OASIS/ebXML Registry Information Model v2.02

Bug Fixes To Approved OASIS Standard

OASIS/ebXML Registry Technical Committee

May 2002

1 Status of this Document

Distribution of this document is unlimited.

This version:
http://www.oasis-open.org/committees/regrep/documents/2.02/specs/ebRIM.pdf

***Latest version*:**
http://www.oasis-open.org/committees/regrep/documents/2.02/specs/ebRIM.pdf

2 OASIS/ebXML Registry Technical Committee

This document has no standing and currently represents works-in-progress of the OASIS ebXML Registry TC. A future version of this document will be finalized and approved by the Registry TC as version 2.1.

At the time of v2.0 committee approval, the following were members of the OASIS/ebXML Registry Technical Committee:

Kathryn Breininger, Boeing (TC Chair)
Lisa Carnahan, US NIST
Joseph M. Chiusano, LMI
Suresh Damodaran, Sterling Commerce
Mike DeNicola Fujitsu
Anne Fischer, Drummond Group
Sally Fuger, AIAG
Jong Kim InnoDigital
Kyu-Chul Lee, Chungnam National University
Joel Munter, Intel
Farrukh Najmi, Sun Microsystems
Joel Neu, Vitria Technologies
Sanjay Patil, IONA
Neal Smith, ChevronTexaco
Nikola Stojanovic, Encoda Systems, Inc.
Prasad Yendluri, webMethods
Yutaka Yoshida, Sun Microsystems

2.1 Contributors

The following persons contributed to the content of this document, but are not voting members of the OASIS/ebXML Registry Technical Committee.

Len Gallagher, NIST
Sekhar Vajjhala, Sun Microsystems

Table of Contents

Table of Figures

Table of Tables

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

The following conventions are used throughout this document:

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

The term *"repository item"* is used to refer to an object that has resides in a repository for storage and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the Registry by a RegistryObject instance.

The term "*RegistryEntry*" is used to refer to an object that provides metadata about a *repository item*.

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.

Capitalized Italic words are defined in the ebXML Glossary.

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].

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

3.2.1 Naming Conventions

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

- o Element name is in *UCC* convention
  (example: <UpperCamelCaseElement/>)
- o Attribute name is in *LCC* convention

(example: <UpperCamelCaseElement lowerCamelCaseAttribute="whatEver"/>)
- o *Class*, Interface names use UCC convention (examples: ClassificationNode, Versionable)
- o 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:
- 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 Services Specification [ebRS] - defines the actual *Registry Services* based on this information model
b) ebXML Collaboration-Protocol Profile and Agreement Specification [ebCPP] - defines how profiles can be defined for a *Party* and how two *Parties'* profiles may be used to define a *Party* agreement

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

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, ebXML *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:

- o Defines what types of objects are stored in the *Registry*
- o Defines how stored objects are organized in the *Registry*

5.4 How the Registry Information Model Works

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

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

5.5 Where the Registry Information Model May Be Implemented

The Registry Information Model MAY be implemented within an ebXML *Registry* in 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*.

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.

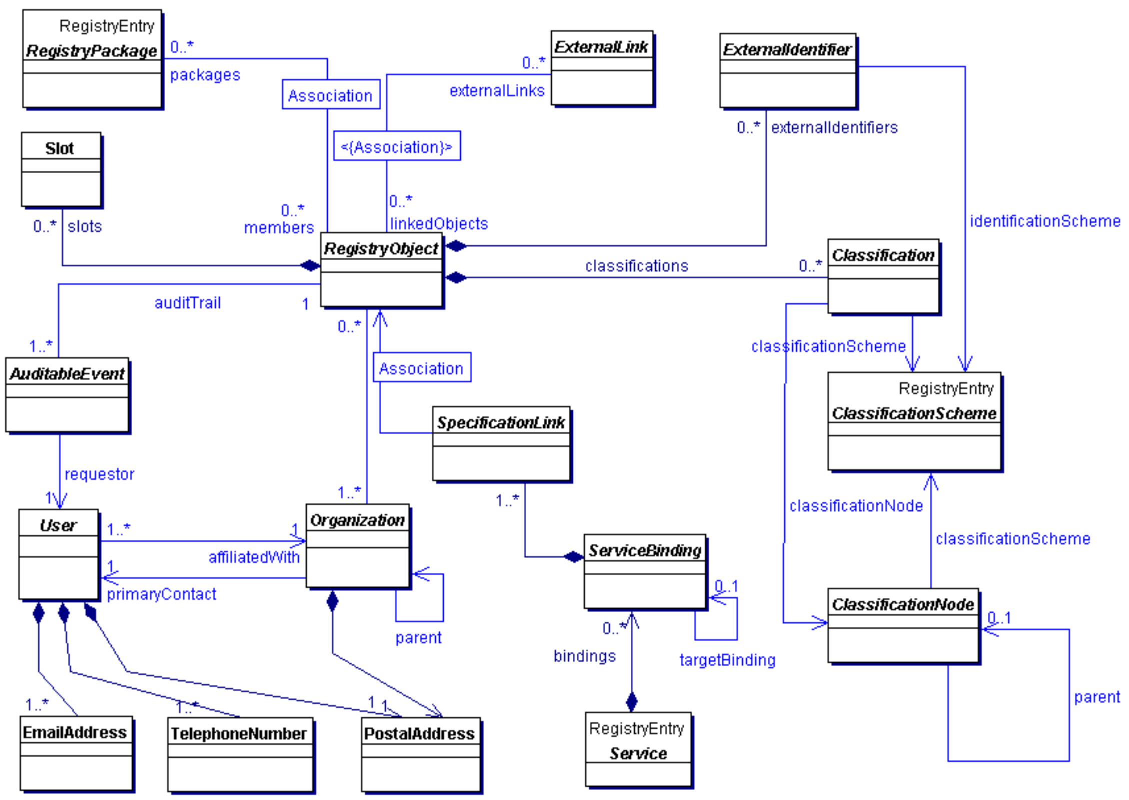

Figure 1: Information Model High Level Public View

6.1 RegistryObject

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

6.2 Slot

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

6.3 Association

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

6.4 ExternalIdentifier

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

6.5 ExternalLink

ExternalLink instances are RegistryObject instances 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*. A RegistryObject instance 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 RegistryObject. The user may click on such links and navigate to an external web page referenced by the link.

6.6 ClassificationScheme

ClassificationScheme instances are RegistryEntry instances that describe a structured way to classify or categorize RegistryObject instances. The structure of the classification scheme may be defined internal or external to the registry, resulting in a distinction between internal and external classification schemes. A very common example of a classification scheme in science is the *Classification of living things* where living things are categorized in a tree like structure. Another example is the Dewey Decimal system used in libraries to categorize books and other publications. ClassificationScheme is described in detail in section 10.

6.7 ClassificationNode

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

6.8 Classification

Classification instances are RegistryObject instances that are used to classify other RegistryObject instances. A Classification instance identifies a ClassificationScheme instance and taxonomy value defined within the classification scheme. Classifications can be internal or external depending on whether the referenced classification scheme is internal or external. Classification is described in detail in section 10.

6.9 RegistryPackage

RegistryPackage instances are RegistryEntry instances that group logically related RegistryObject instances together.

6.10 AuditableEvent

AuditableEvent instances are RegistryObject instances that are used to provide an audit trail for RegistryObject instances. AuditableEvent is described in detail in section 8.

6.11 User

User instances are RegistryObject instances that are used to provide information about registered users within the *Registry*. User objects are used in audit trail for RegistryObject instances. User is described in detail in section 8.

6.12 PostalAddress

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

6.13 EmailAddress

EmailAddress is a simple reusable *Entity Class* that defines a ttributes of an email address.

6.14 Organization

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

6.15 Service

Service instances are RegistryEntry instances that provide information on services (e.g., web services).

6.16 ServiceBinding

ServiceBinding instances are RegistryObject instances that represent technical information on a specific way to access a specific interface offered by a Service instance. A Service has a collection of ServiceBindings.

6.17 SpecificationLink

A SpecificationLink provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service with that ServiceBinding. For example, a ServiceBinding may have a SpecificationLink instance that describes how to access the service using a technical specification in the form of a WSDL document or a CORBA IDL document.

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 class Association will be covered in detail separately in section 9. The classes ClassificationScheme, Classification, and ClassificationNode will be covered in detail separately in section 10.

