Registry Information Model

v1.0

Registry Team

11 May 2001

(This document is the non-normative version formatted for printing, July 2001)

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

Copyright © UN/CEFACT and OASIS, 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.

Table of Contents

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

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

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

1 Status of this Document

This document specifies an ebXML Technical Specification 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/specs/ebRIM.pdf

Latest version:

http://www.ebxml.org/specdrafts/RegistryInfoModelv1.0.pdf

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

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
Kyu-Chul Lee	Chungnam National University
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

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

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

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

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 an object that has been submitted to a Registry for storage and safekeeping (e.g. an XML document or a DTD). Every repository item is described by a RegistryEntry instance.

- The term “RegistryEntry” is used to refer to an object that provides metadata about a 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 deal with the actual content of the repository. 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.

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

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" *(UC*C) and "Lower Camel Case" *(LC*C) 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:

- [ebRS] ebXML Registry Services Specification v1.0 - defines the actual *Registry Services* based on this information model

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

- [ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification v1.0 - defines how profiles can be defined for a *Party* and how two *Parties'* profiles may be used to define a *Party* agreement
- [ebBPSS] ebXML Business Process Specification Schema v1.01
- [ebTA] ebXML Technical Architecture Specification v1.04

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

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

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

5 System Overview

5.1 Role of ebXML Registry

The *Registry* provides a stable store where information submitted by a *Submitting Organization* is made persistent. Such information 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.

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

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 the 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 to 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*.

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

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.

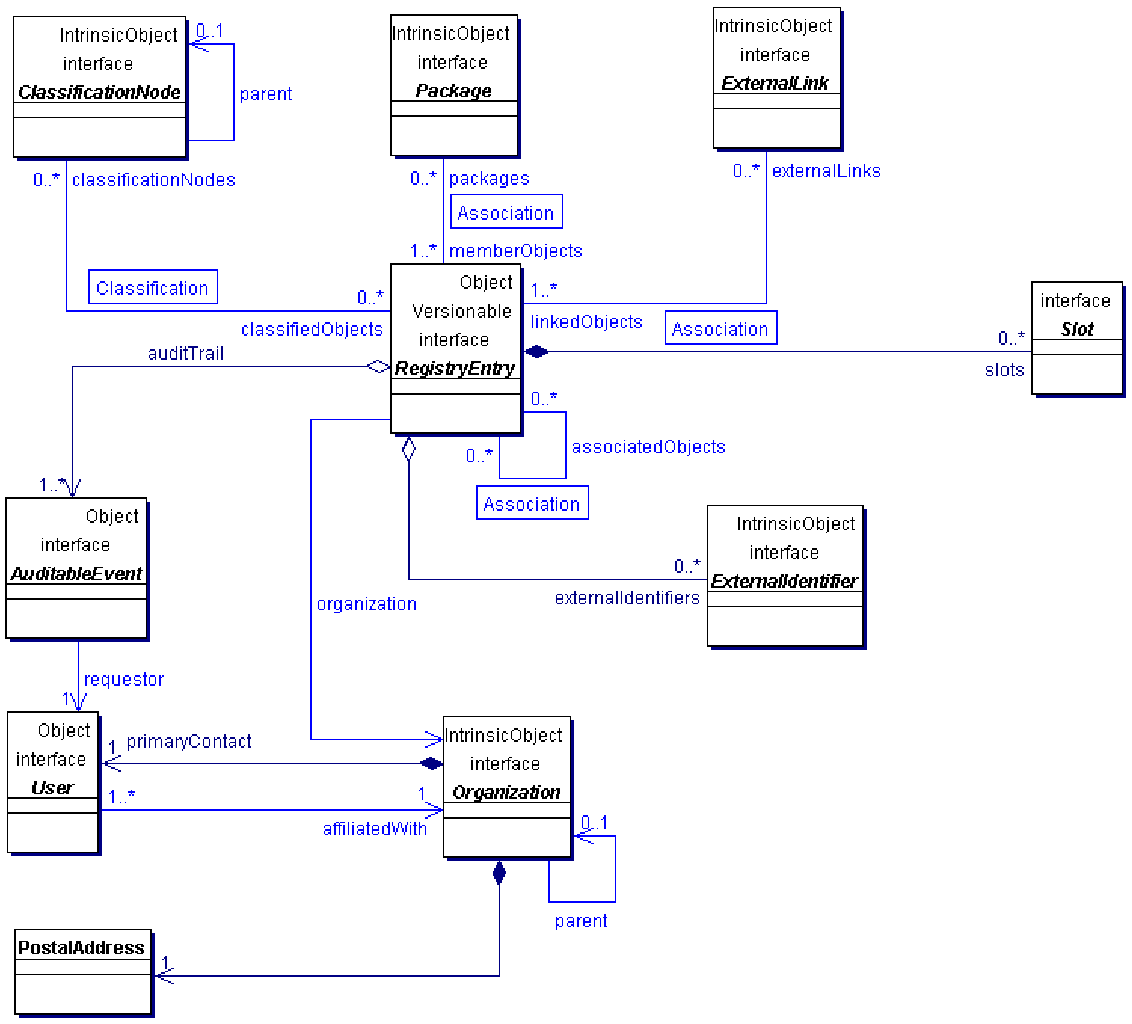

Figure 1: Information Model High Level Public View

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

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. 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.

