because their instances do not have an independent existence and unique identity. They are always a part of some other class's instance (e.g. Organization has a PostalAddress).

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessControlPolicy.getAccessControlPolicy()</td>
<td>Gets the AccessControlPolicy object associated with this Object. An AccessControlPolicy defines the security model associated with the Object in terms of “who is permitted to do what” with that Object. Maps to attribute named accessControlPolicy.</td>
</tr>
<tr>
<td>String getDescription()</td>
<td>Gets the context independent textual description for this object. Maps to attribute named description.</td>
</tr>
<tr>
<td>String getName()</td>
<td>Gets user friendly context independent name of object in repository. Maps to attribute named name.</td>
</tr>
<tr>
<td>String getID()</td>
<td>Gets the universally unique ID (UUID) for this object. Maps to attribute named id.</td>
</tr>
<tr>
<td>void setDescription(String description)</td>
<td>Sets the context independent textual description for this object.</td>
</tr>
<tr>
<td>void setName(String name)</td>
<td>Sets user friendly context independent name of object in repository.</td>
</tr>
<tr>
<td>void setID(String id)</td>
<td>Sets the universally unique ID (UUID) for this object.</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:
Object, 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><strong>Collection</strong></td>
<td><strong>getAssociatedObjects()</strong></td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td><strong>getAuditTrail()</strong></td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td><strong>getClassificationNodes()</strong></td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td><strong>getExternalLinks()</strong></td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><strong>getObjectType()</strong></td>
</tr>
<tr>
<td><strong>Collection</strong></td>
<td><strong>getPackages()</strong></td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><strong>getStatus()</strong></td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><strong>getUserVersion()</strong></td>
</tr>
<tr>
<td><strong>void</strong></td>
<td><strong>setUserVersion(String UserVersion)</strong></td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><strong>getStability()</strong></td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td><strong>getExpirationDate()</strong></td>
</tr>
</tbody>
</table>
guarantee provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes STABILITY_DYNAMIC implying that content can change at any time and in any manner. A null value implies that there is no expiration on stability attribute. Maps to attribute named expirationDate.

```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 object. Maps to attribute named slots.

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

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

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

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

### Methods inherited from interface

- `getAccessControlPolicy`, `getDescription`, `getName`, `getID`, `setDescription`, `setName`, `setID`

### Methods inherited from interface


#### 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 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</td>
</tr>
</tbody>
</table>
been submitted to the Registry and has been subsequently deprecated.

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

### 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 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 Collaboration Protocol Agreement (CPA) 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 XML document called Collaboration Protocol Profile (CPP) 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 Role in a Collaboration Protocol Profile (CPP).</td>
</tr>
<tr>
<td>ServiceInterface</td>
<td>An ExtrinsicObject of this type catalogues an XML description of a service interface as defined by [CPA].</td>
</tr>
<tr>
<td>SoftwareComponent</td>
<td>An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or class library).</td>
</tr>
<tr>
<td>Transport</td>
<td>An ExtrinsicObject of this type catalogues an XML description of a transport configuration as defined by [CPA].</td>
</tr>
<tr>
<td>UMLModel</td>
<td>An ExtrinsicObject of this type catalogues a UML model.</td>
</tr>
</tbody>
</table>
| XMLSchema       | An ExtrinsicObject of this type catalogues an XML schema.
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><strong>getName()</strong></td>
<td>Gets the name of this object. Maps to attribute named name.</td>
</tr>
<tr>
<td><strong>setName(String name)</strong></td>
<td>Sets the name of this object. Slot names are locally unique within a RegistryEntry instance.</td>
</tr>
<tr>
<td><strong>getSlotType()</strong></td>
<td>Gets the slotType or category for this slot. Maps to attribute named slotType.</td>
</tr>
<tr>
<td><strong>setSlotType(String slotType)</strong></td>
<td>Sets the slotType or category for this slot.</td>
</tr>
<tr>
<td><strong>getValues()</strong></td>
<td>Gets the collection of values for this object. The type for each value is String. Maps to attribute named values.</td>
</tr>
<tr>
<td><strong>setValues(Collection values)</strong></td>
<td>Sets the collection of values for this object.</td>
</tr>
</tbody>
</table>

### 7.5 Interface ExtrinsicObject

**All Superinterfaces:**
- RegistryEntry, Object, 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>
