Creating A Single Global Electronic Market

ebXML Registry Information Model

ebXML Registry Project Team

Working Draft 1/19/2001

This version: Version 0.55

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/registryInfoModelv0.55.pdf

Latest version:
http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.55.pdf

Previous version:
http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf

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.

Joseph Baran - Extol
Lisa Carnahan – NIST
Joe Dalman - Tie
Philippe DeSmedt - Viquity
Sally Fuger - AIAG
Steve Hanna - Sun Microsystems
Scott Hinkelman - IBM
Michael Kass, NIST
Jong.L Kim – Innodigital
Bob Miller - GXS
Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan
Dale Moberg – Sterling Commerce
Ron Monzillo – Sun Microsystems
JP Morgenthal – XML Solutions
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
Kim Tae Soo - Government of Korea
Nikola Stojanovic - Columbine JDS Systems
David Webber - XML Global
Yutaka Yoshida - Sun Microsystems
Prasad Yendluri - webmethods
Peter Z. Zhoo - Knowledge For the new Millennium

Table of Contents

1 STATUS OF THIS DOCUMENT 1

2 EBXML PARTICIPANTS 2

3 INTRODUCTION 6

3.1 SUMMARY OF CONTENTS OF DOCUMENT 6
3.2 GENERAL CONVENTIONS 6
3.3 AUDIENCE..6
3.4 RELATED DOCUMENTS 7

4 DESIGN OBJECTIVES 7

4.1 GOALS 7
4.2 CAVEATS AND ASSUMPTIONS 7

5 SYSTEM OVERVIEW 7

5.1 ROLE OF EBXML REGISTRY..7
5.2 REGISTRY SERVICES 8
5.3 WHAT THE REGISTRY INFORMATION MODEL DOES 8
5.4 HOW THE REGISTRY INFORMATION MODEL WORKS 8
5.5 WHERE THE REGISTRY INFORMATION MODEL MAY BE IMPLEMENTED 8

6 REGISTRY INFORMATION MODEL: PUBLIC VIEW....................................8

6.1 MANAGEDOBJECT 9
6.2 ASSOCIATION..9
6.3 EXTERNALLINK 9
6.4 CLASSIFICATIONNODE..10
6.5 CLASSIFICATION 10
6.6 PACKAGE 10
6.7 AUDITABLEEVENT..10
6.8 POSTALADDRESS 10
6.9 CONTACT..10
6.10 ORGANIZATION..10

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

7.1 INTERFACE *OBJECT*..12
7.2 INTERFACE *VERSIONABLE*..14
7.3 INTERFACE *MANAGEDOBJECT*..14
7.3.1 Pre-defined ManagedObject Status Types..15
7.4 INTERFACE *EXTRINSICOBJECT*..16
7.4.1 Pre-Defined Extrinsic Object Types 17
7.5 INTERFACE *INTRINSICOBJECT*..18

7.6 INTERFACE *PACKAGE* 18
7.7 INTERFACE *EXTERNALLINK* 18

8 REGISTRY AUDIT TRAIL 19

8.1 INTERFACE *AUDITABLEEVENT* 19
8.2 INTERFACE *AUDITABLEIDENTITY* 20
8.3 INTERFACE *ORGANIZATION* 21
8.4 CLASS *CONTACT* 21
8.5 CLASS *POSTALADDRESS* 22
8.6 CLASS *TELEPHONENUMBER* 22
8.7 CLASS *PERSONNAME* 23

9 MANAGED OBJECT NAMING 23

10 ASSOCIATION OF MANAGED OBJECTS 24

10.1 INTERFACE *ASSOCIATION* 24

11 CLASSIFICATION OF MANAGED OBJECTS 26

11.1 INTERFACE *CLASSIFICATIONNODE* 28
11.2 INTERFACE *CLASSIFICATION* 29
11.2.1 Context Sensitive Classification 30
11.3 EXAMPLE OF CLASSIFICATION SCHEMES 31

12 QUERYING OF MANAGED OBJECTS 31

12.1 OBJECT QUERY USE CASES 31
12.1.1 Browse and Drill Down Query 31
12.1.2 Ad Hoc Queries Based on Object Metadata And Content 32
12.1.3 Keyword Search Query 32

13 INFORMATION MODEL: SECURITY VIEW 32

13.1 INTERFACE *ACCESSCONTROLPOLICY* 33
13.2 INTERFACE *PERMISSION* 34
13.3 INTERFACE *PRIVILEGE* 34
13.4 INTERFACE *PRIVILEGEATTRIBUTE* 35
13.5 INTERFACE *ROLE* 35
13.6 INTERFACE *GROUP* 35
13.7 INTERFACE *IDENTITY* 35
13.8 INTERFACE *PRINCIPAL* 36

14 REFERENCES 37

15 DISCLAIMER 37

16 CONTACT INFORMATION 38

COPYRIGHT STATEMENT 39

Table of Figures

Figure 1: Information Model Public View 9
Figure 3: Information Model Inheritance View 12
Figure 4: Example of Managed Object Association 24
Figure 5: Example showing a Classification Tree 27
Figure 6: Information Model Classification View 28
Figure 7: Classification Instance Diagram 28
Figure 8: Context Sensitive Classification 30
Figure 9: Information Model: Security View 33

Table of Tables

Table 1: Sample Classification Schemes 31

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

- o UML diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific implementation or methodology requirements.
- o 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.*
- o The term *“managed object content”* is used to refer to actual Registry content (e.g. a DTD, as opposed to metadata about the DTD).
- o The term *“ManagedObject”* is used to refer to an object that provides metadata about content instance (*managed object content*).