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.

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

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 items 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.

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

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.

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

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.

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

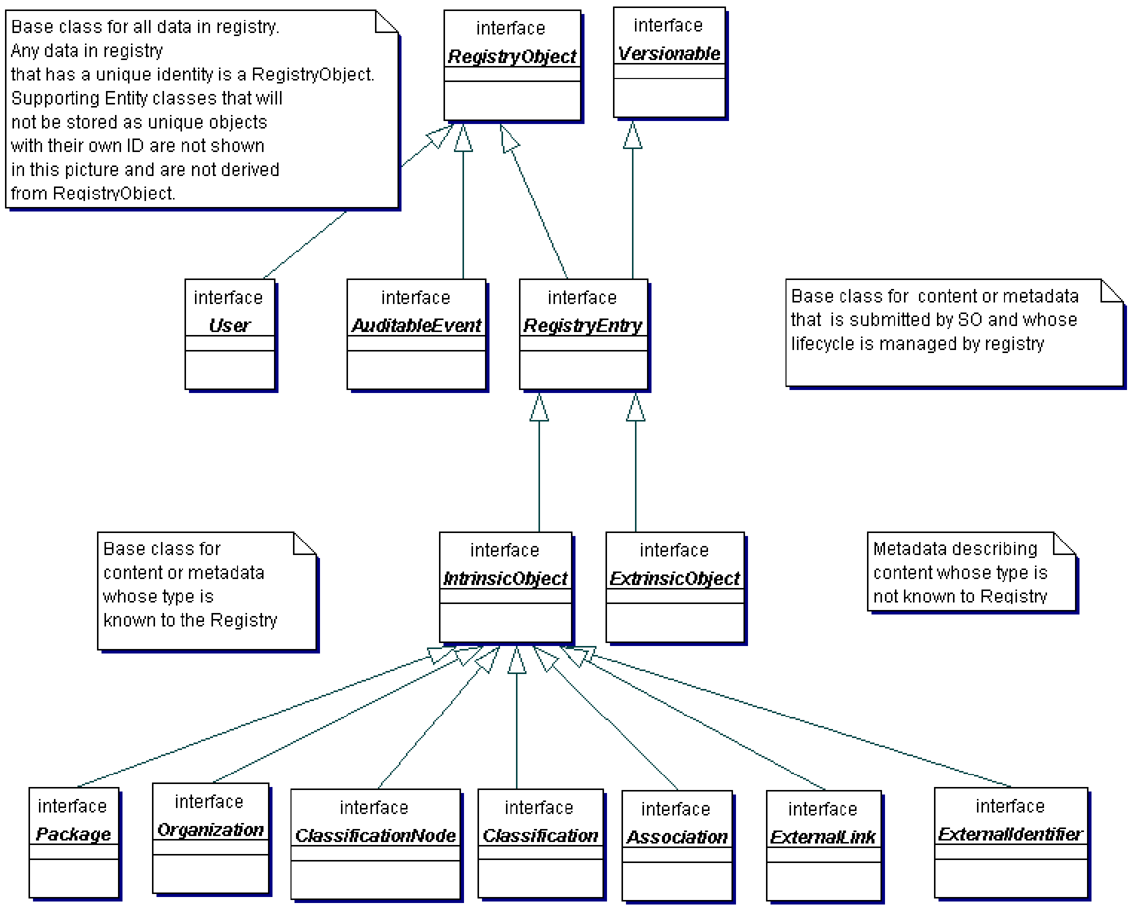
Base class for all data in registry. Any data in registry that has a unique identity is a RegistryObject. Supporting Entity classes that will not be stored as unique objects with their own ID are not shown in this picture and are not derived from RegistryObject.
interface RegistryObject
interface Versionable
interface User
interface AuditableEvent
interface RegistryEntry
Base class for content or metadata that is submitted by SO and whose lifecycle is managed by registry
Base class for content or metadata whose type is known to the Registry
interface IntrinsicObject
interface ExtrinsicObject
Metadata describing content whose type is not known to Registry
interface Package
interface Organization
interface ClassificationNode
interface Classification
interface Association
interface ExternalLink
interface ExternalIdentifier

Figure 2: Information Model *Inheritance* View

7.1 Interface RegistryObject

All Known Subinterfaces:

Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, RegistryEntry, Organization, Package, User, AuditableEvent, ExternalIdentifier

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 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).