<tbody>
<tr>
<td><strong>getContentURI()</strong></td>
<td>Gets the URI to the content catalogued by this ExtrinsicObject.</td>
</tr>
</tbody>
</table>
### 7.6 Interface IntrinsicObject

**All Superinterfaces:**
- RegistryEntry, Object, 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, Object, Versionable

---
Logically related registry entries 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

<table>
<thead>
<tr>
<th>Collection</th>
<th><code>getMemberObjects()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Get the collection of RegistryEntries that are members of this Package. Maps to attribute named <code>memberObjects</code>.</td>
</tr>
</tbody>
</table>

#### 7.8 Interface `ExternalIdentifier`

**All Superinterfaces:**

- `IntrinsicObject`, `RegistryEntry`, `Object`, `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 `Object` 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:**

<table>
<thead>
<tr>
<th>Method Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>String</strong> <code>getValue()</code></td>
</tr>
<tr>
<td>Gets the value of this <code>ExternalIdentifier</code>. Maps to attribute named <code>value</code>.</td>
</tr>
<tr>
<td><strong>Void</strong> <code>setValue(String value)</code></td>
</tr>
<tr>
<td>Sets the value of this <code>ExternalIdentifier</code>.</td>
</tr>
</tbody>
</table>

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

#### 7.9 Interface `ExternalLink`

**All Superinterfaces:**

- `IntrinsicObject`, `RegistryEntry`, `Object`, `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 objects 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 chapter describes the information model elements that support the audit trail capability of the Registry. Several classes in this chapter 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:**
- Object

AuditableEvent instances provide a long-term record of events that effect a change of state in a RegistryEntry. A RegistryEntry is associated with an ordered Collection of AuditableEvent instances that provide a complete audit trail for that Object.

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 Object.

8.1.1 Pred-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

User

getUser()

Gets the User that sent the request that generated
this event. Maps to attribute named user.

String

gETEventType()

The type of this event as defined by the name
attribute of an event type as defined in section 8.1.1. Maps
to attribute named eventType.

RegistryEntry

gETRegistryEntry()

Gets the RegistryEntry associated with this
AuditableEvent. Maps to attribute named
registryEntry.

Timestamp

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:

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

### 8.3 Interface Organization

**All Superinterfaces:**

*IntrinsicObject, RegistryEntry, Object, 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.

See Also:

Method Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostalAddress</td>
<td>getAddress()</td>
<td>Gets the PostalAddress for this Organization. Maps to attribute named address.</td>
</tr>
<tr>
<td>User</td>
<td>getPrimaryContact()</td>
<td>Gets the primary Contact for this Organization. The primary contact is a reference to a User object. Maps to attribute named primaryContact.</td>
</tr>
<tr>
<td>TelephoneNumber</td>
<td>getFax()</td>
<td>Gets the FAX number for this Organization. Maps to attribute named fax.</td>
</tr>
<tr>
<td>Organization</td>
<td>getParent()</td>
<td>Gets the parent Organization for this Organization. Maps to attribute named parent.</td>
</tr>
<tr>
<td>TelephoneNumber</td>
<td>getTelephone()</td>
<td>Gets the main telephone number for this Organization. Maps to attribute named telephone.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String city</td>
<td>The city</td>
</tr>
<tr>
<td>String country</td>
<td>The country</td>
</tr>
<tr>
<td>String postalCode</td>
<td>The postal or zip code</td>
</tr>
<tr>
<td>String state</td>
<td>The state</td>
</tr>
<tr>
<td>String street</td>
<td>The street</td>
</tr>
</tbody>
</table>
8.5 Class TelephoneNumber

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

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>areaCode</td>
<td>String</td>
<td>Area code</td>
</tr>
<tr>
<td>countryCode</td>
<td>String</td>
<td>Country code</td>
</tr>
<tr>
<td>extension</td>
<td>String</td>
<td>Internal extension if any</td>
</tr>
<tr>
<td>number</td>
<td>String</td>
<td>The telephone number suffix not including the country or area code.</td>
</tr>
<tr>
<td>url</td>
<td>String</td>
<td>A URL that can dial this number electronically</td>
</tr>
</tbody>
</table>

8.6 Class PersonName

A simple entity class for a person’s name.

Field Summary

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
<td>String</td>
<td>The first name for this person.</td>
</tr>
<tr>
<td>lastName</td>
<td>String</td>
<td>The last name (surname) for this person.</td>
</tr>
<tr>
<td>middleName</td>
<td>String</td>
<td>The middle name for this person.</td>
</tr>
</tbody>
</table>

9 Registry Entry 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 Registry Entries

A RegistryEntry may be associated with 0 or more objects. The information model defines an Association class. An instance of the Association class represents an association between a RegistryEntry and another Object. 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.

