Creating A Single Global Electronic Market

1

OASIS/ebXML Registry Services Specification v1.02 DRAFT

OASIS/ebXML Registry Technical Committee

26 November 2001

This page intentionally left blank.

1 Status of this Document

Distribution of this document is unlimited.

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

This version:
http://www.oasis-open.org/committees/regrep/document/rsV1-01.pdf

***Latest version*:**
http://www.oasis-open.org/committees/regrep/documents/rsV1-01.pdf

2 OASIS/ebXML Registry Technical Committee

The OASIS/ebXML Registry Technical Committee has approved this document, as a DRAFT Specification, in its current form. At the time of this approval the following were members of the OASIS/ebXML Registry Technical Committee.

Table of Contents

Table of Figures

Table of Tables

3 Introduction

3.1 Summary of Contents of Document

This document defines the interface to the ebXML *Registry* Services as well as interaction protocols, message definitions and XML schema.

A separate document, *ebXML Registry Information Model* [ebRIM], provides information on the types of metadata that are stored in the Registry as well as the relationships among the various metadata classes.

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

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

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].

3.3 Audience

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

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

3.3.1.1.1 Related Documents

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

a) *ebXML Registry Information Model* [ebRIM]
b) *ebXML Message Service Specification* [ebMS]
c) *ebXML Business Process Specification Schema* [ebBPM]
d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

- Communicate functionality of Registry services to software developers
- Specify the interface for Registry clients and the Registry
- Provide a basis for future support of more complete ebXML Registry requirements
- Be compatible with other ebXML specifications

4.2 Caveats and Assumptions

The Registry Services specification is first in a series of phased deliverables. Later versions of the document will include additional functionality planned for future development. It is assumed that:

Interoperability requirements dictate that that at least one of the normative interfaces as referenced in this specification must be supported.

1. All access to the Registry content is exposed via the interfaces defined for the Registry Services.
2. The Registry makes use of a Repository for storing and retrieving persistent information required by the Registry Services. This is an implementation detail that will not be discussed further in this specification.

5 System Overview

5.1 What The ebXML Registry Does

The ebXML Registry provides a set of services that enable sharing of information between interested parties for the purpose of enabling *business process* integration between such parties based on the ebXML specifications. The shared information is maintained as objects in a repository and managed by the ebXML Registry Services defined in this document.

5.2 How The ebXML Registry Works

This section describes at a high level some use cases illustrating how Registry clients may make use of Registry Services to conduct B2B exchanges. It is meant to be illustrative and not prescriptive.

The following scenario provides a high level textual example of those use cases in terms of interaction between Registry clients and the Registry. It is not a complete listing of the use cases that could be envisioned. It assumes for purposes of example, a buyer and a seller who wish to conduct B2B exchanges using the RosettaNet PIP3A4 Purchase Order business protocol. It is assumed that both buyer and seller use the same Registry service provided by a third party. Note that the architecture supports other possibilities (e.g. each party uses its own private Registry).

5.2.1 Schema Documents Are Submitted

A third party such as an industry consortium or standards group submits the necessary schema documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the ObjectManager service of the Registry described in Section 7.3.

5.2.2 Business Process Documents Are Submitted

A third party, such as an industry consortium or standards group, submits the necessary business process documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the ObjectManager service of the Registry described in Section 7.3.

5.2.3 Seller’s Collaboration Protocol Profile Is Submitted

The seller publishes its *Collaboration Protocol* Profile or CPP as defined by [ebCPP] to the Registry. The CPP describes the seller, the role it plays, the services it offers and the technical details on how those services may be accessed. The seller classifies their Collaboration Protocol Profile using the Registry’s flexible *Classification* capabilities.

5.2.4 Buyer Discovers The Seller

The buyer browses the Registry using *Classification* schemes defined within the Registry using a Registry Browser GUI tool to discover a suitable seller. For example the buyer may look for all parties that are in the Automotive Industry, play a seller role, support the RosettaNet PIP3A4 process and sell Car Stereos.

The buyer discovers the seller’s CPP and decides to engage in a partnership with the seller.

5.2.5 CPA Is Established

The buyer unilaterally creates a *Collaboration Protocol Agreement* or CPA as defined by [ebCPP] with the seller using the seller's CPP and their own CPP as input. The buyer proposes a trading relationship to the seller using the unilateral CPA. The seller accepts the proposed CPA and the trading relationship is established.