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

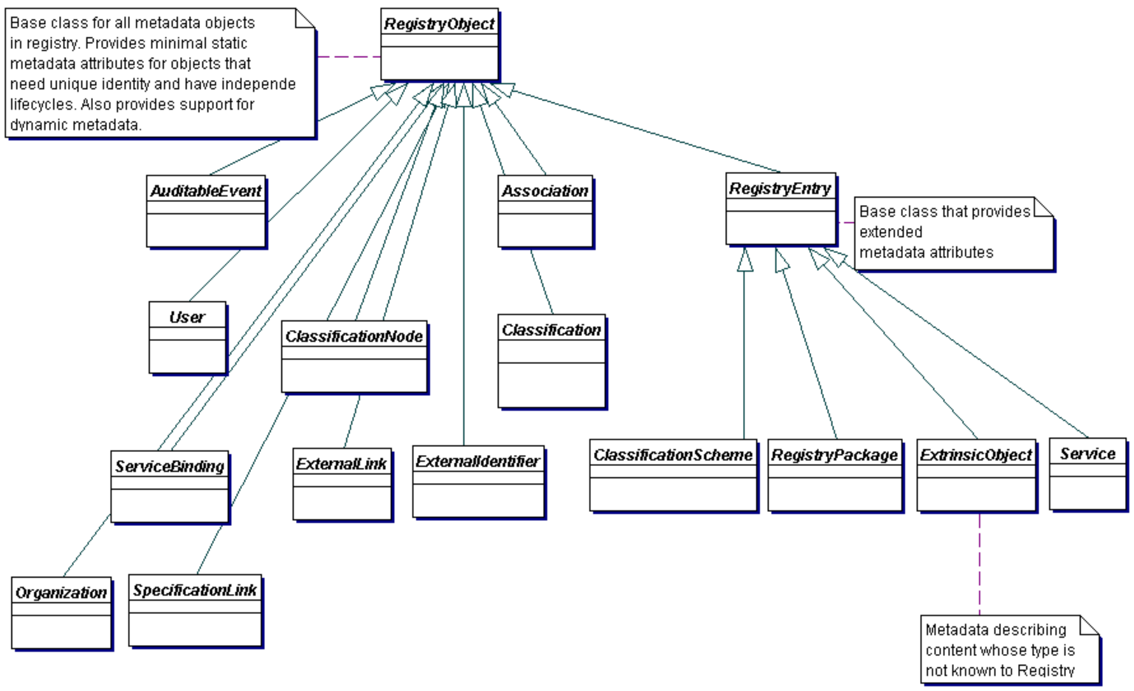
Base class for all metadata objects in registry. Provides minimal static metadata attributes for objects that need unique identity and have independe lifecycles. Also provides support for dynamic metadata.
RegistryObject
AuditableEvent
Association
RegistryEntry
Base class that provides extended metadata attributes
User
ClassificationNode
Classification
ServiceBinding
ExternalLink
ExternalIdentifier
ClassificationScheme
RegistryPackage
ExtrinsicObject
Service
Organization
SpecificationLink
Metadata describing content whose type is not known to Registry

Figure 2: Information Model *Inheritance* View

7.1 Attribute and Methods of Information Model Classes

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

Information model classes may also have methods defined for them. These methods provide additional behavior for the class they are defined within. Methods are currently used in mapping to filter query and the SQL query capabilities defined in [ebRS].

Since the model supports inheritance between classes, it is usually the case that a class in the model inherits attributes and methods from its base classes, in addition to defining its own specialized attributes and methods.

7.2 Data Types

The following table lists the various data types used by the attributes within information model classes:

Data Type	XML Schema Data Type	Description	Length
Boolean	boolean	Used for a true or false value	
String4	string	Used for 4 character long strings	4 characters
String8	string	Used for 8 character long strings	8 characters
String16	string	Used for 16 character long strings	16 characters
String32	string	Used for 32 character long strings	32 characters
String	string	Used for unbounded Strings	unbounded
ShortName	string	A short text string	64 characters
LongName	string	A long text string	128 characters
FreeFormText	string	A very long text string for free-form text	256 characters
UUID	string	DCE 128 Bit Universally unique Ids used for referencing another object	64 characters
URI	string	Used for URL and URN values	256 characters
Integer	integer	Used for integer values	4 bytes
DateTime	dateTime	Used for a timestamp value such as Date	

7.3 Internationalization (I18N) Support

Some information model classes have String attributes that are I18N capable and may be localized into multiple native languages. Examples include the name and description attributes of the RegistryObject class in 7.4.

The information model defines the InternationalString and the LocalizedString interfaces to support I18N capable attributes within the information model classes. These classes are defined below.

7.3.1 Class InternationalString

This class is used as a replacement for the String type whenever a String attribute needs to be I18N capable. An instance of the InternationalString class composes within it a Collection of LocalizedString instances, where each String is specific to a particular locale. The InternationalString class provides set/get methods for adding or getting locale specific String values for the InternationalString instance.

7.3.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
localized-Strings	Collection of Localized-String	No		Client	Yes

7.3.1.2 Attribute localizedStrings

Each InternationalString instance may have localizedString attribute that is a Collection of zero or more LocalizedString instances.

7.3.2 Class LocalizedString

This class is used as a simple wrapper class that associates a String with its locale. The class is needed in the InternationalString class where a Collection of LocalizedString instances are kept. Each LocalizedString instance has a charset and lang attribute as well as a value attribute of type String.

7.3.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
lang	language	No	en-us	Client	Yes
charset	string	No	UTF-8	Client	Yes
value	string	Yes		CLient	Yes

7.3.2.2 Attribute lang

Each LocalizedString instance may have a lang attribute that specifies the language used by that LocalizedString.

7.3.2.3 Attribute charset

Each LocalizedString instance may have a charset attribute that specifies the name of the character set used by that LocalizedString.

7.3.2.4 Attribute value

Each LocalizedString instance must have a value attribute that specifies the string value used by that LocalizedString.

7.4 Class RegistryObject

Direct Known Subclasses:

Association, AuditableEvent, Classification, ClassificationNode, ExternalIdentifier, ExternalLink, Organization, RegistryEntry, User, Service, ServiceBinding, SpecificationLink

---

RegistryObject provides a common base class for almost all objects in the information model. Information model *Classes* whose instances have a unique identity are descendants of the RegistryObject *Class*.

Note that Slot, PostalAddress, and a few other classes 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).

7.4.1 Attribute Summary

The following is the first of many tables that summarize the attributes of a class. The columns in the table are described as follows:

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

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

7.4.2 Attribute accessControlPolicy

Each RegistryObject instance may have an accessControlPolicy instance associated with it. An accessControlPolicy instance defines the *Security Model* associated with the RegistryObject in terms of “who is permitted to do what” with that RegistryObject.

7.4.3 Attribute description

Each RegistryObject instance may have textual description in a human readable and user-friendly manner. This attribute is I18N capable and therefore of type InternationalString.

7.4.4 Attribute id

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

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

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

7.4.5 Attribute name

Each RegistryObject instance may have human readable name. The name does not need to be unique with respect to other RegistryObject instances. This attribute is I18N capable and therefore of type InternationalString.

7.4.6 Attribute objectType

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

7.4.6.1 Pre-defined Object Types

The following table lists pre-defined object types. Note that for an ExtrinsicObject there are many types defined based on the type of repository item the ExtrinsicObject catalogs. In addition there are object types defined for all leaf sub-classes of RegistryObject.

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

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. See [ebCPP] for details.
Process	An ExtrinsicObject of this type catalogues a process description document.
SoftwareComponent	An ExtrinsicObject of this type catalogues a software component (e.g., an EJB or *Class* library).
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.).
RegistryPackage	A RegistryPackage object
ExternalLink	An ExternalLink object
ExternalIdentifier	An ExternalIdentifier object
Association	An Association object
ClassificationScheme	A ClassificationScheme object
Classification	A Classification object
ClassificationNode	A ClassificationNode object
AuditableEvent	An AuditableEvent object
User	A User object
Organization	An Organization object
Service	A Service object
ServiceBinding	A ServiceBinding object
SpecificationLink	A SpecificationLink object

7.4.7 Method Summary

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

Method Summary for RegistryObject	
`Collection`	`getAuditTrail()` Gets the complete audit trail of all requests that effected a state change in this object as an ordered Collection of AuditableEvent objects.
`Collection`	`getClassifications()` Gets the Classification that classify this object.