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

Method Summary of RegistryObject	
AccessControlPolicy	**getAccessControlPolicy**`()` 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`.
`String`	**getDescription**`()` Gets the context independent textual description for this RegistryObject. Maps to attribute named `description`.
`String`	**getName**`()` Gets user friendly, context independent name for this RegistryObject. Maps to attribute named `name`.
`String`	**getID**`()` Gets the universally unique ID, as defined by [UUID], for this RegistryObject. Maps to attribute named `id`.
`void`	**setDescription**`(String description)` Sets the context, independent textual description for this RegistryObject.
`void`	**setName**`(String name)` Sets user friendly, context independent name for this RegistryObject.
`void`	**setID**`(String id)` Sets the universally unique ID, as defined by [UUID], for this RegistryObject.

7.2 Interface Versionable

All Known Subinterfaces:

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

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 of Versionable	
`int`	**getMajorVersion**`()` Gets the major revision number for this version of the Versionable object. Maps to attribute named `majorVersion`.
`int`	**getMinorVersion**`()` Gets the minor revision number for this version of the Versionable object. Maps to attribute named `minorVersion`.

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

Method Summary of Versionable	
`void`	**setMajorVersion**`(int majorVersion)` Sets the major revision number for this version of the Versionable object.
`void`	**setMinorVersion**`(int minorVersion)` Sets the minor revision number for this version of the Versionable object.

7.3 Interface RegistryEntry

All Superinterfaces:

RegistryObject, Versionable

All Known Subinterfaces:

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

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 of RegistryEntry	
`Collection`	**getAssociatedObjects**`()` Returns the collection of RegistryObjects associated with this RegistryObject. Maps to attribute named `associatedObjects`.
`Collection`	**getAuditTrail**`()` 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 `auditTrail`.
`Collection`	**getClassificationNodes**`()` Returns the collection of ClassificationNodes associated with this RegistryObject. Maps to attribute named `classificationNodes`.
`Collection`	**getExternalLinks**`()` Returns the collection of ExternalLinks associated with this RegistryObject. Maps to attribute named `externalLinks`.
`Collection`	**getExternalIdentifiers**`()` Returns the collection of ExternalIdentifiers associated with this RegistryObject. Maps to attribute named `externalIdentifiers`.
`String`	**getObjectType**`()` 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 `objectType`.

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

Method Summary of RegistryEntry	
`Collection`	**getOrganizations**`()` Returns the collection of Organizations associated with this RegistryObject. Maps to attribute named `organizations`.
`Collection`	**getPackages**`()` Returns the collection of Packages associated with this RegistryObject. Maps to attribute named `packages`.
`String`	**getStatus**`()` 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 `status`.
`String`	**getUserVersion**`()` Gets the userVersion attribute of the RegistryEntry within the *Registry*. The userVersion is the version for the RegistryEntry as assigned by the user.
`void`	**setUserVersion**`(String UserVersion)` Sets the userVersion attribute of the RegistryEntry within the *Registry*.
`String`	**getStability**`()` Gets the stability indicator for the RegistryEntry within the *Registry*. The stability indicator is provided by the submitter as a guarentee of the level of stability for the content. This SHOULD be the name of a stability type as described in 7.3.3. Maps to attribute named `stability`.
`Date`	**getExpirationDate**`()` Gets expirationDate attribute of the RegistryEntry within the *Registry*. This attribute defines a time limit upon the stability guarentee provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes STABILITY_DYNAMIC implying that content can change at any time and in any manner. A null value implies that there is no expiration on stability attribute. Maps to attribute named `expirationDate`.
`void`	**setExpirationDate**`(Date ExpirationDate)` Sets expirationDate attribute of the RegistryEntry within the *Registry*.
`Collection`	**getSlots**`()` Gets the collection of slots that have been dynamically added to this RegistryObject. Maps to attribute named `slots`.
`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.
`void`	**removeSlots**`(Collection slotNames)` Removes one or more slots from this RegistryObject. Slots to be removed are identified by their name.

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

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

Methods inherited from interface Versionable
getMajorVersion, getMinorVersion, setMajorVersion, 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 *Classification* scheme. While the scheme may easily be extended, a *Registry* MUST support the status types listed below.