Once the seller accepts the CPA, the parties may begin to conduct B2B transactions as defined by [ebMS].

5.3 Registry Users

We describe the actors who use the registry from the point of view of security and analyze the security concerns of the registry below. This analysis leads up to the security requirements for V2. Some of the actors are defined in Section 9.4.1 of [ebRS]. Note that same entity may take on multiple roles. For example, a Registration Authority and Registry Administrator may have the same identity.

Table 1: Registry Users

Actor	Function	ISO/IEC 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY be the same as Registration Authority
Registered User	Has a *contract* with the Registration Authority and MUST be authenticated by Registration Authority.		The contract could be a ebXML CPA or some other form of contract.
Registry Guest	Has no *contract* with Registration Authority. Does not have to be authenticated for Registry access. Cannot change contents of the Registry (MAY be permitted to *read* some RegistryObjects.)		Note that a Registry Guest is *not* a Registry Reader.
Submitting Organization	A Registered User who does lifecycle operations on permitted RegistryObjects.	Submitting Organization (SO)	
Registry Reader	A Registered User who has only *read* access		
Responsible Organization	Creates Registry Objects	Responsible Organization (RO)	RO MAY have the same identity as SO
Registry Client	Registered User or Registered Guest		

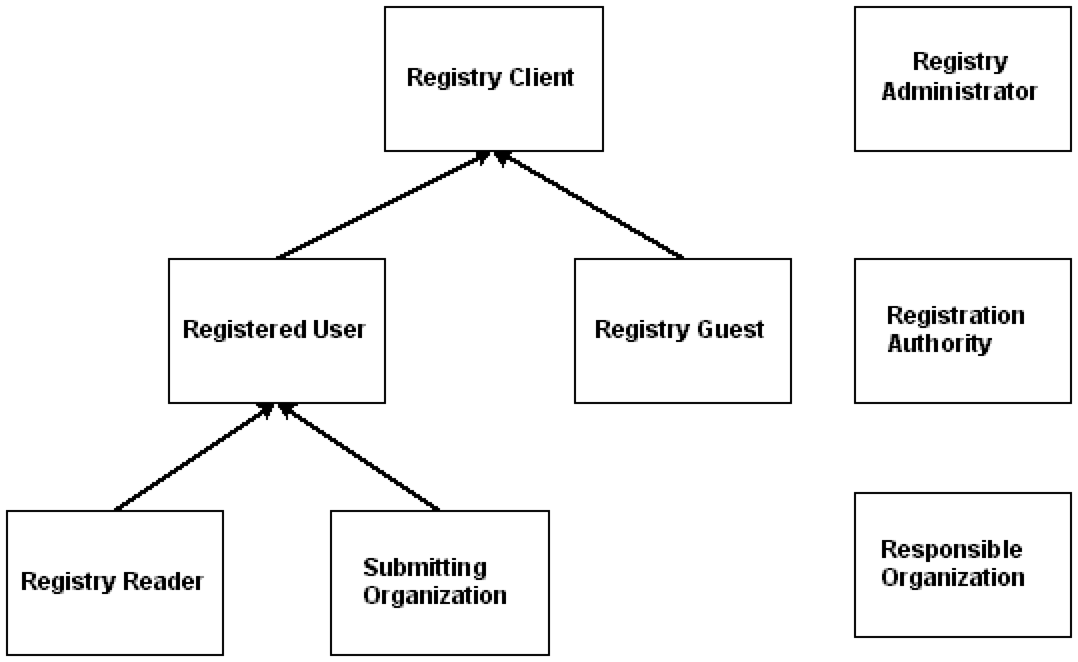


Figure 1: Actor Relationships

<u>**Note:**</u>

In V2, we are not distinguishing between Submitting Organization and Responsible Organization.

Registration of a user happens out-of-band for V2.

For V2 we do not distinguish between Registry Administrator and Registration Authority.

5.4 Where the Registry Services May Be Implemented

The Registry Services may be implemented in several ways including, as a public web site, as a private web site, hosted by an ASP or hosted by a VPN provider.