The information model *does not* contain *any* elements that are the actual content of the Registry (*managed object content*). 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:

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

3.4 Related Documents

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

a) *ebXML Registry Business Domain Model* [BDM] - defines requirements for ebXML Registry Services
b) *ebXML Registry Services Specification* [RS] - defines the actual Registry services based on this information model
c) *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
d) *ebXML Business Process Specification Schema* [BPM]

4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

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

- o Defines what types of objects are stored in the Registry
- o Defines how stored objects are organized in the Registry
- o 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]Note that 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.

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 modeling metadata and not actual content.

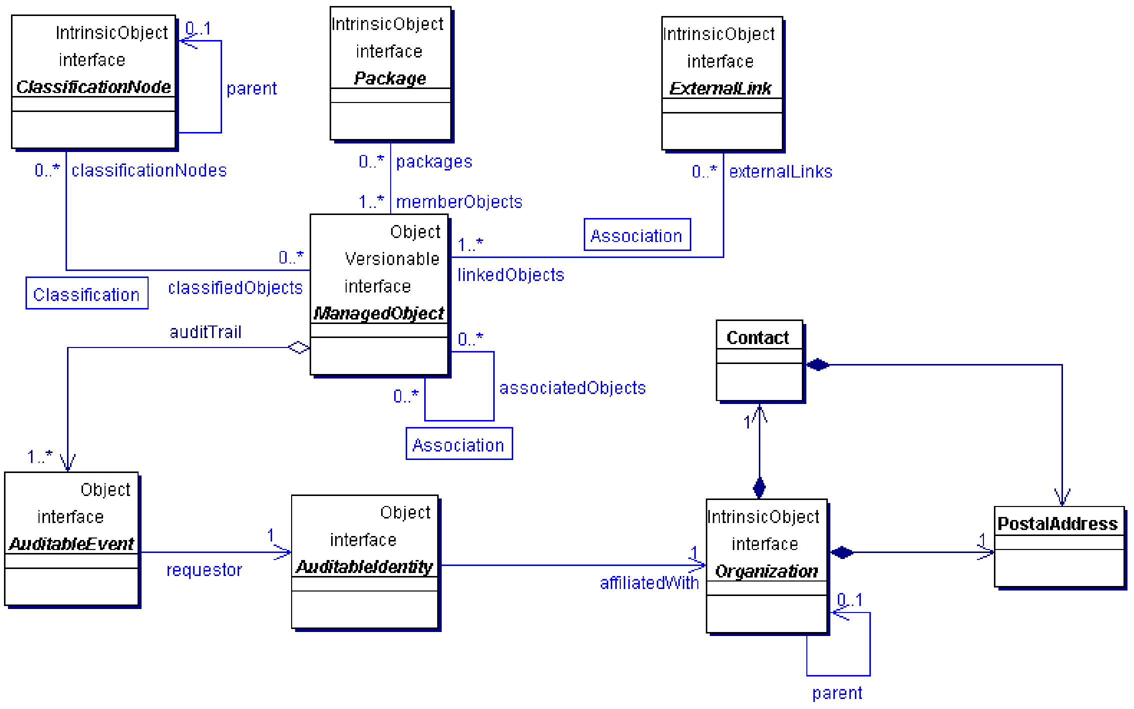


Figure 1: Information Model Public View

6.1 ManagedObject

The central object in the information model is a ManagedObject. An instance of ManagedObject exists for each content instance submitted to the Registry. Instances of the ManagedObject class provide metadata about a managed object content in the Registry. The actual managed object content (e.g. a DTD) is not contained in an instance of the ManagedObject class. Note that most classes in the information model are specialized sub-classes of ManagedObject.

6.2 Association

Association instances are ManagedObjects that are used to define many-to-many associations between objects in the information model. Associations are described in detail in chapter 10.

6.3 ExternalLink

ExternalLink instances are ManagedObjects that model a named URI to content that may reside outside the Registry. ManagedObject may be associated with any number of ExternalLinks.

Consider the case where a Submitting Organization submits a managed object content (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 ManagedObject. The user may click on such links and navigate to an external web page referenced by the link.

6.4 ClassificationNode

ClassificationNode instances are ManagedObjects 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.

6.5 Classification

Classification instances are ManagedObjects that are used to classify managed object content by associating their ManagedObject instance with a ClassificationNode within a classification scheme. Classification is described in detail in chapter 11.

6.6 Package

Package instances are ManagedObjects that group logically related ManagedObjects 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.

6.7 AuditableEvent

AuditableEvent instances are Objects that are used to provide an audit trail for ManagedObjects. AuditableEvent is described in detail in chapter 8.

6.8 PostalAddress

PostalAddress is a simple reusable entity class that defines attributes of a postal address.

6.9 Contact

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

6.10 Organization

Organization instances are ManagedObjects 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 3 shows the inheritance or “is a” relationships between the classes in the information model. Note that it does not show the relationships since they have already been shown in . 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 modeling metadata and not actual content.