Name	Description
Submitted	Status of a RegistryEntry that catalogues content that has been submitted to the *Registry*.
Approved	Status of a RegistryEntry that catalogues content that has been submitted to the *Registry* and has been subsequently approved.
Deprecated	Status of a RegistryEntry that catalogues content that has been submitted to the *Registry* and has been subsequently deprecated.
Withdrawn	Status of a RegistryEntry that catalogues content that has been withdrawn from the *Registry*.

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.

Name	Description
Unknown	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
CPA	An ExtrinsicObject of this type catalogues an *XML* document *Collaboration Protocol Agreement* (*CPA*) representing a technical agreement between two parties on how they plan to communicate with each other using a specific protocol.
CPP	An ExtrinsicObject of this type catalogues an document called *Collaboration Protocol Profile* (*CPP*) that provides information about a *Party* participating in a *Business* transaction.
Process	An ExtrinsicObject of this type catalogues a process description document.

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

Name	Description
Role	An ExtrinsicObject of this type catalogues an *XML* description of a *Role* in a *Collaboration Protocol Profile* (*CPP*).
ServiceInterface	An ExtrinsicObject of this type catalogues an *XML* description of a service interface as defined by [ebCPP].
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* MAY support the stability types listed below.

Name	Description
Dynamic	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed arbitrarily by submitter at any time.
DynamicCompatible	Stability of a RegistryEntry that indicates that the content is dynamic and may be changed in a backward compatible way by submitter at any time.
Static	Stability of a RegistryEntry that indicates that the content is static and will not be changed by submitter.

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

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 of Slot	
`String`	**getName**`()` Gets the name of this RegistryObject. Maps to attribute named `name`.
`void`	**setName**`(String name)` Sets the name of this RegistryObject. Slot names are locally unique within a RegistryEntry *Instance*.
`String`	**getSlotType**`()` Gets the slotType or category for this slot. Maps to attribute named `slotType`.
`void`	**setSlotType**`(String slotType)` Sets the slotType or category for this slot.
`Collection`	**getValues**`()` Gets the collection of values for this RegistryObject. The type for each value is String. Maps to attribute named `values`.
`void`	**setValues**`(Collection values)` Sets the collection of values for this RegistryObject.

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.

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

Method Summary of Extrinsic Object	
`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 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, ExternalIdentifier

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:

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

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 of Package	
`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 of ExternalIdentifier	
String	**getValue**`()` Gets the value of this ExternalIdentifier. Maps to attribute named `value`.
`Void`	**setValue**`(String value)` Sets the value of this ExternalIdentifier.

Note 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.

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

Method Summary of ExternalLink	
`Collection`	**getLinkedObjects**`()` Gets the collection of RegistryObjects that use this external link. Maps to attribute named `linkedObjects`.
`URI`	**getExternalURI**`()` Gets URI to the external content. Maps to attribute named `externalURI`.
`void`	**setExternalURI**`(URI uri)` Sets URI to the external content.

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

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

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:

RegistryObject

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 AuditableEvent 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.

Name	Description
Created	An *Event* that created a RegistryEntry.

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

Name	Description
Deleted	An *Event* that deleted a RegistryEntry.
Deprecated	An *Event* that deprecated a RegistryEntry.
Updated	An *Event* that updated the state of a RegistryEntry.
Versioned	An *Event* that versioned a RegistryEntry.

Method Summary of AuditableEvent	
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* occured. Maps to attribute named `timestamp.`

Note 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 of User	
Organization	**getOrganization**() Gets the *Submitting Organization* that sent the request that effected this change. Maps to attribute named `organization.`
PostalAddress	**getAddress**() Gets the postal address for this user. Maps to attribute named `address.`
`String`	**getEmail**() Gets the email address for this user. Maps to attribute named `email.`

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

Method Summary of User	
TelephoneNumber	**getFax**() The FAX number for this user. Maps to attribute named `fax.`
TelephoneNumber	**getMobilePhone**() The mobile telephone number for this user. Maps to attribute named `mobilePhone.`
PersonName	**getPersonName**() Name of contact person. Maps to attribute named `personName.`
TelephoneNumber	**getPager**() The pager telephone number for this user. Maps to attribute named `pager.`
TelephoneNumber	**getTelephone**() The default (land line) telephone number for this user. Maps to attribute named `telephone.`
`URL`	**getUrl**() The *URL* to the web page for this contact. Maps to attribute named `url.`

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 of Organization	
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.`

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

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

Note 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	
`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.

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

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.

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

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.

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

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**.