5.5 Implementation Conformance

An implementation is a *conforming* ebXML Registry if the implementation meets the conditions in Section 5.4.1. An implementation is a conforming ebXML Registry Client if the implementation meets the conditions in Section 5.4.2. An implementation is a conforming ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to the conditions of Section 5.4.1 and Section 5.4.2. An implementation shall be a conforming ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and Registry Client.

5.5.1 Conformance as an ebXML Registry

An implementation conforms to this specification as an ebXML registry if it meets the following conditions:

1. Conforms to *the ebXML Registry Information Model [ebRIM]*.
2. Supports the syntax and semantics of the Registry Interfaces and Security Model.
3. Supports the defined ebXML Registry DTD (Appendix A)
4. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

5.5.2 Conformance as an ebXML Registry Client

An implementation conforms to this specification, as an ebXML Registry Client if it meets the following conditions:

1. Supports the ebXML CPA and bootstrapping process.
2. Supports the syntax and the semantics of the Registry Client Interfaces.
3. Supports the defined ebXML Error Message DTD.
4. Supports the defined ebXML Registry DTD.

6 ebXML Registry Architecture

The ebXML Registry architecture consists of an ebXML Registry Service and ebXML Registry Clients. The ebXML Registry Service provides the methods for managing a repository. An ebXML Registry Client is an application used to access the Registry.

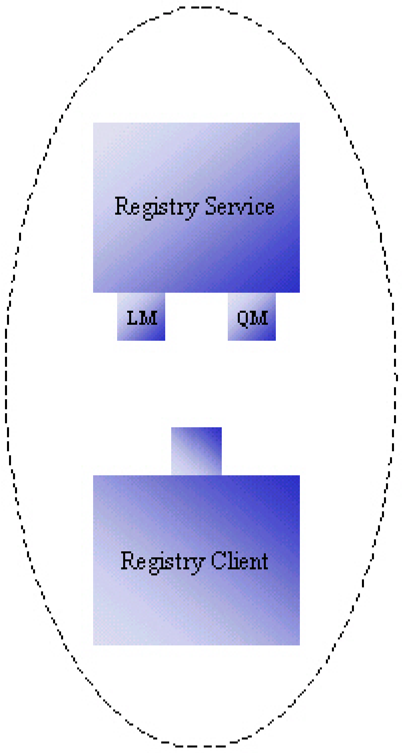


Figure 2: ebXML Registry Service Architecture

6.1 Registry Service Description

The ebXML Registry Service is comprised of a robust set of interfaces designed to fundamentally manage the objects and inquiries associated with the ebXML Registry. The two primary interfaces for the Registry Service consist of:

- A Life Cycle Management interface that provides a collection of methods for managing objects within the Registry.
- A Query Management Interface that controls the discovery and retrieval of information from the Registry.

A registry client program utilizes the services of the registry by invoking methods on one of the above interfaces defined by the Registry Service. This specification defines the interfaces exposed by the Registry Service (Sections 6.4 and 6.5) as well as the interface for the Registry Client (Section 6.6).

6.2 Abstract Registry Service

The architecture defines the ebXML Registry as an abstract registry service that is defined as:

1. A set of interfaces that must be supported by the registry.
2. The set of methods that must be supported by each interface.
3. The parameters and responses that must be supported by each method.

The abstract registry service neither defines any specific implementation for the ebXML Registry, nor does it specify any specific protocols used by the registry. Such implementation details are described by concrete registry services that realize the abstract registry service.

The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide two key functional interfaces called `QueryManager`[1] (QM) and `LifeCycleManager`[2] (LM).

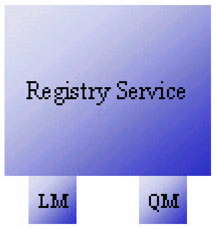


Figure 3: The Abstract ebXML Registry Service

Appendix A.1 describes the abstract service definition in the Web Service Description Language (WSDL) syntax.

6.3 Concrete Registry Services

The architecture allows the abstract registry service to be mapped to one or more concrete registry services defined as:

- Implementations of the interfaces defined by the abstract registry service.
- Bindings of these concrete interfaces to specific communication protocols.

This specification describes two concrete bindings for the abstract registry service:

- A SOAP binding using the HTTP protocol
- An ebXML Messaging Service (ebMS) binding

A registry may implement one or both of the concrete bindings for the abstract registry service as shown in Figure 4.