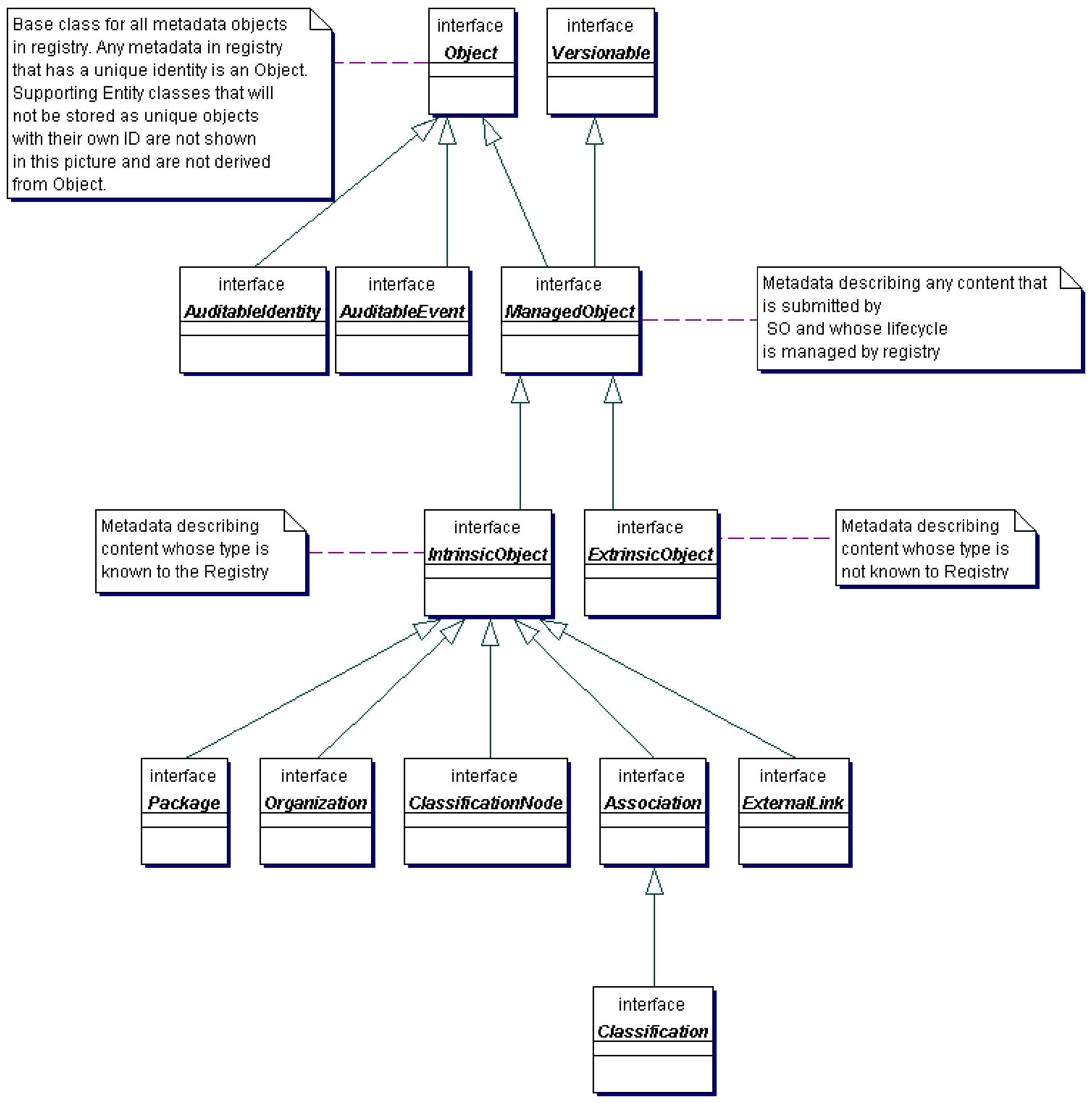

Base class for all metadata objects in registry. Any metadata in registry that has a unique identity is an Object. Supporting Entity classes that will not be stored as unique objects with their own ID are not shown in this picture and are not derived from Object.
interface Object
interface Versionable
interface AuditableIdentity
interface AuditableEvent
interface ManagedObject
Metadata describing any content that is submitted by SO and whose lifecycle is managed by registry
Metadata describing content 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 Association
interface ExternalLink
interface Classification

Figure 3: Information Model Inheritance View

7.1 Interface *Object*

All Known Subinterfaces:

Association, Classification, ClassificationNode, ExternalLink, ExtrinsicObject, IntrinsicObject, ManagedObject, 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 Contact 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	
AccessControlPolicy	**getAccessControlPolicy**() 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.
String	**getDescription**() Gets the context independent textual description for this object.
String	**getName**() Gets user friendly context independent name of object in repository.
String	**getID**() Gets the universally unique ID (UUID) for this object. Note that this ID also serves as URI for this object.
void	**setDescription**(String description) Sets the context independent textual description for this object.
void	**setName**(String name) Sets user friendly context independent name of object in repository.
void	**setID**(String id) Sets the universally unique ID (UUID) for this object. Note that this ID also serves as URI for this object.