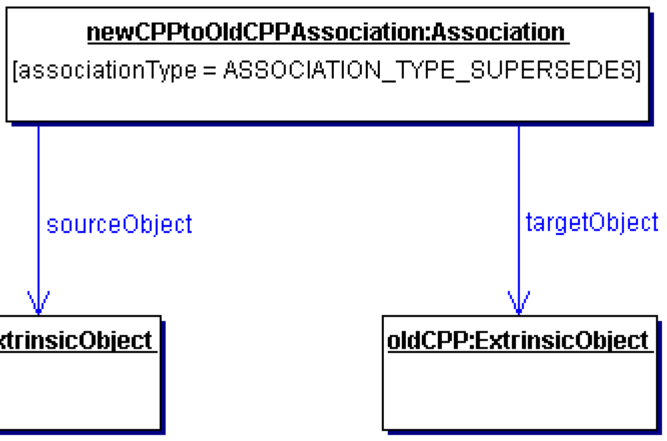
newCPPtoOldCPPAssociation:Association
[associationType = ASSOCIATION_TYPE_SUPERSEDES]
sourceObject
targetObject
cpp:ExtrinsicObject
oldCPP:ExtrinsicObject

Figure 3: Example of RegistryEntry Association

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 of Association	
`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`.

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

Method Summary of Association	
Object	**getSourceObject**() Gets the RegistryObject that is the source of this Association. Maps to attribute named `sourceObject`.
`String`	**getSourceRole**() Gets the name of the *Role* played by the source RegistryObject in this Association. Maps to attribute named `sourceRole`.
Object	**getTargetObject**() Gets the RegistryObject that is the target of this Association. Maps to attribute named `targetObject`.
`String`	**getTargetRole**() Gets the name of the *Role* played by the target RegistryObject 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 RegistryObject in this Association.
`void`	**setTargetRole**(`String targetRole`) Sets the name of the *Role* played by the destination RegistryObject 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.

Name	Description
RelatedTo	Defines that source RegistryObject is related to target RegistryObject.
HasMember	Defines that the source Package object has the target RegistryEntry object as a member. Reserved for use in Packaging of RegistryEntries.
ExternallyLinks	Defines that the source ExternalLink object externally links the target RegistryEntry object. Reserved for use in associating ExternalLinks with RegistryEntries.
ExternallyIdentifies	Defines that the source ExternalIdentifier object identifies the target RegistryEntry object. Reserved for use in associating ExternalIdentifiers with RegistryEntries.
ContainedBy	Defines that source RegistryObject is contained by the target RegistryObject.
Contains	Defines that source RegistryObject contains the target RegistryObject.

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

Name	Description
Extends	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
Implements	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
InstanceOf	Defines that source RegistryObject is an *Instance* of target RegistryObject.
SupersededBy	Defines that the source RegistryObject is superseded by the target RegistryObject.
Supersedes	Defines that the source RegistryObject supersedes the target RegistryObject.
UsedBy	Defines that the source RegistryObject is used by the target RegistryObject in some manner.
Uses	Defines that the source RegistryObject uses the target RegistryObject in some manner.
ReplacedBy	Defines that the source RegistryObject is replaced by the target RegistryObject in some manner.
Replaces	Defines that the source RegistryObject replaces the target RegistryObject in some manner.

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.

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

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).

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

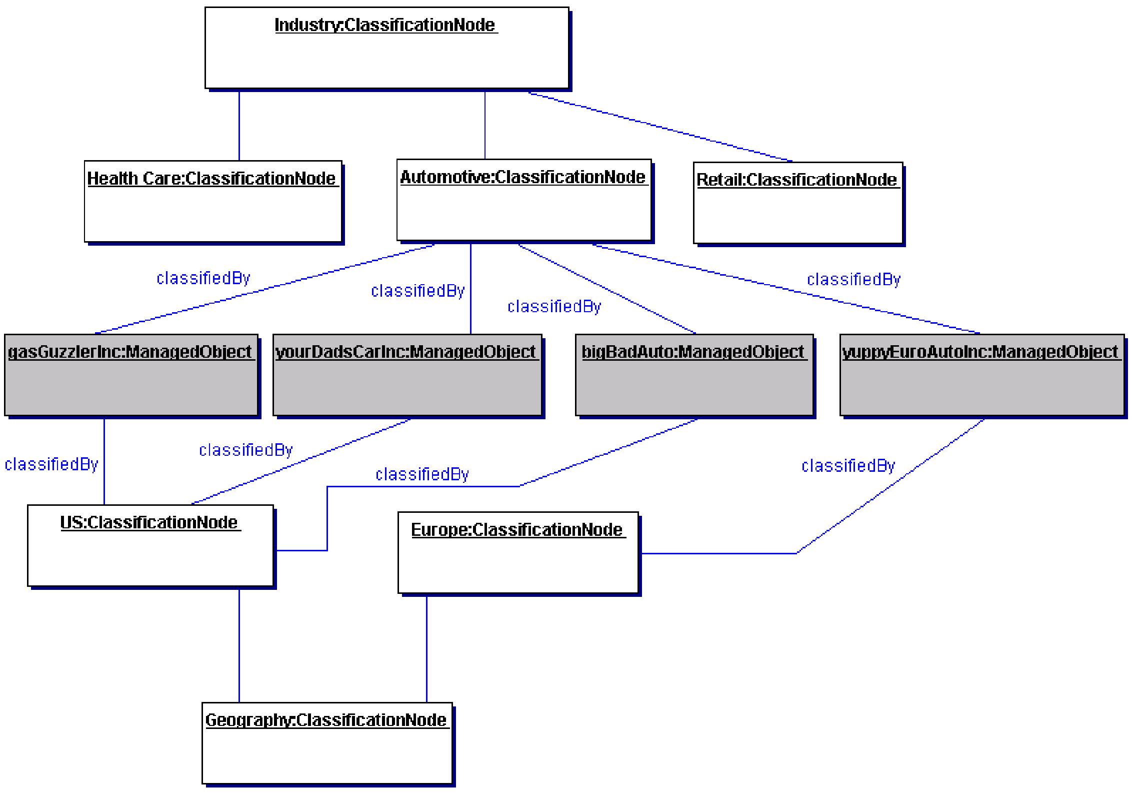
Industry:ClassificationNode
Health Care:ClassificationNode
Automotive:ClassificationNode
Retail:ClassificationNode
classifiedBy
classifiedBy
classifiedBy
classifiedBy
gasGuzzlerInc:ManagedObject
yourDadsCarInc:ManagedObject
bigBadAuto:ManagedObject
yuppyEuroAutoInc:ManagedObject
classifiedBy
classifiedBy
classifiedBy
classifiedBy
US:ClassificationNode
Europe:ClassificationNode
Geography:ClassificationNode