`Collection`	**getExternalIdentifiers**( ) Gets the collection of ExternalIdentifiers associated with this object.
`Collection`	**getExternalLinks**( ) Gets the ExternalLinks associated with this object.
`Collection`	**getRegistryPackages**( ) Gets the RegistryPackages that this object is a member of.
`Collection`	**getSlots**( ) Gets the Slots associated with this object.

7.5 Class RegistryEntry

Super Classes:
RegistryObject

Direct Known Subclasses:
ClassificationScheme, ExtrinsicObject, RegistryPackage, Service

RegistryEntry is a common base *Class* for classes in the information model that require additional metadata beyond the minimal metadata provided by RegistryObject class. RegistryEntry is used as a base class for high level coarse grained objects in the registry. Their life cycle typically requires more management (e.g. may require approval, deprecation). They typically have relatively fewer instances but serve as a root of a composition hierarchy consisting of numerous objects that are sub-classes of RegistryObject but not RegistryEntry.

The additional metadata is described by the attributes of the RegistryEntry class below.

7.5.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
expiration	DateTime	No		Client	Yes
majorVersion	Integer	Yes	1	Registry	Yes
minorVersion	Integer	Yes	0	Registry	Yes
stability	LongName	No		Client	Yes
status	LongName	Yes		Registry	Yes
userVersion	ShortName	No		Client	Yes

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

7.5.2 Attribute expiration

Each RegistryEntry instance may have an expirationDate. This attribute defines a time limit upon the stability indication provided by the stability attribute. Once the expirationDate has been reached the stability attribute in effect becomes STABILITY_DYNAMIC implying that the repository item can change at any time and in any manner. A null value implies that there is no expiration on stability attribute.

7.5.3 Attribute majorVersion

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

7.5.4 Attribute minorVersion

Each RegistryEntry instance must have a minor revision number for the current version of the RegistryEntry instance. This number is assigned by the registry when the object is created. This number may be updated by the registry when an object is updated.

7.5.5 Attribute stability

Each RegistryEntry instance may have a stability indicator. The stability indicator is provided by the submitter as an indication of the level of stability for the repository item.

7.5.5.1 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 *ClassificationScheme*. 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.

7.5.6 Attribute status

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

7.5.6.1 Pre-defined RegistryObject Status Types

The following table lists pre-defined choices for RegistryObject status attribute. These pre-defined status types are defined as a *ClassificationScheme*.

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

7.5.7 Attribute userVersion

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

7.6 Class Slot

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

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

7.6.1 Attribute Summary

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

7.6.2 Attribute name

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

7.6.3 Attribute slotType

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

7.6.4 Attribute values

A Slot instance must have a Collection of values. The collection of values may be empty. 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.

7.7 Class ExtrinsicObject

Super Classes:

RegistryEntry, RegistryObject

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).

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

Examples of content described by ExtrinsicObject include *Collaboration Protocol Profiles* [eb*CPP*], *Business Process* descriptions, and schemas.

7.7.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isOpaque	Boolean	No		Client	No
mimeType	LongName	No		Client	No

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

7.7.2 Attribute isOpaque

Each ExtrinsicObject instance may have an isOpaque attribute defined. This attribute 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*.

7.7.3 Attribute mimeType

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

7.8 Class RegistryPackage

Super Classes:

RegistryEntry, RegistryObject

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

7.8.1 Attribute Summary

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

7.8.2 Method Summary

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

Method Summary of RegistryPackage	
`Collection`	`getMemberObjects()` Get the collection of RegistryObject instances that are members of this RegistryPackage.

7.9 Class ExternalIdentifier

Super Classes:

RegistryObject

ExternalIdentifier instances provide the additional identifier information to RegistryObject such as DUNS number, Social Security Number, or an alias name of the organization. The attribute *identificationScheme* is used to reference the identification scheme (e.g., “DUNS”, “Social Security #”), and the attribute *value* contains the actual information (e.g., the DUNS number, the social security number). Each RegistryObject may contain 0 or more ExternalIdentifier instances.

7.9.1 Attribute Summary

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

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

7.9.2 Attribute identificationScheme

Each ExternalIdentifier instance must have an identificationScheme attribute that references a ClassificationScheme. This ClassificationScheme defines the namespace within which an identifier is defined using the value attribute for the RegistryObject referenced by the RegistryObject attribute.

7.9.3 Attribute registryObject

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

7.9.4 Attribute value

Each ExternalIdentifier instance must have a value attribute that provides the identifier value for this ExternalIdentifier (e.g., the actual social security number).

7.10 Class ExternalLink

Super Classes:

RegistryObject

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.

7.10.1 Attribute Summary

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

7.10.2 Attribute externalURI

Each ExternalLink instance must have an externalURI attribute defined. The externalURI attribute provides a URI to the external resource pointed to by this ExternalLink instance. If the URI is a URL then a registry must validate the URL to be resolvable at the time of submission before accepting an ExternalLink submission to the registry.

7.10.3 Method Summary

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

Method Summary of ExternalLink	
`Collection`	`getLinkedObjects()` Gets the collection of RegistryObjects that are linked by this ExternalLink to content outside the registry.

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. They are analogous to the “struct” construct in the C programming language.

The getAuditTrail() method of a RegistryObject returns an ordered Collection of AuditableEvents. These AuditableEvents constitute the audit trail for the RegistryObject. 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 Class AuditableEvent

Super Classes:

RegistryObject

AuditableEvent instances provide a long-term record of *Events* that effect a change in a RegistryObject. A RegistryObject 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 RegistryObject. For example a client request could Create, Update, Deprecate or Delete a

RegistryObject. An AuditableEvent is created if and only if a request creates or alters the content or ownership of a RegistryObject. Read-only requests do not generate an AuditableEvent. No AuditableEvent is generated for a RegistryObject when it is classified, assigned to a RegistryPackage or associated with another RegistryObject.

8.1.1 Attribute Summary

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

8.1.2 Attribute eventType

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

8.1.2.1 Pre-defined Auditable Event Types

The following table lists pre-defined auditable event types. These pre-defined event types are defined as a pre-defined *ClassificationScheme* with name "EventType". A *Registry* MUST support the event types listed below.

Name	description
Created	An *Event* that created a RegistryObject.
Deleted	An *Event* that deleted a RegistryObject.
Deprecated	An *Event* that deprecated a RegistryObject.
Updated	An *Event* that updated the state of a RegistryObject.
Versioned	An *Event* that versioned a RegistryObject.

8.1.3 Attribute registryObject

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

8.1.4 Attribute timestamp

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

8.1.5 Attribute user

Each AuditableEvent must have a user attribute that identifies the User that sent the request that generated this event affecting the RegistryObject instance.

8.2 Class User

Super Classes:

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.

8.2.1 Attribute Summary

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

8.2.2 Attribute address

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

8.2.3 Attribute emailAddresses

Each User instance has an attribute emailAddresses that is a Collection of EmailAddress instances. Each EmailAddress provides an email address for that user. A User must have at least one email address.

8.2.4 Attribute organization

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

8.2.5 Attribute personName

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

8.2.6 Attribute telephoneNumbers

Each User instance must have a telephoneNumbers attribute that contains the Collection of TelephoneNumber instances for each telephone number defined for that user. A User must have at least one telephone number.

8.2.7 Attribute url

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

8.3 Class Organization

Super Classes:

RegistryObject

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

8.3.1 Attribute Summary

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

8.3.2 Attribute address

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

8.3.3 Attribute parent

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

8.3.4 Attribute primaryContact

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

8.3.5 Attribute telephoneNumbers

Each Organization instance must have a te lephoneNumbers attribute that contains the Collection of TelephoneNumber instances for each telephone number defined for that organization. An Organization must have at least one telephone number.

8.4 Class PostalAddress

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

8.4.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
city	ShortName	No		Client	Yes
country	ShortName	No		Client	Yes
postalCode	ShortName	No		Client	Yes
state	ShortName	No		Client	Yes
street	ShortName	No		Client	Yes
streetNumber	String32	No		Client	Yes

8.4.2 Attribute city

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

8.4.3 Attribute country

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

8.4.4 Attribute postalCode

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

8.4.5 Attribute state

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

8.4.6 Attribute street

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

8.4.7 Attribute streetNumber

Each PostalAddress may have a streetNumber attribute identifying the street number (e.g., 65) for the street address.