7.2 Interface *Versionable*

All Known Subinterfaces:

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

---

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

Method Summary	
`int`	`getMajorVersion()` Gets the major revision number for this version of the Versionable object.
`int`	`getMinorVersion()` Gets the minor revision number for this version of the Versionable object.
`void`	`setMajorVersion(int majorVersion)` Gets 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 *ManagedObject*

All Superinterfaces:

Object, Versionable

All Known Subinterfaces:

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

---

ManagedObject 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 ManagedObject.

Method Summary	
Collection	**getAssociatedObjects**() Returns the collection of Objects associated with this object.
Collection	**getAuditTrail**() Returns the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.
Collection	**getClassificationNodes**() Returns the collection of ClassificationNodes associated with this object.
Collection	**getExternalLinks**() Returns the collection of ExternalLinks associated with this object.
Collection	**getPackages**() Returns the collection of Packages associated with this object.
int	**getStatus**() Gets the life cycle status of the ManagedObject within the Registry.
void	**setStatus**(int status) Sets the life cycle status of the ManagedObject within the Registry.

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

Methods inherited from interface
getMajorVersion, getMinorVersion, setMajorVersion, setMinorVersion

7.3.1 Pre-defined ManagedObject Status Types

The following table lists pre-defined choices for ManagedObject status attribute.

Field Summary	
static int	**STATUS_APPROVED** Status of a ManagedObject that catalogues content that has been submitted to the Registry and has been subsequently approved.
static int	**STATUS_DEPRECATED** Status of a ManagedObject that catalogues content that has been deprecated.
static int	**STATUS_SUBMITTED** Status of a ManagedObject that catalogues content that has been submitted to the Registry.

7.4 Interface *ExtrinsicObject*

All Superinterfaces:

ManagedObject, 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	
String	**getContentURI**() Gets the URI to the content catalogued by this ExtrinsicObject.
String	**getMimeType**() Gets the mime type associated with the content catalogued by this ExtrinsicObject.
int	**getObjectType**() Gets the pre-defined object type associated with the content catalogued by this ExtrinsicObject.
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.
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	**setObjectType**(int type) Sets the pre-defined object type associated with the content catalogued by this ExtrinsicObject.
void	**setOpaque**(boolean isOpaque) Sets whether the content catalogued by this ExtrinsicObject is opaque to (not readable by) the Registry.

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

7.4.1 Pre-Defined Extrinsic Object Types

The following table lists pre-defined types of ExtrinsicObjects.

Field Summary	
static int	**OBJECT_TYPE_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.
static int	**OBJECT_TYPE_CPP** 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.
static int	**OBJECT_TYPE_PROCESS** An ExtrinsicObject of this type catalogues a process description document.
static int	**OBJECT_TYPE_ROLE** An ExtrinsicObject of this type catalogues an XML description of a Role in a Collaboration Protocol Profile (CPP).
static int	**OBJECT_TYPE_SERVICE_INTERFACE** An ExtrinsicObject of this type catalogues an XML description of a service interface as defined by [CPA].
static int	**OBJECT_TYPE_SOFTWARE_COMPONENT** An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or class library).
static int	**OBJECT_TYPE_TRANSPORT** An ExtrinsicObject of this type catalogues an XML description of a transport configuration as defined by [CPA]..
static int	**OBJECT_TYPE_UML_MODEL** An ExtrinsicObject of this type catalogues a UML model.
static int	**OBJECT_TYPE_UNKNOWN**

	An ExtrinsicObject that catalogues content whose type is unspecified or unknown.
static int	**OBJECT_TYPE_XML_SCHEMA** An ExtrinsicObject of this type catalogues an XML schema (DTD, XML Schema, RELAX grammar, etc.).

7.5 Interface *IntrinsicObject*

All Superinterfaces:

ManagedObject, 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.

7.6 Interface *Package*

All Superinterfaces:

IntrinsicObject, ManagedObject, Object, Versionable

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

Method Summary	
Collection	**getMemberObjects**() Get the collection of ManagedObjects that are members of this Package

7.7 Interface *ExternalLink*

All Superinterfaces:

IntrinsicObject, ManagedObject, 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	
URI	`getExternalURI()` Gets URI to the external content.
void	`setExternalURI(URI uri)` Sets URI to the external content.

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 ManagedObject returns an ordered Collection of AuditableEvents. These AuditableEvents constitute the audit trail for the ManagedObject. AuditableEvents include a timestamp for the event. Each AuditableEvent has an AuditableIdentity identifying the specific user that performed an action that resulted in an AuditableEvent. Each AuditableIdentity has 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 ManagedObject. A ManagedObject 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 ManagedObject. For example a client request could Create, Update, Deprecate or Delete a

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

Field Summary	
`static int`	**`EVENT_TYPE_CREATED`** An event that created a ManagedObject
`static int`	**`EVENT_TYPE_DELETED`** An event that deleted a ManagedObject
`static int`	**`EVENT_TYPE_DEPRECATED`** An event that deprecated a ManagedObject
`static int`	**`EVENT_TYPE_UPDATED`** An event that updated the state of a ManagedObject
`static int`	**`EVENT_TYPE_VERSIONED`** An event that versioned a ManagedObject

Method Summary	
`AuditableIdentity`	**`getAuditableIdentity`**`()` Gets the AuditableIdentity that sent the request that generated this event.
`int`	**`getEventType`**`()` The type of this event as defined in table above.
`ManagedObject`	**`getManagedObject`**`()` Gets the ManagedObject associated with this AuditableEvent
`Timestamp`	**`getTimestamp`**`()` Gets the Timestamp for when this event occured.