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**.

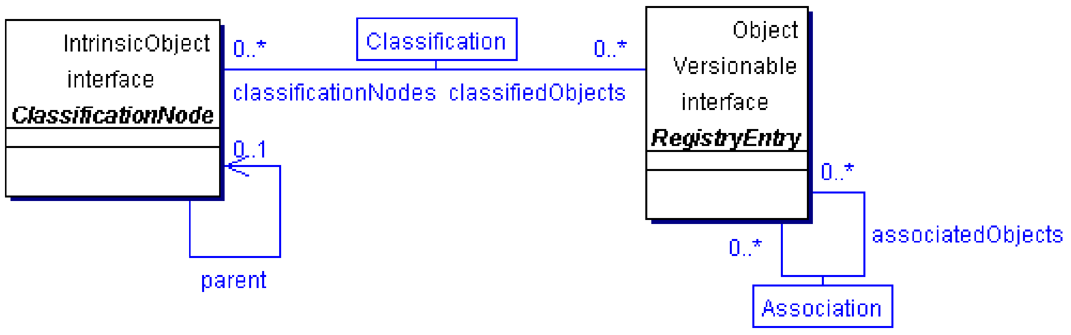

Figure 5: Information Model *Classification* View

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

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.

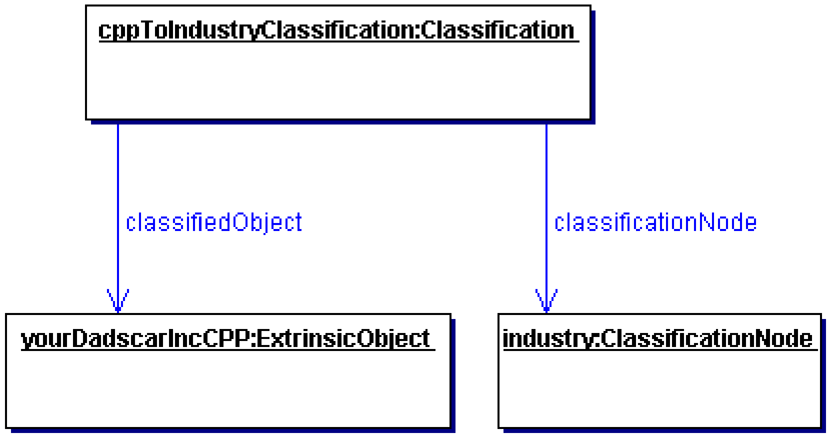
cppToIndustryClassification:Classification
classifiedObject
classificationNode
yourDadscarIncCPP:ExtrinsicObject
industry:ClassificationNode

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 of ClassificationNode	
`Collection`	**getClassifiedObjects**`()` Get the collection of RegistryObjects classified by this ClassificationNode. Maps to attribute named `classifiedObjects.`
ClassificationNode	**getParent**`()` Gets the parent ClassificationNode for this ClassificationNode. Maps to attribute named `parent.`

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

Method Summary of ClassificationNode	
`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 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 of Classification	
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 Methods inherited from the base interfaces of this interface are not shown.