8.4.8 Method Summary

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

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

8.5 Class TelephoneNumber

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

8.5.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
areaCode	String4	No		Client	Yes
countryCode	String4	No		Client	Yes
extension	String8	No		Client	Yes
number	String16	No		Client	Yes
phoneType	String32	No		Client	Yes
url	URI	No		Client	Yes

8.5.2 Attribute areaCode

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

8.5.3 Attribute countryCode

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

8.5.4 Attribute extension

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

8.5.5 Attribute number

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

8.5.6 Attribute phoneType

Each TelephoneNumber instance may have phoneType attribute that provides the type for the TelephoneNumber. Some examples of phoneType are “home”, “office”.

8.6 Class EmailAddress

A simple reusable *Entity Class* that defines attributes of an email address.

8.6.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
address	ShortName	Yes		Client	Yes
type	String32	No		Client	Yes

8.6.2 Attribute address

Each EmailAddress instance must have an address attribute that provides the actual email address.

8.6.3 Attribute type

Each EmailAddress instance may have a type attribute that provides the type for that email address. This is an arbitrary value. Examples include “home”, “work” etc.

8.7 Class PersonName

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

8.7.1 Attribute Summary

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

8.7.2 Attribute firstName

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

8.7.3 Attribute lastName

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

8.7.4 Attribute middleName

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

8.8 Class Service

Super Classes:

RegistryEntry, RegistryObject

Service instances provide information on services, such as web services.

8.8.1 Attribute Summary

The Service class does not define any specialized attributes other than its inherited attributes.

8.8.2 Method Summary

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

Method Summary of Service	
`Collection`	**getServiceBindings**() Gets the collection of ServiceBinding instances defined for this Service.

8.9 Class ServiceBinding

Super Classes:

RegistryObject

ServiceBinding instances are RegistryObjects that represent technical information on a specific way to access a specific interface offered by a Service instance. A Service has a Collection of ServiceBindings.
The description attribute of ServiceBinding provides details about the relationship between several specification links comprising the Service Binding. This description can be useful for human understanding such that the runtime system can be appropriately configured by the human being. There is possibility of enforcing a structure on this description for enabling machine processing of the Service Binding, which is however not addressed by the current document.

8.9.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
accessURI	URI	No		Client	Yes
targetBinding	UUID	No		Client	Yes

8.9.2 Attribute accessURI

A ServiceBinding may have an accessURI attribute that defines the URI to access that ServiceBinding. This attribute is ignored if a targetBinding attribute is specified for the ServiceBinding. If the URI is a URL then a registry must validate the URL to be resolvable at the time of submission before accepting a ServiceBinding submission to the registry.

8.9.3 Attribute targetBinding

A ServiceBinding may have a targetBinding attribute defined which references another ServiceBinding. A targetBinding may be specified when a service is being redirected to another service. This allows the rehosting of a service by another service provider.

8.9.4 Method Summary

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

Method Summary of ServiceBinding	
`Collection`	`getSpecificationLinks()` Get the collection of SpecificationLink instances defined for this ServiceBinding.

8.10 Class SpecificationLink

Super Classes:

RegistryObject

A SpecificationLink provides the linkage between a ServiceBinding and one of its technical specifications that describes how to use the service using the ServiceBinding. For example, a ServiceBinding may have a SpecificationLink instances that describe how to access the service using a technical specification in form of a WSDL document or a CORBA IDL document.

8.10.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
specificationObject	UUID	Yes		Client	Yes
usageDescription	InternationalString	No		Client	Yes
usageParameters	Collection of FreeFormText	No		Client	Yes

8.10.2 Attribute specificationObject

A SpecificationLink instance must have a specificationObject attribute that provides a reference to a RegistryObject instance that provides a technical specification for the parent ServiceBinding. Typically, this is an ExtrinsicObject instance representing the technical specification (e.g., a WSDL document).

8.10.3 Attribute usageDescription

A SpecificationLink instance may have a usageDescription attribute that provides a textual description of how to use the optional usageParameters attribute described next. The usageDescription is of type InternationalString, thus allowing the description to be in multiple languages.

8.10.4 Attribute usageParameters

A SpecificationLink instance may have a usageParameters attribute that provides a collection of Strings representing the instance specific parameters needed to use the technical specification (e.g., a WSDL document) specified by this SpecificationLink object.

9 Association of Registry Objects

A RegistryObject instance may be *associated* with zero or more RegistryObject instances. The information model defines an Association class, an instance of which may be used to associate any two RegistryObject instances.

9.1 Example of an Association

One example of such an association is between two ClassificationScheme instances, where one ClassificationScheme supersedes the other ClassificationScheme as shown in Figure 3. This may be the case when a new version of a ClassificationScheme is submitted.

In Figure 3, we see how an Association is defined between a new version of the NAICS ClassificationScheme and an older version of the NAICS ClassificationScheme.

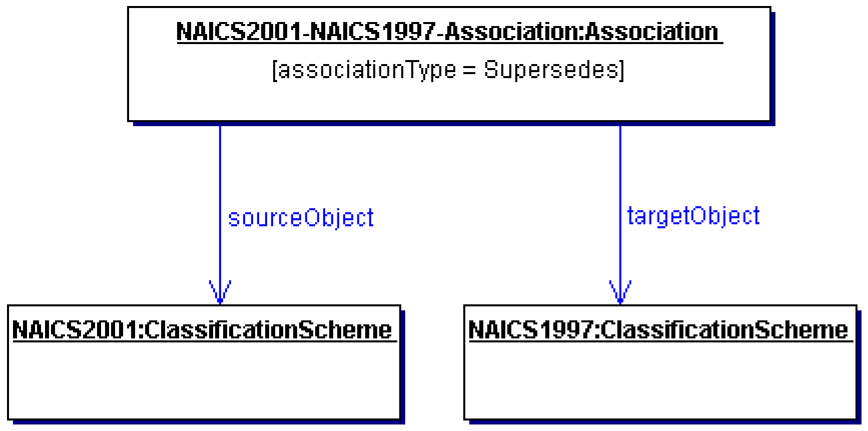
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme

Figure 3: Example of RegistryObject Association

9.2 Source and Target Objects

An Association instance represents an association between a *source* RegistryObject and a *target* RegistryObject. These are referred to as *sourceObject* and *targetObject* for the Association instance. It is important which object is the sourceObject and which is the targetObject as it determines the directional semantics of an Association.

In the example in Figure 3, it is important to make the newer version of NAICS ClassificationScheme be the sourceObject and the older version of NAICS be the targetObject because the associationType implies that the sourceObject supersedes the targetObject (and not the other way around).

9.3 Association Types

Each Association must have an associationType attribute that identifies the type of that association.

9.4 Intramural Association

A common use case for the Association class is when a User “u” creates an Association “a” between two RegistryObjects “o1” and “o2” where association “a” and RegistryObjects “o1” and “o2” are objects that were created by the same User “u.” This is the simplest use case, where the association is between two objects that are owned by the same User that is defining the Association. Such associations are referred to as *intramural associations*.
Figure 4 below, extends the previous example in Figure 3 for the intramural association case.

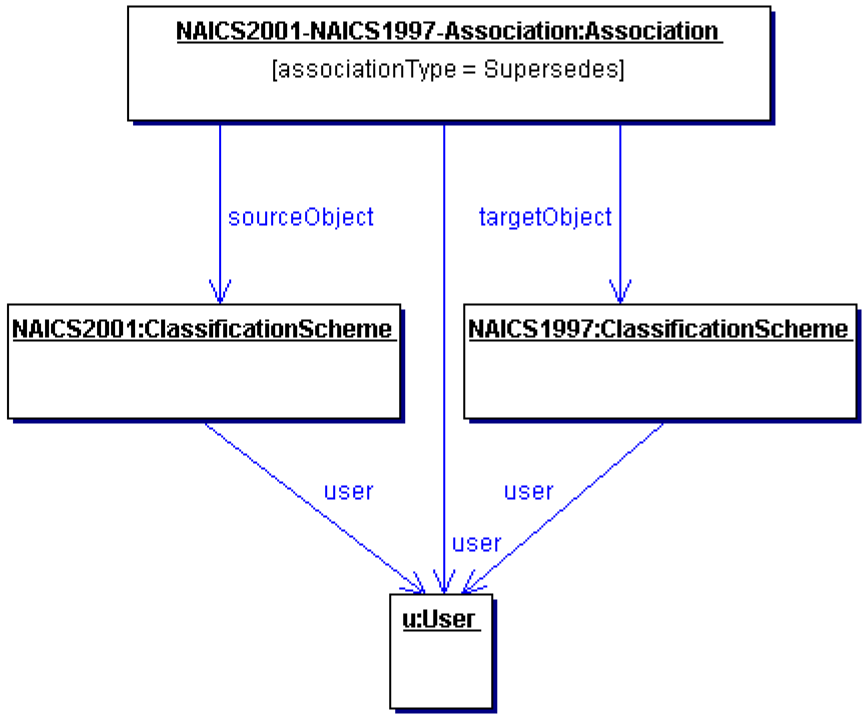
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme
user
user
user
u:User

Figure 4: Example of Intramural Association

9.5 Extramural Association

The information model also allows more sophisticated use cases. For example, a User “u1” creates an Association “a” between two RegistryObjects “o1” and “o2” where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2” are owned by User “u2” and User “u3” respectively.
In this use case an Association is defined where either or both objects that are being associated are owned by a User different from the User defining the Association. Such associations are referred to as *extramural associations*. The Association class provides a convenience method called `isExtramural` that returns "true" if the Association instance is an extramural Association.