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

8.2 Interface *AuditableIdentity*

All Superinterfaces:

Object

AuditableIdentity 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	
Organization	**getOrganization**() Gets the Submitting Organization that sent the request that effected this change.

8.3 Interface *Organization*

All Superinterfaces:

IntrinsicObject, ManagedObject, 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	
PostalAddress	**getAddress**() Gets the PostalAddress for this Organization.
Contact	**getContact**() Gets the primary Contact for this Organization.
TelephoneNumber	**getFax**() Gets the FAX number for this Organization.
Organization	**getParent**() Gets the parent Organization for this Organization.
TelephoneNumber	**getTelephone**() Gets the main telephone number for this Organization.

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

8.4 Class *Contact*

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

Field Summary	
PostalAddress	**address** The postal address for this Contact.
String	**email** The email address for this Contact.
TelephoneNumber	**fax** The FAX number for this Contact.
TelephoneNumber	**mobilePhone** The mobile telephone number for this Contact.
PersonName	**name** Name of contact person
TelephoneNumber	**pager** The pager telephone number for this Contact.
TelephoneNumber	**telephone** The default (land line) telephone number for this Contact.
URL	**url** The URL to the web page for this contact.

8.5 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
String	**street** The street

8.6 Class *TelephoneNumber*

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

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

8.7 Class *PersonName*

A simple entity class for a person's name.

Field Summary	
`String`	**`firstName`** The first name for this Contact.
`String`	**`lastName`** The last name (surname) for this Contact.
`String`	**`middleName`** The middle name for this Contact.

9 Managed Object Naming

A ManagedObject has a name that may or may not be unique within the Registry.

In addition a ManagedObjects 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 Managed Objects

A ManagedObject 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 ManagedObject 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 4.

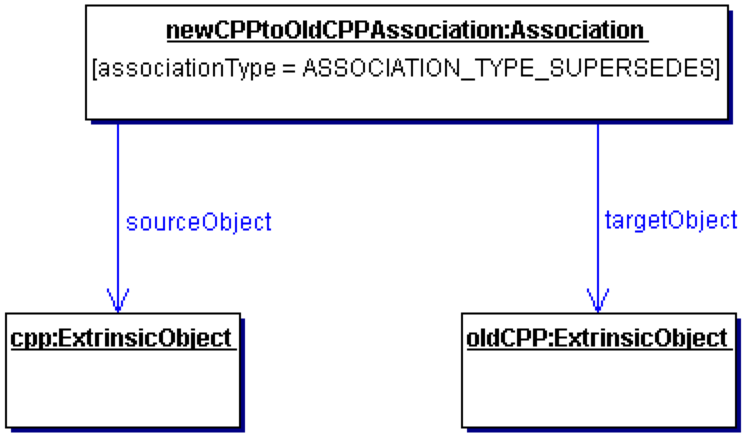

newCPPtoOldCPPAssociation:Association
[associationType = ASSOCIATION_TYPE_SUPERSEDES]
sourceObject
targetObject
cpp:ExtrinsicObject
oldCPP:ExtrinsicObject

Figure 4: Example of Managed Object Association

10.1 Interface *Association*

All Superinterfaces:

IntrinsicObject, ManagedObject, 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.

Field Summary	
`static int`	**`ASSOCIATION_TYPE_CLASSIFIED_BY`** Defines that the source object is classified by the target object.
`static int`	**`ASSOCIATION_TYPE_CONTAINED_BY`**

	Defines that source object is contained by the target object.
static int	**ASSOCIATION_TYPE_CONTAINS** Defines that source object contains the target object.
static int	**ASSOCIATION_TYPE_EXTENDS** Defines that source object inherits from or specializes the target object.
static int	**ASSOCIATION_TYPE_IMPLEMENTS** Defines that source object implements the functionality defined by the target object.
static int	**ASSOCIATION_TYPE_INSTANCE_OF** Defines that source object is an instance of target object
static int	**ASSOCIATION_TYPE_RELATED_TO** Defines that source object is an instance of target object.
static int	**ASSOCIATION_TYPE_SUPERSEDED_BY** Defines that the source object is superseded by the target object.
static int	**ASSOCIATION_TYPE_SUPERSEDES** Defines that the source object supersedes the target object.
static int	**ASSOCIATION_TYPE_USED_BY** Defines that the source object is used by the target object in some manner.
static int	**ASSOCIATION_TYPE_USES** Defines that the source object uses the target object in some manner.

568

Method Summary	
int	**getAssociationType**() Gets the predefined association type for this Association.
Object	**getSourceObject**() Gets the Object that is the source of this Association.
String	**getSourceRole**() Gets the name of the role played by the source Object in this Association.
Object	**getTargetObject**() Gets the Object that is the target of this Association.
String	**getTargetRole**() Gets the name of the role played by the target Object in this Association.
boolean	**isBidirectional**()

	Determine whether this Association is bi-directional.
`void`	`setAssociationType(int associationType)` Sets the predefined association type for this Association.
`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.