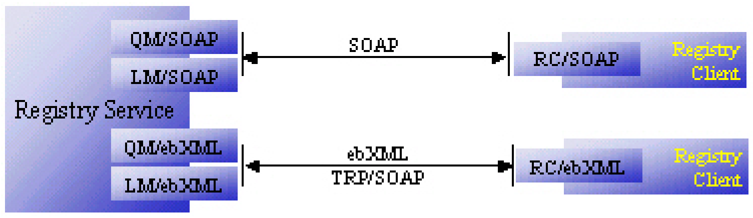


Figure 4: A Concrete ebXML Registry Service

Figure 4 shows a concrete implementation of the abstract ebXML Registry (RegistryService) on the left side. The RegistryService provides the QueryManager and LifeCycleManager interfaces available with multiple protocol bindings (SOAP and ebMS).

Figure 4 also shows two different clients of the ebXML Registry on the right side. The top client uses SOAP interface to access the registry while the lower client uses ebMS interface. Clients use the appropriate concrete interface within the RegistryService service based upon their protocol preference.

[1] Known as ObjectQueryManager in V1.0

[2] Known as ObjectManager in V1.0

6.3.1 SOAP Binding

6.3.1.1 WSDL Terminology Primer

This section provides a brief introduction to Web Service Description Language (WSDL) since the SOAP binding is described using WSDL syntax. WSDL provides the ability to describe a web service in abstract as well as with concrete bindings to specific protocols. In WSDL, an abstract service consists of one or more **`port types`** or end-points. Each port type consists of a collection of **`operations`**. Each operation is defined in terms of `messages` that define what data is exchanged as part of that operation. Each message is typically defined in terms of elements within an XML Schema definition.

An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract service is bound to a specific protocol by providing a **`binding`** definition for each abstract port type that defines additional protocols specific details. Finally, a concrete **`service`** definition is defined as a collection of **`ports`**, where each port simply adds address information such as a URL for each concrete port.

6.3.1.2 Concrete Binding for SOAP

This section assumes that the reader is somewhat familiar with SOAP and WSDL. The SOAP binding to the ebXML Registry is defined as a web service description in WSDL as follows:

- A single service element with name “RegistryService” defines the concrete SOAP binding for the registry service.
- The service element includes two port definitions, where each port corresponds with one of the interfaces defined for the abstract registry service. Each port includes an HTTP URL for accessing that port.
- Each port definition also references a binding element, one for each interface defined in the WSDL for the abstract registry service.

```
<service name = "RegistryService">
        <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
              <soap:address location = "http://your_URL_to_your_QueryManager"/>
        </port>

        <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
              <soap:address location = "http://your_URL_to_your_QueryManager"/>
        </port>
</service>
```

The complete WSDL description for the SOAP binding is described in Appendix A.2

6.3.2 ebXML Message Service Binding

6.3.2.1 Service and Action Elements

When using the ebXML Messaging Services Specification, ebXML Registry Service elements correspond to Messaging Service elements as follows:

- The value of the Service element in the MessageHeader is an ebXML Registry Service interface name (e.g., “LifeCycleManager”). The type attribute of the Service element should have a value of “ebXMLRegistry”.
- The value of the Action element in the MessageHeader is an ebXML Registry Service method name (e.g., “submitObjects”).

```
<eb:Service eb:type=”ebXMLRegistry”>LifeCycleManger</eb:Service>
<eb:Action>submitObjects</eb:Action>
```

Note that the above allows the Registry Client only one interface/method pair per message. This implies that a Registry Client can only invoke one method on a specified interface for a given request to a registry.

6.3.2.2 Synchronous and Asynchronous Responses

All methods on interfaces exposed by the registry return a response message.

Asynchronous response

When a message is sent asynchronously, the Registry will return two response messages. The first message will be an immediate response to the request and does not reflect the actual response for the request. This message will contain:

- MessageHeader;
- RegistryResponse element with empty content (e.g., **NO** AdHocQueryResponse);
  - status attribute with value **unavailable**.

The Registry delivers the actual Registry response element with non-empty content asynchronously at a later time. The delivery is accomplished by the Registry invoking the onResponse method on the RegistryClient interface as implemented by the registry client application. The onResponse method includes a RegistryResponse element which hasa complete as defined by the Synchronous response section below. The Registry response includes:

- MessageHeader;
- RegistryResponse element including;
  - Status attribute (success, failure, warning);
  - Optional RegistryErrorList.