Figure 5 below, extends the previous example in Figure 3 for the extramural association case. Note that it is possible for an extramural association to have two distinct Users rather than three distinct Users as shown in Figure 5. In such case, one of the two users owns two of the three objects involved (Association, sourceObject and targetObject).

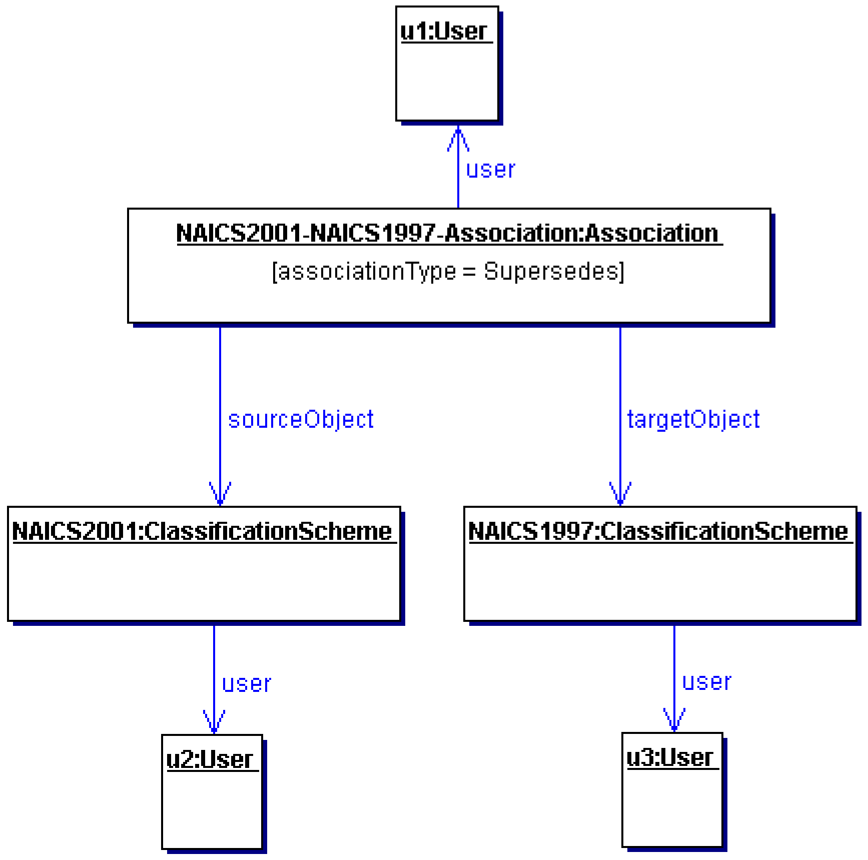
u1:User
user
NAICS2001-NAICS1997-Association:Association
[associationType = Supersedes]
sourceObject
targetObject
NAICS2001:ClassificationScheme
NAICS1997:ClassificationScheme
user
user
u2:User
u3:User

Figure 5: Example of Extramural Association

9.6 Confirmation of an Association

An association may need to be confirmed by the parties whose objects are involved in that Association as the sourceObject or targetObject. This section describes the semantics of confirmation of an association by the parties involved.

9.6.1 Confirmation of Intramural Associations

Intramural associations may be viewed as declarations of truth and do not require any explicit steps to confirm that Association as being true. In other words, intramural associations are implicitly considered confirmed.

9.6.2 Confirmation of Extramural Associations

Extramural associations may be thought of as a unilateral assertion that may not be viewed as truth until it has been confirmed by the other (extramural) parties involved (Users “u2” and “u3” in the example in section 9.5).
To confirm an extramural association, each of the extramural parties (parties that own the source or target object but do not own the Association) must submit an identical Association (clone Association) as the Association they are intending to confirm using a SubmitObjectsRequest. The clone Association must have the same id as the original Association.

9.6.3 Deleting an Extramural Associations

An Extramural Association is deleted like any other type of RegistryObject, using the RemoveObjectsRequest as defined in [ebRS]. However, in some cases deleting an extramural Association may not actually delete it but instead only revert a confirmed association to unconfirmed state.

An Association must always be deleted when deleted by the owner of that Association, irrespective of its confirmation state. An extramural Association must become unconfirmed by the owner of its source/target object when deleted by the owner of its source/target object when the requestor is not the owner of the Association itself.

9.7 Visibility of Unconfirmed Associations

Extramural associations require each extramural party to confirm the assertion being made by the extramural Association before the Association is visible to third parties that are not involved in the Association. This ensures that unconfirmed Associations are not visible to third party registry clients.

9.8 Possible Confirmation States

Assume the most general case where there are three distinct User instances as shown in Figure 5 for an extramural Association. The extramural Association needs to be confirmed by both the other (extramural) parties (Users “u2” and “u3” in example) in order to be fully confirmed. The methods `isConfirmedBySourceOwner` and `isConfirmedByTargetOwner` in the Association class provide access to the confirmation state for both the sourceObject and targetObject. A third convenience method called `isConfirmed` provides a way to determine whether the Association is fully confirmed or not. So there are the following four possibilities related to the confirmation state of an extramural Association:

- The Association is confirmed neither by the owner of the sourceObject nor by the owner of the targetObject.
- The Association is confirmed by the owner of the sourceObject but it is not confirmed by the owner of the targetObject.
- The Association is not confirmed by the owner of the sourceObject but it is confirmed by the owner of the targetObject.

- o The Association is confirmed by both the owner of the sourceObject and the owner of the targetObject. This is the only state where the Association is fully confirmed.

9.9 Class *Association*

Super Classes:

RegistryObject

Association instances are used to define many-to-many associations among RegistryObjects in the information model.

An *Instance* of the Association *Class* represents an association between two RegistryObjects.

9.9.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
associationType	LongName	Yes		Client	No
sourceObject	UUID	Yes		Client	No
targetObject	UUID	Yes		Client	No
IsConfirmedBy-SourceOwner	boolean	No	false	Registry	No
IsConfirmedBy-TargetOwner	boolean	No	false	Registry	No

9.9.2 Attribute associationType

Each Association must have an associationType attribute that identifies the type of that association.

9.9.2.1 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 RegistryPackage object has the target RegistryObject object as a member. Reserved for use in Packaging of RegistryEntries.

`ExternallyLinks`	Defines that the source ExternalLink object externally links the target RegistryObject object. Reserved for use in associating ExternalLinks with RegistryEntries.
`Contains`	Defines that source RegistryObject contains the target RegistryObject. The details of the containment relationship are specific to the usage. For example a parts catalog may define an Engine object to have a contains relationship with a Transmission object.
`EquivalentTo`	Defines that source RegistryObject is equivalent to the target RegistryObject.
`Extends`	Defines that source RegistryObject inherits from or specializes the target RegistryObject.
`Implements`	Defines that source RegistryObject implements the functionality defined by the target RegistryObject.
`InstanceOf`	Defines that source RegistryObject is an *Instance* of target RegistryObject.
`Supersedes`	Defines that the source RegistryObject supersedes the target RegistryObject.
`Uses`	Defines that the source RegistryObject uses the target RegistryObject in some manner.
`Replaces`	Defines that the source RegistryObject replaces the target RegistryObject in some manner.
`SubmitterOf`	Defines that the source Organization is the submitter of the target RegistryObject.
`ResponsibleFor`	Defines that the source Organization is responsible for the ongoing maintainence of the target RegistryObject.
`OffersService`	Defines that the source Organization object offers the target Service object as a service. Reserved for use in indicating that an Organization offers a Service.