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

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.

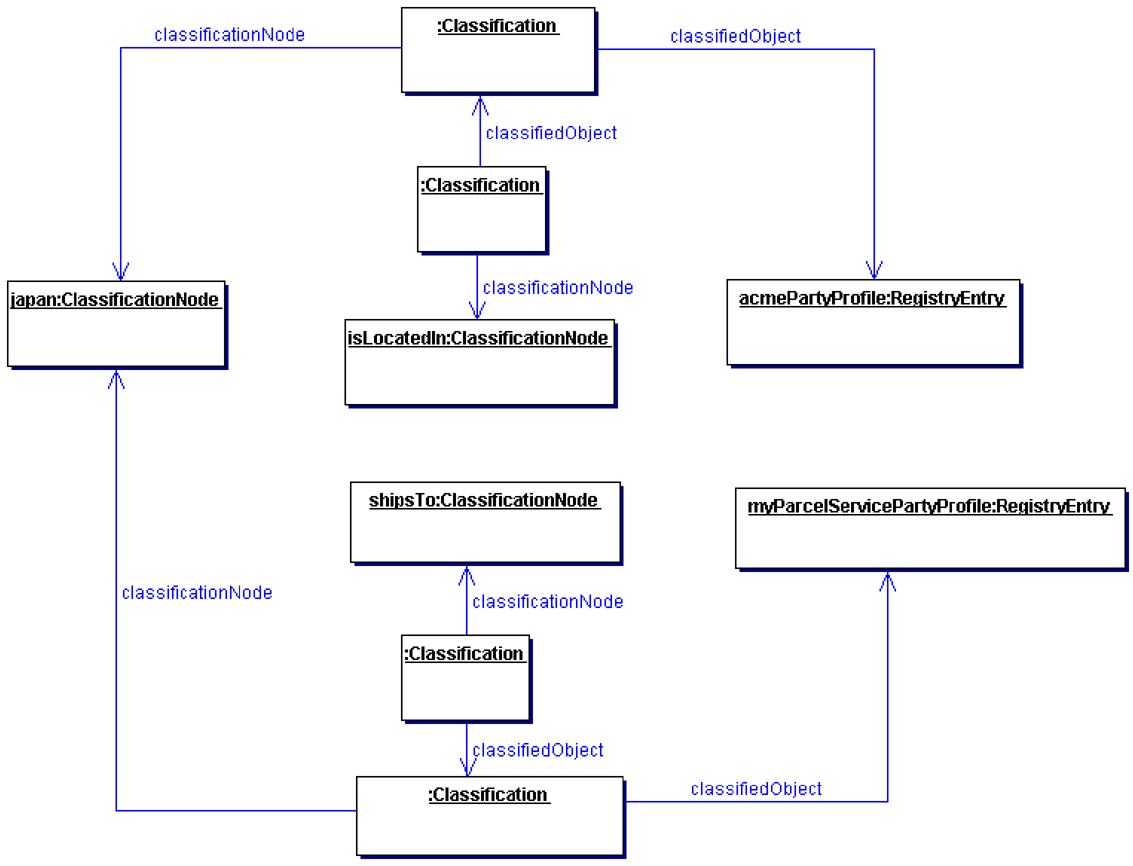

Figure 7: Context Sensitive *Classification*

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:

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

- 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.

Classification Scheme (Context)	**Usage Example**
Industry	Find all Parties in Automotive industry
Process	Find a ServiceInterface that implements a Process
Product	Find a *Business* that sells a product
Locale	Find a Supplier located in Japan
Temporal	Find Supplier that can ship with 24 hours
Role	Find All Suppliers that have a *Role* of “Seller”

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

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

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.

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

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.