Synchronous response

When a message is sent synchronously, the Message Service Handler will hold open the communication mechanism until the Registry returns a response. This message will contain:

- MessageHeader;
- RegistryResponse element including;
  - Status attribute (success, failure, warning);
  - Optional RegistryErrorList.

6.3.2.3 ebXML Registry Collaboration Profiles and Agreements

The ebXML CPP specification [ebCPP] defines a Collaboration-Protocol Profile (CPP) and a Collaboration-Protocol Agreement (CPA) as mechanisms for two parties to share information regarding their respective business processes. That specification assumes that a CPA has been agreed to by both parties in order for them to engage in B2B interactions.

This specification does not mandate the use of a CPA between the Registry and the Registry Client. However if the Registry does not use a CPP, the Registry shall provide an alternate mechanism for the Registry Client to discover the services and other information provided by a CPP. This alternate mechanism could be a simple URL.

The CPA between clients and the Registry should describe the interfaces that the Registry and the client expose to each other for Registry-specific interactions. The definition of the Registry CPP template and a Registry Client CPP template are beyond the scope of this document.

6.4 LifeCycleManager Interface

This is the interface exposed by the Registry Service that implements the object life cycle management functionality of the Registry. Its' methods are invoked by the Registry Client. For example, the client may use this interface to submit objects, to classify and associate objects and to deprecate and remove objects. For this specification the semantic meaning of submit, classify, associate, deprecate and remove is found in [ebRIM].

Table 2: LifeCycle Manager Summary

Method Summary of LifeCycleManager	
`RegistryResponse`	**`approveObjects`**`(ApproveObjectsRequest req)` Approves one or more previously submitted objects.
`RegistryResponse`	**`deprecateObjects`**`(DeprecateObjectsRequest req)` Deprecates one or more previously submitted objects.
`RegistryResponse`	**`removeObjects`**`(RemoveObjectsRequest req)` Removes one or more previously submitted objects from the Registry.
`RegistryResponse`	**`submitObjects`**`(SubmitObjectsRequest req)` Submits one or more objects and possibly related metadata such as Associations and Classifications.
`RegistryResponse`	**`updateObjects`**`(UpdateObjectsRequest req)` Updates one or more previously submitted objects.
`RegistryResponse`	**`addSlots`**`(AddSlotsRequest req)` Add slots to one or more registry entries.
`RegistryResponse`	**`removeSlots`**`(RemoveSlotsRequest req)` Remove specified slots from one or more registry entries.

6.5 QueryManager Interface

This is the interface exposed by the Registry that implements the Query management service of the Registry. Its' methods are invoked by the Registry Client. For example, the client may use this interface to perform browse and drill down queries or ad hoc queries on registry content.

Table 3: Query Manager

Method Summary of QueryManager	
`RegistryResponse`	**`submitAdhocQuery`**`(AdhocQueryRequest req)` Submit an ad hoc query request.

6.6 Registry Clients

6.6.1 Registry Client Description

The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts the two possible topologies supported by the registry architecture with respect to the Registry and Registry Clients. The picture on the left side shows the scenario where the Registry provides a web based "thin client" application for accessing the Registry that is available to the user using a common web browser. In this scenario the Registry Client interfaces reside across the Internet and are local to the Registry from the user's view. The picture on the right side shows the scenario where the user is using a "fat client" Registry Browser application to access the registry. In this scenario the Registry Client interfaces reside within the Registry Browser tool and are local to the Registry from the user's view. The Registry Client interfaces communicate with the Registry over the Internet in this scenario.

A third topology made possible by the registry architecture is where the Registry Client interfaces reside in a server side business component such as a Purchasing business component. In this topology there may be no direct user interface or user intervention involved. Instead, the Purchasing business component may access the Registry in an automated manner to select possible sellers or service providers based on current business needs.