9.9.3 Attribute sourceObject

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

9.9.4 Attribute targetObject

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

9.9.5 Attribute isConfirmedBySourceOwner

Each Association may have an isConfirmedBySourceOwner attribute that is set by the registry to be true if the association has been confirmed by the owner of the sourceObject. For intramural Associations this attribute is always true. This attribute must be present when the object is retrieved from the registry. This attribute must be ignored if specified by the client when the object is submitted to the registry.

9.9.6 Attribute isConfirmedByTargetOwner

Each Association may have an isConfirmedByTargetOwner attribute that is set by the registry to be true if the association has been confirmed by the owner of the targetObject. For intramural Associations this attribute is always true. This attribute must be present when the object is retrieved from the registry. This attribute must be ignored if specified by the client when the object is submitted to the registry.

Method Summary of Association	
`Boolean`	`isConfirmed()` Returns true if isConfirmedBySourceOwner and isConfirmedByTargetOwner attributes are both true. For intramural Associations always return true. An association should only be visible to third parties (not involved with the Association) if isConfirmed returns true.
`Boolean`	`isExtramural()` Returns true if the sourceObject and/or the targetObject are owned by a User that is different from the User that created the Association.

10 Classification of RegistryObject

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

A RegistryObject may be classified in many ways. For example the RegistryObject 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 *ClassificationScheme* can be viewed as a *Classification* tree. In the example shown in Figure 6, RegistryObject instances 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 ClassificationScheme instance with name “Industry.” Furthermore, the US Automobile manufacturers are classified by the US ClassificationNode under the ClassificationScheme with name “Geography." Similarly, a European automobile manufacturer is classified by the “Europe” ClassificationNode under the ClassificationScheme with name “Geography.”

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

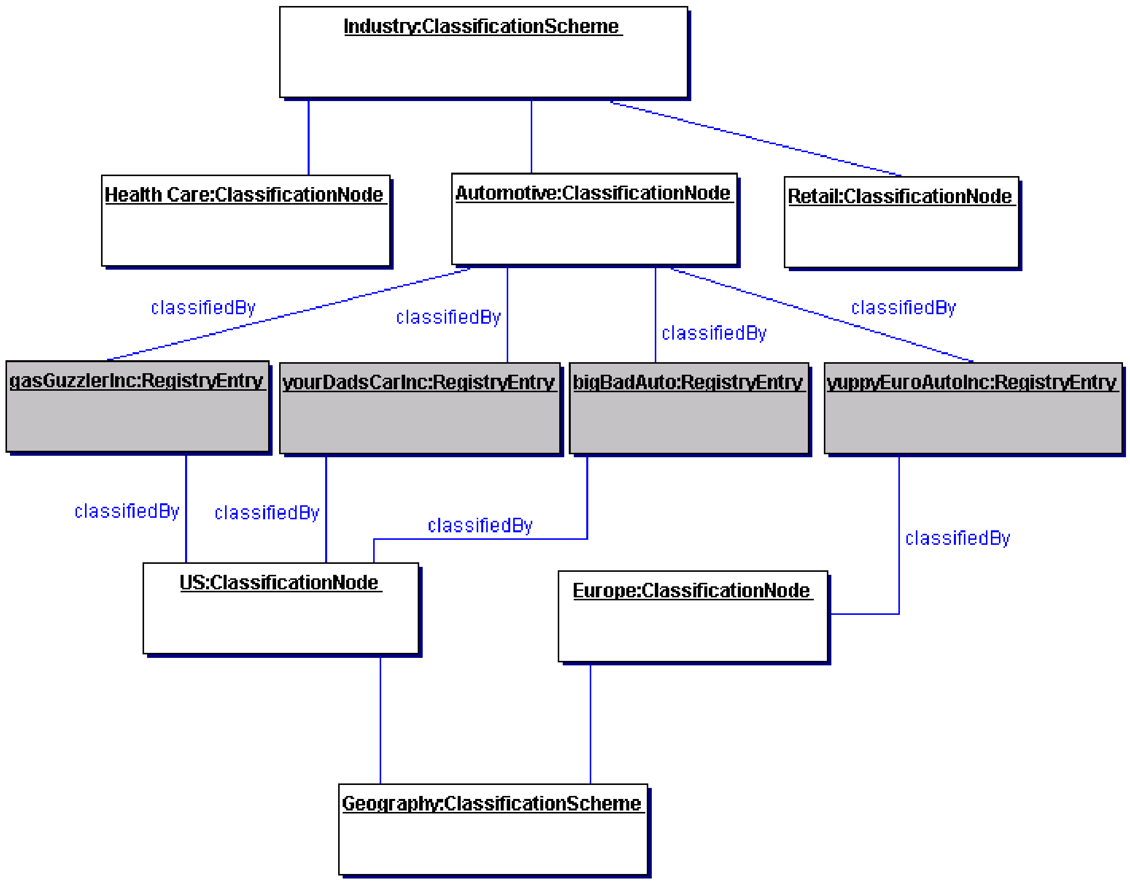
Industry:ClassificationScheme
Health Care:ClassificationNode
Automotive:ClassificationNode
Retail:ClassificationNode
classifiedBy
classifiedBy
classifiedBy
classifiedBy
gasGuzzlerInc:RegistryEntry
yourDadsCarInc:RegistryEntry
bigBadAuto:RegistryEntry
yuppyEuroAutoInc:RegistryEntry
classifiedBy
classifiedBy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

Figure 6: 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 7.

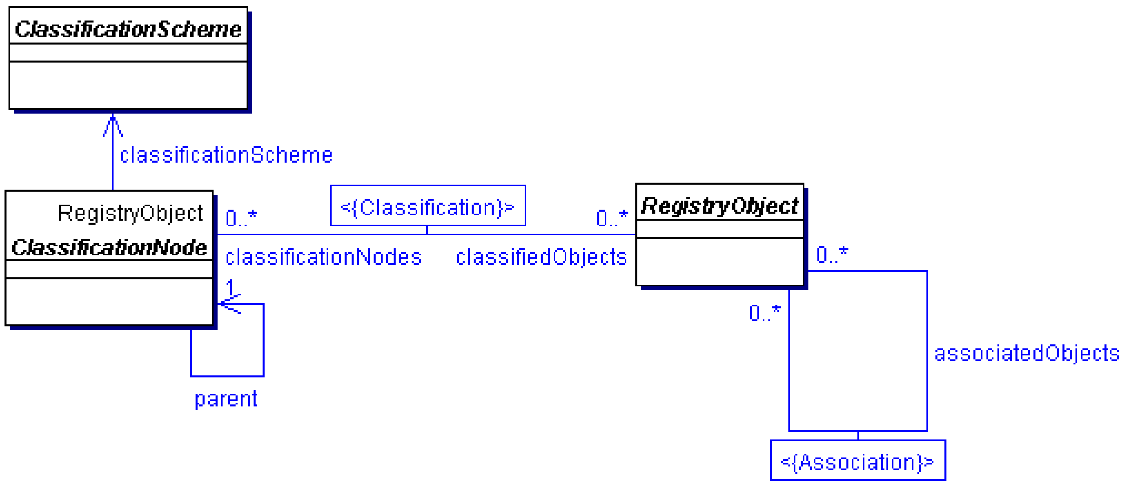

Figure 7: Information Model *Classification* View

A Classification is somewhat like a specialized form of an Association. Figure 8 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.

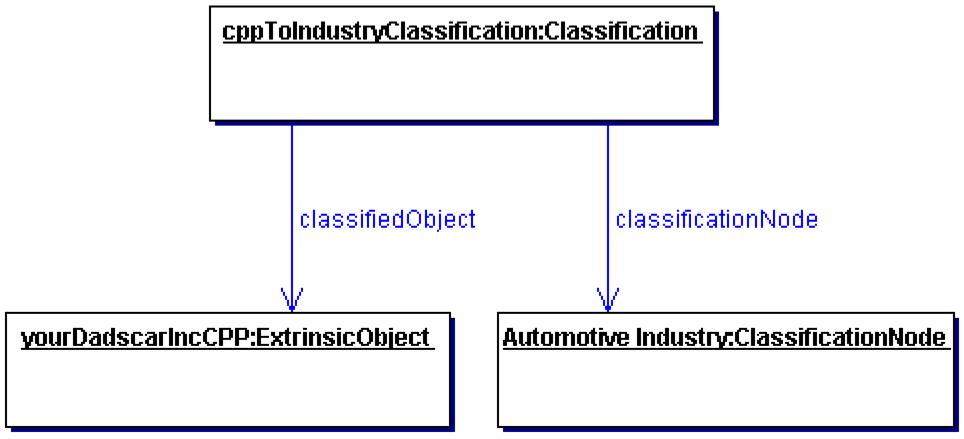
cppToIndustryClassification:Classification
classifiedObject
classificationNode
yourDadscarIncCPP:ExtrinsicObject
Automotive Industry:ClassificationNode

Figure 8: Classification *Instance* Diagram

10.1 Class ClassificationScheme

Base classes:

RegistryEntry, RegistryObject

A ClassificationScheme instance is metadata that describes a registered taxonomy. The taxonomy hierarchy may be defined internally to the Registry by instances of ClassificationNode or it may be defined externally to the Registry, in which case the structure and values of the taxonomy elements are not known to the Registry.