11 Classification of Managed Objects

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

A ManagedObject may be classified in many ways. For example the ManagedObject 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 5, ManagedObjects 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 ManagedObject 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).

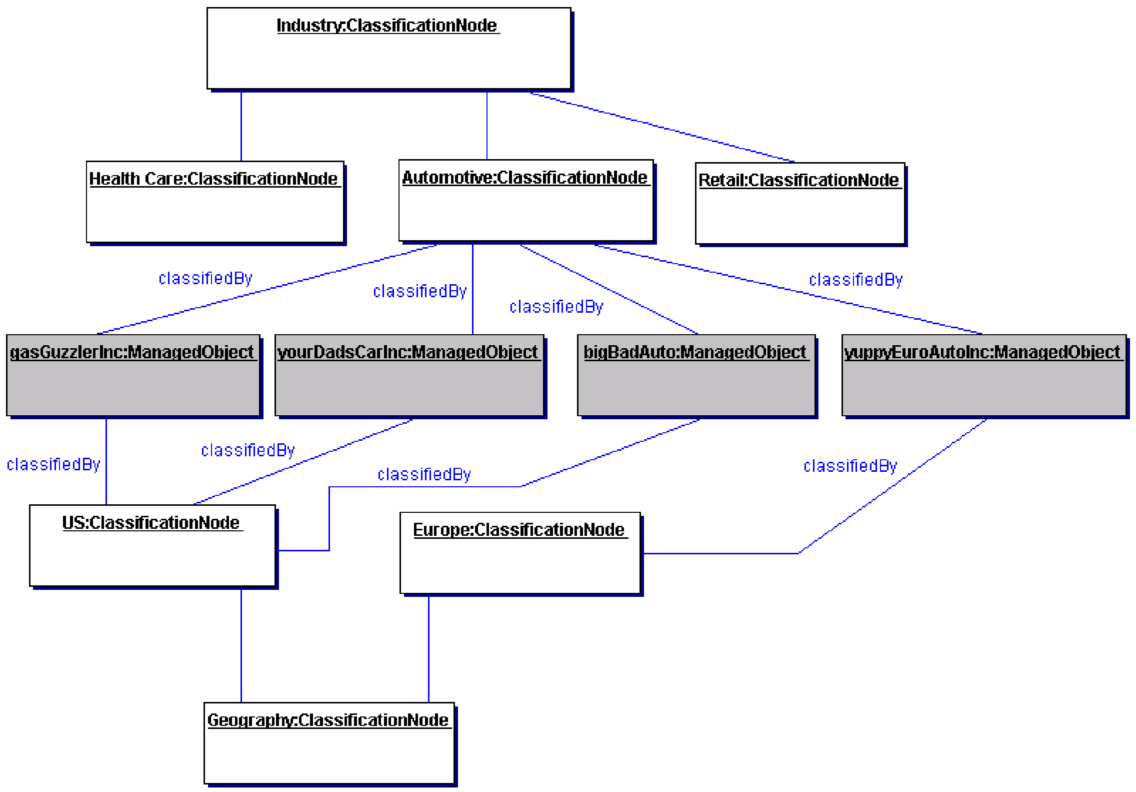

Industry:ClassificationNode
Health Care:ClassificationNode
Automotive:ClassificationNode
Retail:ClassificationNode
classifiedBy
classifiedBy
classifiedBy
classifiedBy
gasGuzzlerInc:ManagedObject
yourDadsCarInc:ManagedObject
bigBadAuto:ManagedObject
yuppyEuroAutoInc:ManagedObject
classifiedBy
classifiedBy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

Figure 5: 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 6.

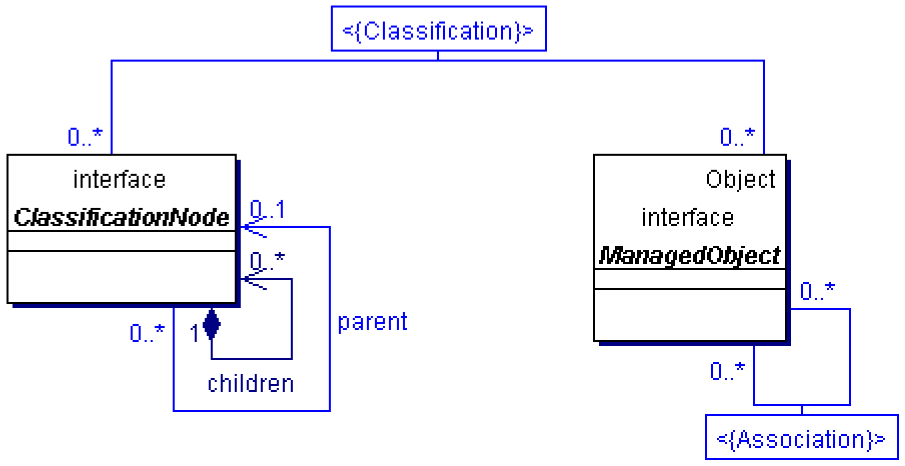
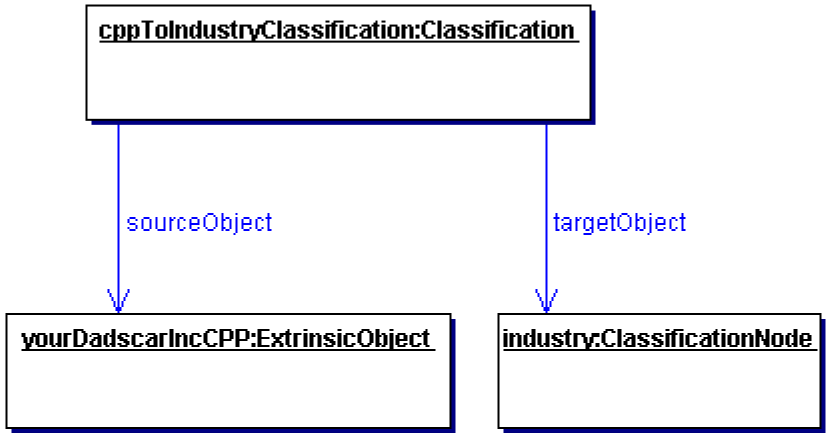