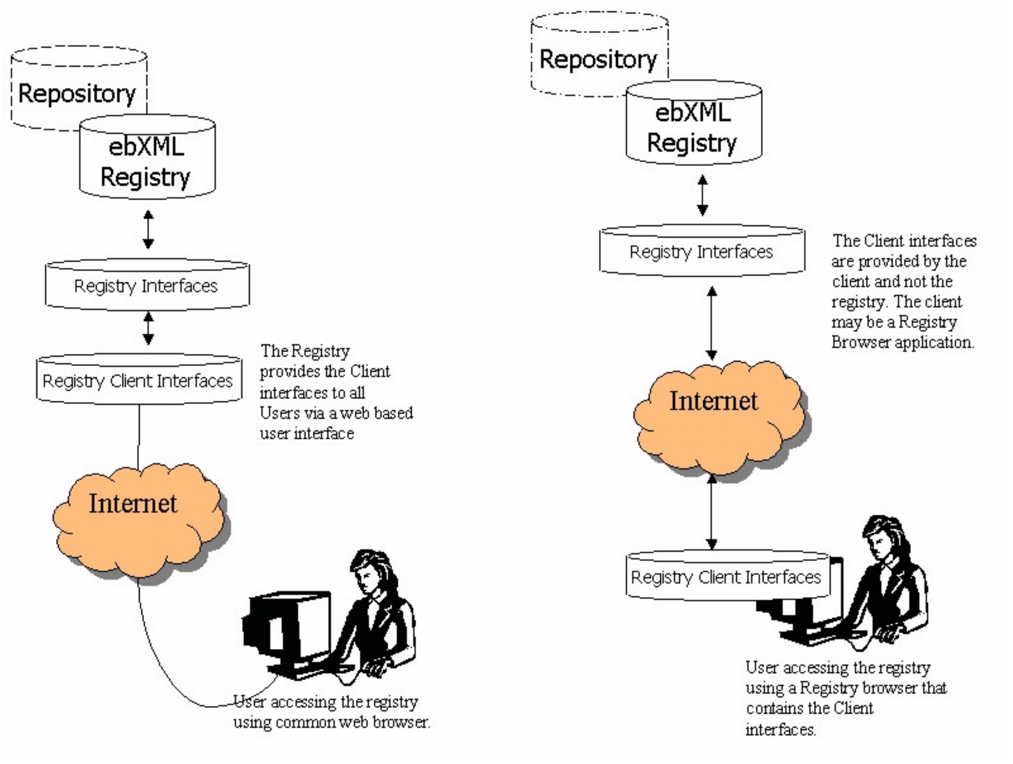

Repository
ebXML Registry
Registry Interfaces
Registry Client Interfaces
The Registry provides the Client interfaces to all Users via a web based user interface
Internet
User accessing the registry using common web browser.
Repository
ebXML Registry
Registry Interfaces
The Client interfaces are provided by the client and not the registry. The client may be a Registry Browser application.
Internet
Registry Client Interfaces
User accessing the registry using a Registry browser that contains the Client interfaces.

Figure 5: Registry Architecture Supports Flexible Topologies

6.6.2 Registry Communication Bootstrapping

Before a client can access the services of a Registry, there must be some communication bootstrappingbetween the client and the registry. The most essential aspect of this bootstrapping process is for the client to discover addressing information (e.g. an HTTP URL) to each of the concrete service interfaces of the Registry. The client may obtain the addressing information by discovering the ebXML Registry in a public registry such as UDDI or within another ebXML Registry.

- In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available in a WSDL desription for the registry. This WSDL conforms to the template WSDL description in Appendix A.2. This WSDL description may be discovered in a public registry such as UDDI.

- In case of ebMS binding, the information exchange between the client and the registry may be accomplished in a registry specific manner, which may involve establishing a CPA between the client and the registry. Once the information exchange has occurred the Registry and the client will have addressing information (e.g. URLs) for the other party.

6.6.2.1 Communication Bootstrapping for SOAP Binding

Each ebXML Registry must provide a WSDL description for its RegistryService as defined by Appendix A.2. A client uses the WSDL description to determine the address information of the RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the RegistryService may be accessed via a URL specified in the WSDL for the registry.

The use of WSDL enables the client to use automated tools such as a WSDL compiler to generate stubs that provide access to the registry in a language specific manner.

At minimum, any client may access the registry over SOAP/HTTP using the address information within the WSDL, with minimal infrastructure requirements other than the ability to make synchronous SOAP call to the SOAP based ports on the RegistryService.