```
newCPPtoOldCPPAssociation:Association
[associationType = ASSOCIATION_TYPE_SUPERSEDES]
```

```
cppExtrinsicObject

sourceObject

oldCPPExtrinsicObject

targetObject
```

Figure 3: Example of Registry Entry Association

10.1 Interface Association

All Superinterfaces:
IntrinsicObject, RegistryEntry, Object, Versionable

All Known Subinterfaces:
Classification

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

An instance of the Association class represents an association between two Objects.
### String `getAssociationType()`

Gets the association type for this Association. This must be the name attribute of an association type as defined by 10.1.1. Maps to attribute named `associationType`.

### Object `getSourceObject()`

Gets the Object that is the source of this Association. Maps to attribute named `sourceObject`.

### String `getSourceRole()`

Gets the name of the role played by the source Object in this Association. Maps to attribute named `sourceRole`.

### Object `getTargetObject()`

Gets the Object that is the target of this Association. Maps to attribute named `targetObject`.

### String `getTargetRole()`

Gets the name of the role played by the target Object in this Association. Maps to attribute named `targetRole`.

### boolean `isBidirectional()`

Determine whether this Association is bi-directional. Maps to attribute named `bidirectional`.

### void `setBidirectional(boolean bidirectional)`

Set whether this Association is bi-directional.

### void `setSourceRole(String sourceRole)`

Sets the name of the role played by the source Object in this Association.

### void `setTargetRole(String targetRole)`

Sets the name of the role played by the destination Object in this Association.

## 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>
<tbody>
<tr>
<td>RelatedTo</td>
<td>Defines that source object is an instance of target object.</td>
</tr>
<tr>
<td>Packages</td>
<td>Defines that the source Package object packages the target RegistryEntry object. Reserved for use in Packaging of Registry Entries.</td>
</tr>
<tr>
<td>ExternallyLinks</td>
<td>Defines that the source ExternalLink object externally</td>
</tr>
</tbody>
</table>
11 Classification of Registry Entries

This section describes the how the information model supports classification of RegistryEntries. 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](image)

[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.
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 5: Information Model Classification View**

**Figure 6: Classification Instance Diagram**

### 11.1 Interface `ClassificationNode`

**All Superinterfaces:**
- IntrinsicObject,
- RegistryEntry,
- Object,
- 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>Collection</th>
<th><code>getClassifiedObjects()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Get the collection of RegistryEntries classified by</td>
<td></td>
</tr>
</tbody>
</table>
### ClassificationNode

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getParent()</td>
<td>Gets the parent ClassificationNode for this ClassificationNode. Maps to attribute named parent.</td>
</tr>
<tr>
<td>getPath()</td>
<td>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.</td>
</tr>
<tr>
<td>setParent(ClassificationNode parent)</td>
<td>Sets the parent ClassificationNode for this ClassificationNode.</td>
</tr>
<tr>
<td>getCode()</td>
<td>Gets the code for this ClassificationNode. See [11.4] for details. Maps to attribute named code.</td>
</tr>
<tr>
<td>setCode(String code)</td>
<td>Sets the parent code for this ClassificationNode. See [11.4] for details.</td>
</tr>
</tbody>
</table>

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, Object, 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object getClassifiedObject()</td>
<td>Gets the Object that is classified by this Classification. Maps to attribute named classifiedObject.</td>
</tr>
<tr>
<td>Object getClassificationNode()</td>
<td>Gets the ClassificationNode that classifies the object in this Classification. Maps to attribute named classificationNode.</td>
</tr>
</tbody>
</table>

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 RegistryItems to codes defined by various taxonomies.

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

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 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 chapter 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 Object is associated with exactly one AccessControlPolicy which defines the policy rules that govern access to operations or methods performed on that Object. 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 Objects in the Registry. The Permissions for an Object are defined in an AccessControlPolicy object.

A Permission object authorizes access to a method in an Object 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 Object. Permissions defined in the Object'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><code>getPrivilegeAttributes()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Gets the PrivilegeAttributes associated with this Privilege.</td>
</tr>
<tr>
<td></td>
<td>Maps to attribute named <code>privilegeAttributes</code>.</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
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

Method Summary

<table>
<thead>
<tr>
<th>Collection</th>
<th>getGroups()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the Groups associated with this Principal. Maps to attribute named groups.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collection</th>
<th>getIdentities()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the Identities associated with this Principal. Maps to attribute named identities.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collection</th>
<th>getRoles()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gets the Roles associated with this Principal. Maps to attribute named roles.</td>
<td></td>
</tr>
</tbody>
</table>
13 References


[T] ebXML Technical Architecture

[OAS] OASIS Information Model

http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html

[ISO] ISO 11179 Information Model


[BDM] Registry and Repository: Business Domain Model

http://www.ebxml.org/specdrafts/RegRepv1-0.pdf

[RS] ebXML Registry Services Specification

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf


http://www.ebxml.org/specdrafts/Busv2-0.pdf

[CPA] Trading-Partner Specification

http://www.ebxml.org/project_teams/trade_partner/private/

[CTB] Context table informal document from Core Components

http://www.ebxml.org/project_teams/core_components/ContextTable.doc

[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. 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 the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or 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.