Figure 6: Information Model Classification View

A Classification is a specialized form of an Association. Figure 7 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
sourceObject
targetObject
yourDadscarIncCPP:ExtrinsicObject
industry:ClassificationNode

Figure 7: Classification Instance Diagram

11.1 Interface *ClassificationNode*

All Superinterfaces:

IntrinsicObject, ManagedObject, 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	
Collection	**getClassifiedObjects**() Get the collection of ManagedObjects classified by this ClassificationNode
ClassificationNode	**getParent**() Gets the parent ClassificationNode for this ClassificationNode.
String	**getPath**() Gets the path from the root ancestor of this ClassificationNode. Each element in the path is separated by a "." character and is the name of a ClassificationNode in the path (e.g "Geography.Asia.Japan").
void	**setParent**(ClassificationNode parent) Sets the parent ClassificationNode for this ClassificationNode.

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

In Figure 5, 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:

Association, IntrinsicObject, ManagedObject, Object, Versionable

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

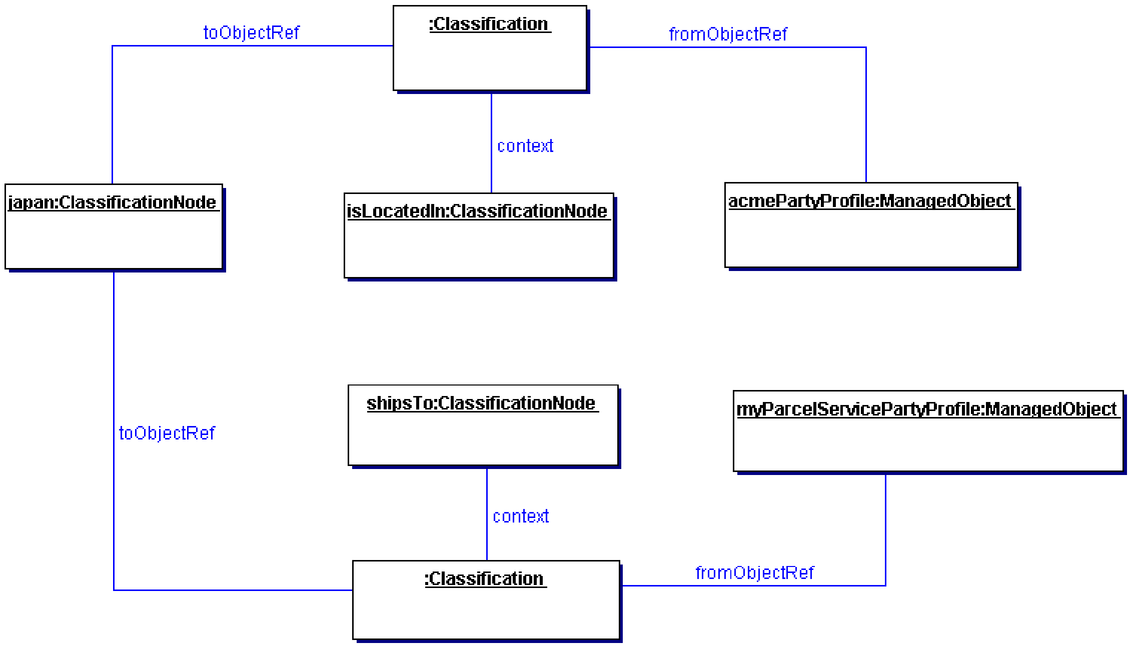
This interface currently does not define any attributes or methods. Note that methods inherited from the base interfaces of this interface are not shown.

Classification is a specialized form of Association from a ManagedObject to a specific ClassificationNode in the classification tree. The information model defines a Classification class as a sub-class of Association class to allow for future specialization as well as to make classification notion be obvious in the model.

In Figure 5, Classification instances are not explicitly shown but are implied as associations between the ManagedObjects (shaded leaf node) and the associated ClassificationNode

11.2.1 Context Sensitive Classification

Consider the case depicted in Figure 8 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.