6.6.2.2 Communication Bootstrapping for ebXML Message Service

Since there is no previously established CPA between the Registry and the RegistryClient, the client must know at least one Transport-specific communication address for the Registry. This communication address is typically a URL to the Registry, although it could be some other type of address such as an email address. For example, if the communication used by the Registry is HTTP, then the communication address is a URL. In this example, the client uses the Registry's public URL to create an implicit CPA with the Registry. When the client sends a request to the Registry, it provides a URL to itself. The Registry uses the client's URL to form its version of an implicit CPA with the client. At this point a session is established within the Registry. For the duration of the client's session with the Registry, messages may be exchanged bidirectionally as required by the interaction protocols defined in this specification.

6.6.3 RegistryClient Interface

This is the principal interface implemented by a Registry client. The client provides this interface when creating a connection to the Registry. It provides the methods that are used by the Registry to deliver asynchronous responses to the client. Note that a client need not provide a RegistryClient interface if the [CPA] between the client and the registry does not support asynchronous responses.

The registry sends all asynchronous responses to operations to the onResponse method.

Table 4: RegistryClient Summary

Method Summary of RegistryClient	
`void`	`onResponse(RegistryResponse resp)` Notifies client of the response sent by registry to previously submitted request.

6.6.4 Registry Response Class Hierarchy

Since many of the responses from the registry have common attributes, they are arranged in a class hierarchy as shown in . This hierarchy is reflected in the registry schema.