In the first case the classification scheme is defined to be *internal* and in the second case the classification scheme is defined to be *external*.

The ClassificationScheme class inherits attributes and methods from the RegistryObject and RegistryEntry classes.

10.1.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
isInternal	Boolean	Yes		Client	No
nodeType	String32	Yes		Client	No

Note that attributes inherited by ClassificationScheme class from the RegistryEntry class are not shown.

10.1.2 Attribute isInternal

When submitting a ClassificationScheme instance the Submitting Organization needs to declare whether the ClassificationScheme instance represents an internal or an external taxonomy. This allows the registry to validate the subsequent submissions of ClassificationNode and Classification instances in order to maintain the type of ClassificationScheme consistent throughout its lifecycle.

10.1.3 Attribute nodeType

When submitting a ClassificationScheme instance the Submitting Organization needs to declare what is the structure of taxonomy nodes that this ClassificationScheme instance will represent. This attribute is an enumeration with the following values:

- UniqueCode. This value says that each node of the taxonomy has a unique code assigned to it.
- EmbeddedPath. This value says that a unique code assigned to each node of the taxonomy at the same time encodes its path. This is the case in the NAICS taxonomy.

- NonUniqueCode. In some cases nodes are not unique, and it is necessary to nominate the full path in order to identify the node. For example, in a geography taxonomy Moscow could be under both Russia and the USA, where there are five cities of that name in different states.

This enumeration might expand in the future with some new values. An example for possible future values for this enumeration might be NamedPathElements for support of Named-Level taxonomies such as Genus/Species.

10.2 Class ClassificationNode

Base classes:

RegistryObject

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

10.2.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
parent	UUID	No		Client	No
code	ShortName	No		Client	No
path	String	No		Registry	No

10.2.2 Attribute parent

Each ClassificationNode may have a parent attribute. The parent attribute either references a parent ClassificationNode or a ClassificationScheme instance in case of first level ClassificationNode instances.

10.2.3 Attribute code

Each ClassificationNode may have a code attribute. The code attribute contains a code within a standard coding scheme.

10.2.4 Attribute path

Each ClassificationNode may have a path attribute. The path attribute must be present when a ClassificationNode is retrieved from the registry. The path attribute must be ignored when the path is specified by the client when the object is submitted to the registry. The path attribute contains the canonical path from the ClassificationScheme of this ClassificationNode. The path syntax is defined in 10.2.6.

10.2.5 Method Summary

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

Method Summary of ClassificationNode	
ClassificationScheme	**getClassificationScheme**() Get the ClassificationScheme that this ClassificationNode belongs to.
Collection	**getClassifiedObjects**() Get the collection of RegistryObjects classified by this ClassificationNode.
Integer	**getLevelNumber()** Gets the level number of this ClassificationNode in the classification scheme hierarchy. This method returns a positive integer and is defined for every node instance.

In Figure 6, several instances of ClassificationNode are defined (all light colored boxes). A ClassificationNode has zero or one parent and zero or more ClassificationNodes for its immediate children. The parent of a ClassificationNode may be another ClassificationNode or a ClassificationScheme in case of first level ClassificationNodes.

10.2.6 Canonical Path Syntax

The path attribute of the ClassificationNode class contains an absolute path in a canonical representation that uniquely identifies the path leading from the ClassificationScheme to that ClassificationNode.
The canonical path representation is defined by the following BNF grammar:

```
canonicalPath ::= '/' schemeId nodePath
nodePath      ::=     '/' nodeCode
              |       '/' nodeCode ( nodePath )?
```

In the above grammar, schemeId is the id attribute of the ClassificationScheme instance, and nodeCode is defined by NCName production as defined by http://www.w3.org/TR/REC-xml-names/#NT-NCName.

10.2.6.1 Example of Canonical Path Representation

The following canonical path represents what the path attribute would contain for the ClassificationNode with code ‘United States’ in the sample Geography scheme in section 10.2.6.2.

```
/Geography-id/NorthAmerica/UnitedStates
```