:Classification
toObjectRef
fromObjectRef
context
japan:ClassificationNode
isLocatedIn:ClassificationNode
acmePartyProfile:ManagedObject
shipsTo:ClassificationNode
myParcelServicePartyProfile:ManagedObject
toObjectRef
context
:Classification
fromObjectRef

Figure 8: 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:

- o A ManagedObject to be classified by defining a Classification that associates it with a ClassificationNode in a classification tree.
- o A ManagedObject to be classified along multiple facets by having multiple classifications that associate it with multiple ClassificationNodes.
- o A classification defined for a ManagedObject 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

12 Querying of Managed Objects

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

12.1 Object Query Use Cases

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

12.1.1 Browse and Drill Down Query

In this scenario a user browses the Registry content using a GUI tool referred to as the Registry Browser. The user expects to initially browse the content based on the pre-defined classification schemes defined in section 11.3. The user may also use additional classification schemes that may have been defined for objects selected by the pre-defined classification scheme chosen. The user will select a managed object content and drill down to view the details of the object.

12.1.2 Ad Hoc Queries Based on Object Metadata And Content

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

12.1.3 Keyword Search Query

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

13 Information Model: Security View

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

Figure 9 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.

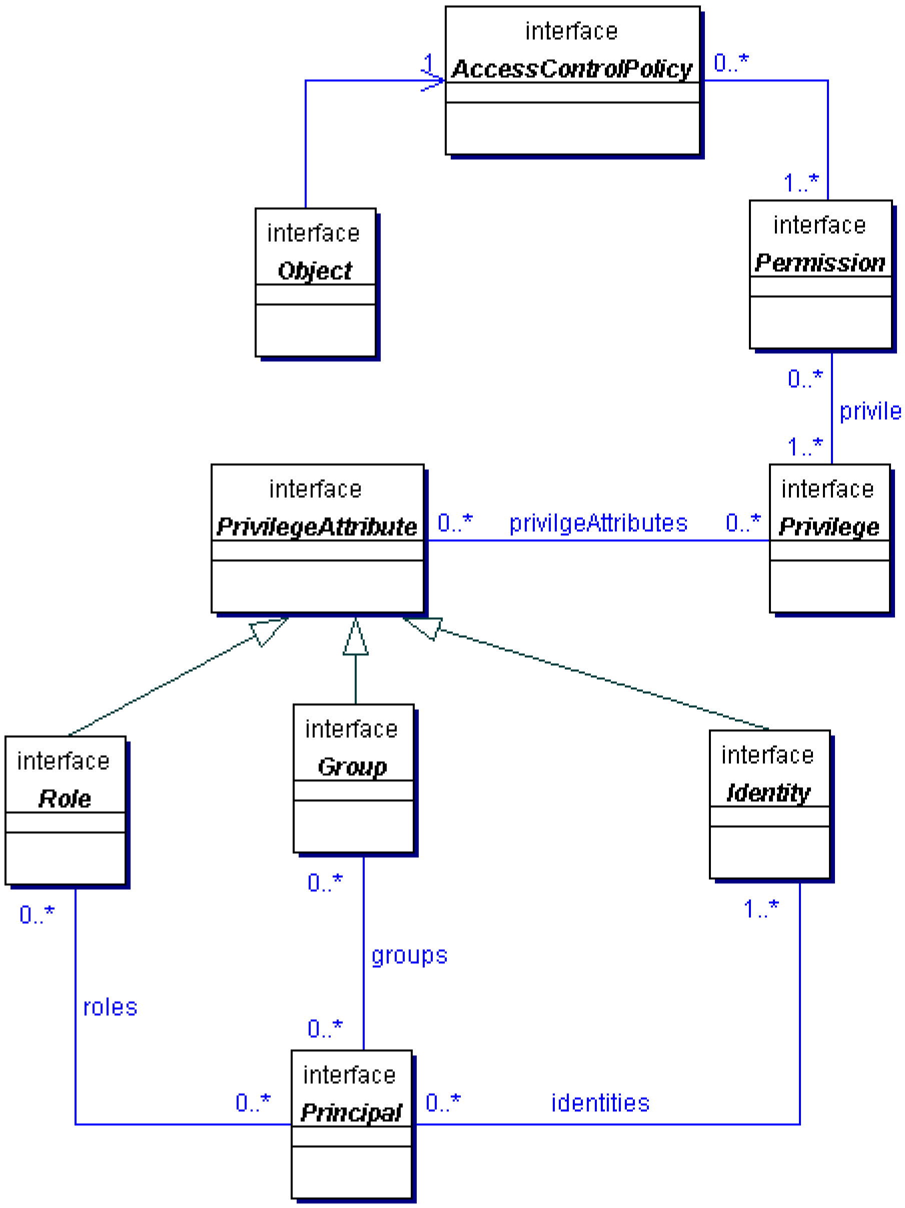


Figure 9: Information Model: Security View

13.1 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

13.2 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.
`Collection`	`getPrivileges()` Gets the Privileges associated with this Permission.

13.3 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	
`Collection`	`getPrivilegeAttributes()` Gets the PrivilegeAttributes associated with this Privilege.

13.4 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`

13.5 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.

13.6 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).

13.7 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.

13.8 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	
`Collection`	**`getGroups`**`()` Gets the Groups associated with this Principal.
`Collection`	**`getIdentities`**`()` Gets the Identities associated with this Principal.
`Collection`	**`getRoles`**`()` Gets the Roles associated with this Principal.

14 References

[GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm

[TA] ebXML Technical Architecture

[OAS] OASIS Information Model
http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html

[ISO] ISO 11179 Information Model
http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

[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

[BPM] ebXML Business Process Metamodel Specification Schema
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

15 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.

16 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