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

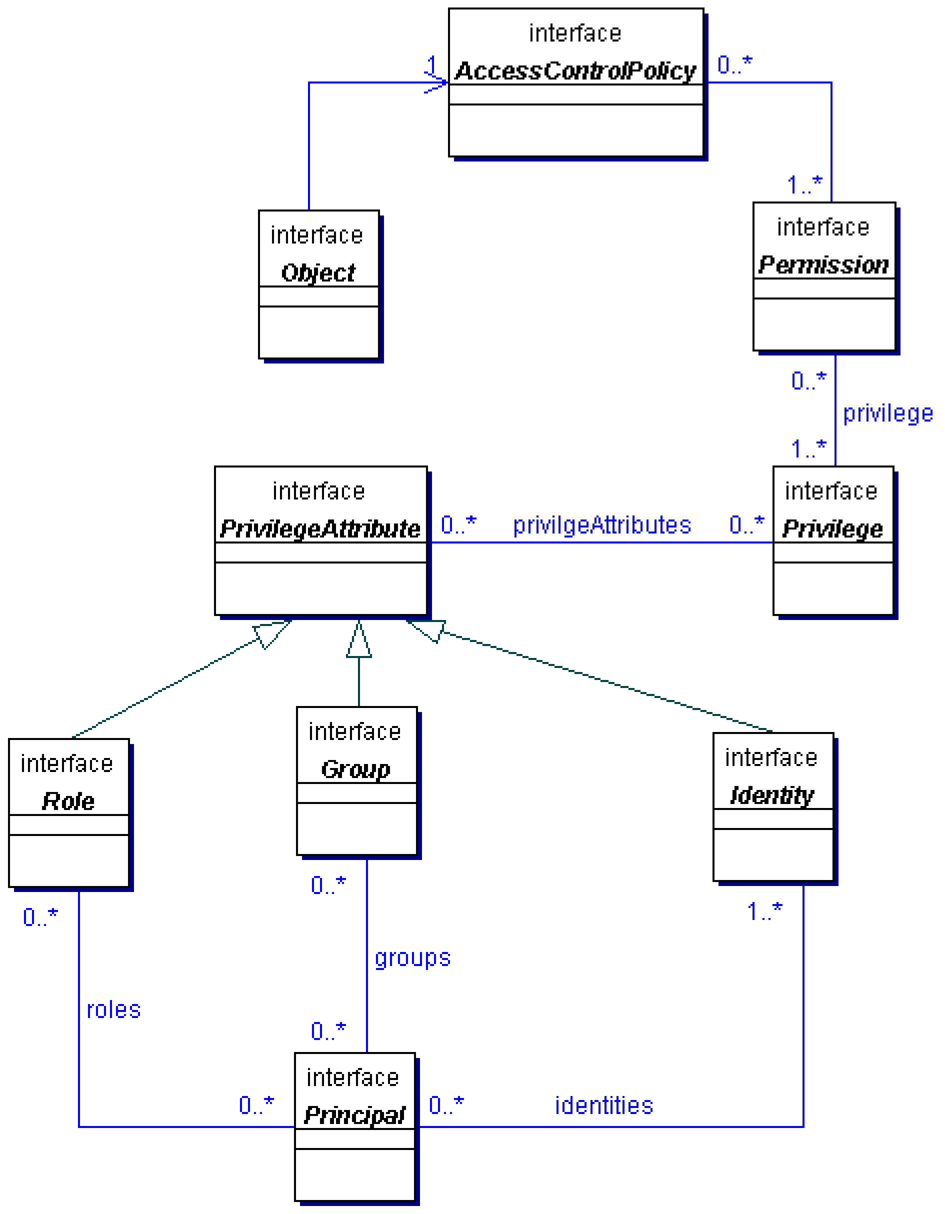

Figure 8: Information Model: Security View

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.

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

Method Summary of AccessControlPolicy	
`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 of Permission	
`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

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

Method Summary of Privilege	
`Collection`	**getPrivilegeAttributes**`()` Gets the PrivilegeAttributes associated with this Privilege. Maps to attribute named `privilegeAttributes.`

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).

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

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

Method Summary of Principal	
`Collection`	**getGroups**`()` Gets the Groups associated with this Principal. Maps to attribute named `groups.`
`Collection`	**getIdentities**`()` Gets the Identities associated with this Principal. Maps to attribute named `identities.`
`Collection`	**getRoles**`()` Gets the Roles associated with this Principal. Maps to attribute named `roles.`

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

13 References

[ebGLOSS] ebXML Glossary,

http://www.ebxml.org/specs/ebGLOSS.pdf

[ebTA] ebXML Technical Architecture Specification

http://www.ebxml.org/specs/ebTA.pdf

[OAS] OASIS Information Model

http://xsun.sdct.itl.nist.gov/regrep/OasisRegrepSpec.pdf

[ISO] ISO 11179 Information Model

http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

[BRA97] IETF (Internet Engineering Task Force). RFC 2119: Key words for use in RFCs to Indicate Requirement Levels

http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2119.html

[ebRS] ebXML Registry Services Specification

http://www.ebxml.org/specs/ebRS.pdf

[ebBPSS] ebXML Business Process Specification Schema

http://www.ebxml.org/specs/ebBPSS.pdf

[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification

http://www.ebxml.org/specs/ebCPP.pdf

[UUID] DCE 128 bit Universal Unique Identifier

http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20

http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

[XPATH] XML Path Language (XPath) Version 1.0

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

http://www.w3.org/TR/xpath

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

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.

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

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

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

Country:	USA
Phone:	781.442.0703
Email:	najmi@east.sun.com

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