10.2.6.2 Sample Geography Scheme

Note that in the following examples, the ID attributes have been chosen for ease of readability and are therefore not valid URN or UUID values.

```
<ClassificationScheme id='Geography-id' name="Geography"/>

<ClassificationNode id="NorthAmerica-id" parent="Geography-id" code=NorthAmerica" />
<ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />

<ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
<ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
<ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

10.3 Class Classification

Base Classes:

RegistryObject

A Classification instance classifies a RegistryObject instance by referencing a node defined within a particular classification scheme. An internal classification will always reference the node directly, by its id, while an external classification will reference the node indirectly by specifying a representation of its value that is unique within the external classification scheme.

The attributes and methods for the Classification class are intended to allow for representation of both internal and external classifications in order to minimize the need for a submission or a query to distinguish between internal and external classifications.

In Figure 6, Classification instances are not explicitly shown but are implied as associations between the RegistryObject instances (shaded leaf node) and the associated ClassificationNode.

10.3.1 Attribute Summary

Attribute	Data Type	Required	Default Value	Specified By	Mutable
classificationScheme	UUID	for external classifications	null	Client	No
classificationNode	UUID	for internal	null	Client	No

		classifications			
classifiedObject	UUID	Yes		Client	No
nodeRepresentation	LongName	for external classifications	null	Client	No

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

10.3.2 Attribute classificationScheme

If the Classification instance represents an external classification, then the classificationScheme attribute is required. The classificationScheme value must reference a ClassificationScheme instance.

10.3.3 Attribute classificationNode

If the Classification instance represents an internal classification, then the classificationNode attribute is required. The classificationNode value must reference a ClassificationNode instance.

10.3.4 Attribute classifiedObject

For both internal and external classifications, the ClassifiedObject attribute is required and it references the RegistryObject instance that is classified by this Classification.

10.3.5 Attribute nodeRepresentation

If the Classification instance represents an external classification, then the nodeRepresentation attribute is required. It is a representation of a taxonomy element from a classification scheme. It is the responsibility of the registry to distinguish between different types of nodeRepresentation, like between the classification scheme node code and the classification scheme node canonical path. This allows client to transparently use different syntaxes for nodeRepresentation.

10.3.6 Context Sensitive *Classification*

Consider the case depicted in Figure 9 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.

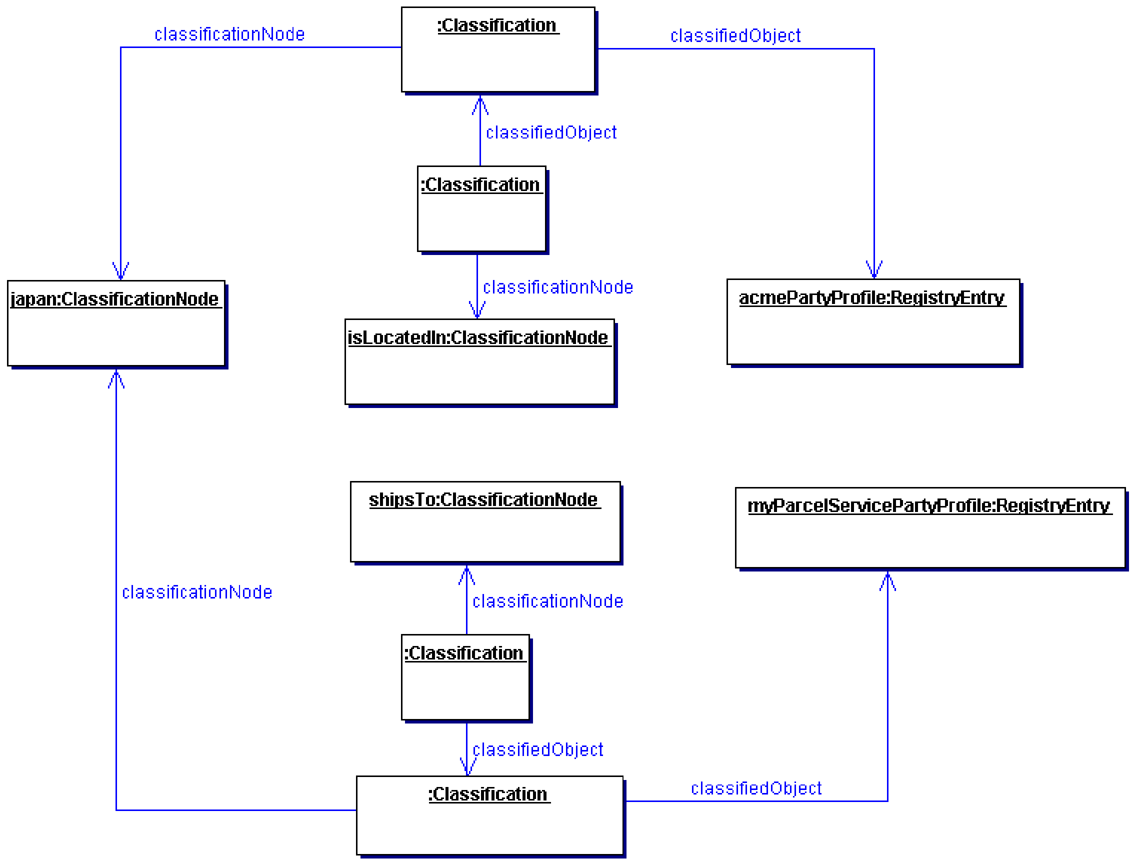

Figure 9: Context Sensitive *Classification*

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:

- A RegistryObject to be classified by defining an internal Classification that associates it with a ClassificationNode in a *ClassificationScheme*.
- A RegistryObject to be classified by defining an external Classification that associates it with a value in an external *ClassificationScheme*.
- A RegistryObject to be classified along multiple facets by having multiple *Classifications* that associate it with multiple ClassificationNodes or value within a ClassificationScheme.
- A *Classification* defined for a RegistryObject to be qualified by the contexts in which it is being classified.

10.3.7 Method Summary

In addition to its attributes, the Classification class also defines the following methods:

Return Type	Method
`UUID`	getClassificationScheme() For an external classification, returns the scheme identified by the classificationScheme attribute. For an internal classification, returns the scheme identified by the same method applied to the ClassificationNode instance
`String`	getPath() For an external classification returns a string that conforms to the canonical path syntax as specified in 10.2.6. For an internal classification, returns the value contained in the path attribute of the ClassificationNode instance identified by the classificationNode attribute.
`ShortName`	getCode() For an external classification, returns a string that represents the declared value of the taxonomy element. It will not necessarily uniquely identify that node. For an internal classification, returns the value of the code attribute of the ClassificationNode instance identified by the classificationNode attribute.

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

Table 1: Sample *Classification* Schemes

11 Information Model: Security View

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

Figure 10 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.

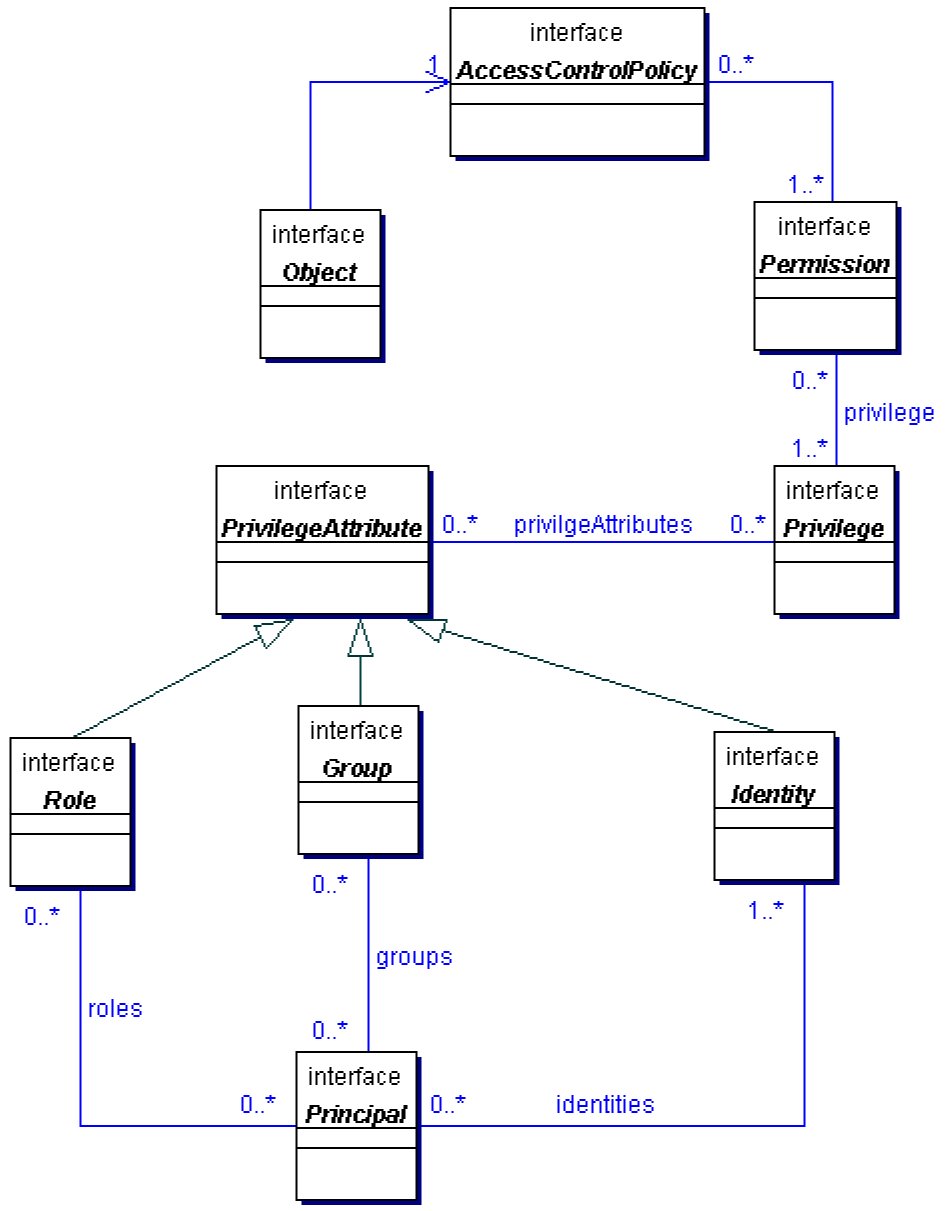

Figure 10: Information Model: Security View

11.1 Class AccessControlPolicy

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

Method Summary of AccessControlPolicy	
`Collection`	**`getPermissions`**`()` Gets the Permissions defined for this AccessControlPolicy. Maps to attribute named `permissions`.

11.2 Class 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`.

11.3 Class Privilege

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

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

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

See Also:

`PrivilegeAttribute`, `Permission`

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

11.4 Class PrivilegeAttribute

All Known Subclasses:

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`

11.5 Class Role

All Superclasses:

PrivilegeAttribute

11.5.1 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.

11.6 Class Group

All Superclasses:

PrivilegeAttribute

11.6.1 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).

11.7 Class Identity

All Superclasses:

PrivilegeAttribute

11.7.1 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.

11.8 Class Principal

Principal is a 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.`

12 References

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

[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/576871ad2f11bba78525662100 5419d7/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.oasisopen.org/committees/regrep/documents/2.1/specs/ebRS.pdf

[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
http://www.ebxml.org/specfrafts/

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

[NCName] Namespaces in XML 19990114
http://www.w3.org/TR/REC-xml-names/#NT-NCName.

13 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.

14 Contact Information

Team Leader

Name:	Lisa Carnahan
Company:	NIST
Street:	100 Bureau Drive STOP 8970
City, State, Postal Code:	Gaithersburg, MD 20899-8970
Country:	USA
Phone:	(301) 975-3362
Email:	lisa.carnahan@nist.gov

Editor

Name:	Sally Fuger
Company:	Automotive Industry Action Group
Street:	26200 Lahser Road, Suite 200
City, State, Postal Code:	Southfield, MI 48034
Country:	USA
Phone:	(248) 358-9744
Email:	sfuger@aiag.org

Technical 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