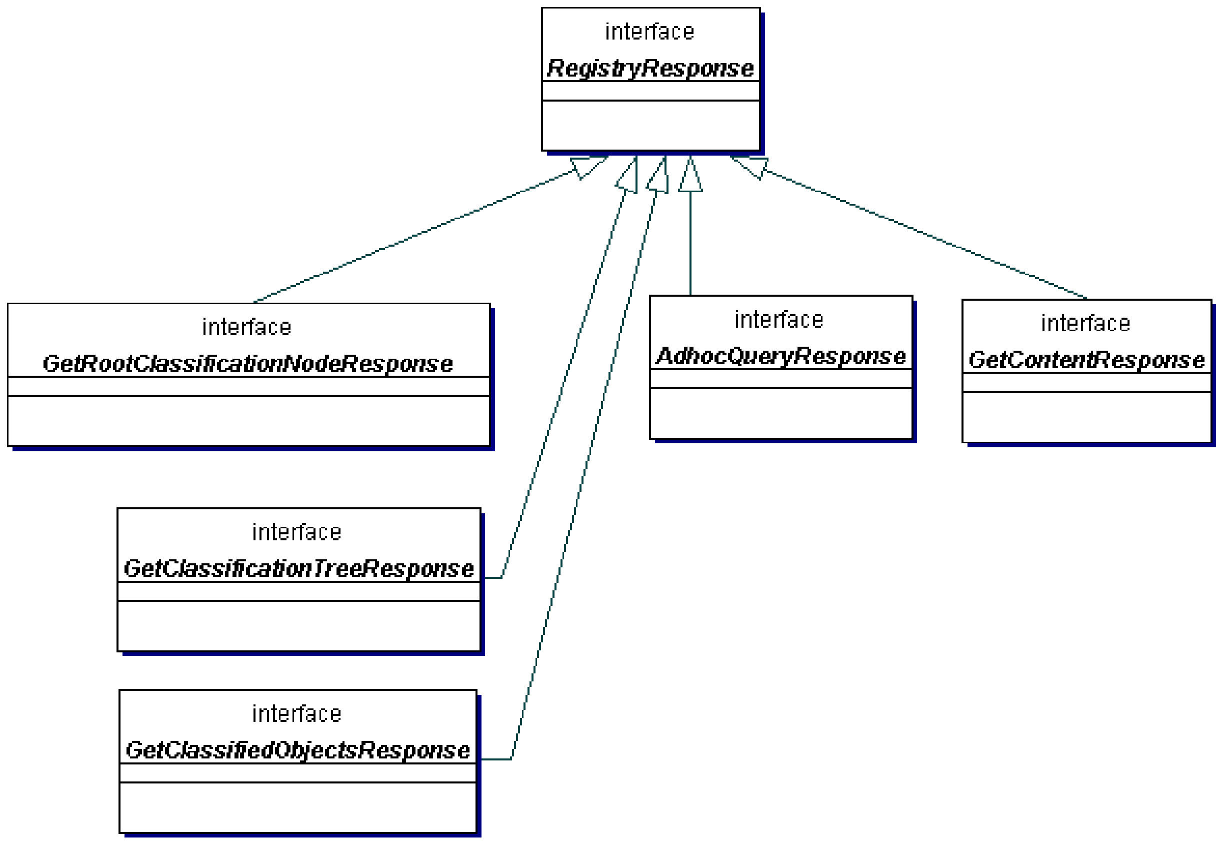


Figure 6: Registry Response Class Hierarchy

6.7 Interoperability Requirements

6.7.1 Client Interoperability

The architecture requires that any ebXML compliant registry client can access any ebXML compliant registry service in an interoperable manner. An ebXML Registry may implement any number of protocol bindings from the set of normative bindings (currently ebXML TRP and SOAP/HTTP) defined in this proposal. The support of additional protocol bindings is optional.

6.7.2 Inter-Registry Cooperation

This version of the specification does not preclude ebXML Registries from cooperating with each other to share information, nor does it preclude owners of ebXML Registries from registering their ebXML registries with other registry systems, catalogs, or directories.

Examples include:

- An ebXML Registry that serves as a registry of ebXML Registries.
- A non-ebXML Registry that serves as a registry of ebXML Registries.
- Cooperative ebXML Registries, where multiple ebXML registries register with each other in order to form a federation.

7 Life Cycle Management Service

This section defines the ObjectManagement service of the Registry. The Life Cycle Management Service is a sub-service of the Registry service. It provides the functionality required by RegistryClients to manage the life cycle of repository items (e.g. XML documents required for ebXML business processes). The Life Cycle Management Service can be used with all types of repository items as well as the metadata objects specified in [ebRIM] such as Classification and Association.

The minimum-security *policy* for an ebXML registry is to accept content from any client if a certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the content. Submitting Organizations do not have to register prior to submitting content.

7.1 Life Cycle of a Repository Item

The main purpose of the ObjectManagement service is to manage the life cycle of repository items. Figure 7 shows the typical life cycle of a repository item. Note that the current version of this specification does not support Object versioning. Object versioning will be added in a future version of this specification

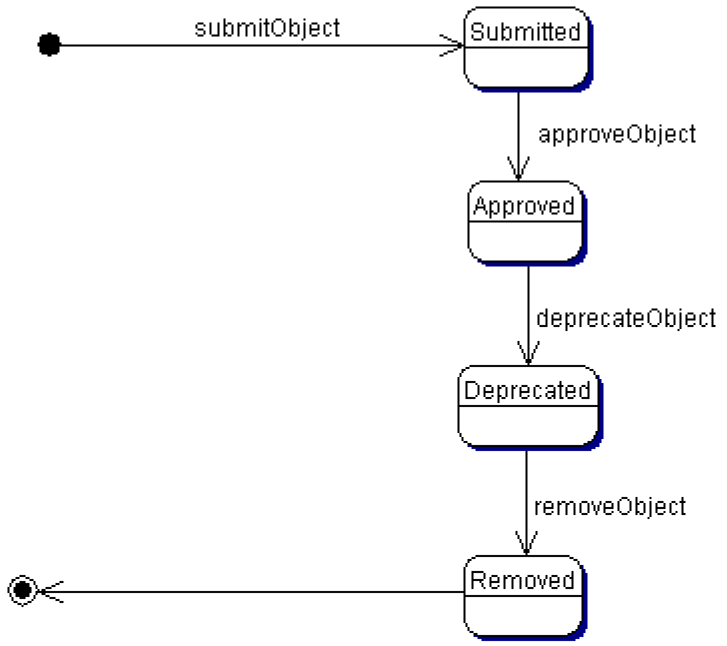


Figure 7: Life Cycle of a Repository Item

7.2 RegistryObject Attributes

A repository item is associated with a set of standard metadata defined as attributes of the RegistryObject class and its sub-classes as described in [ebRIM]. These attributes reside outside of the actual repository item and catalog descriptive information about the repository item. XML elements called ExtrinsicObject and IntrinsicObject (See Appendix B for details) encapsulate all object metadata attributes defined in [ebRIM] as XML attributes.

7.3 The Submit Objects Protocol

This section describes the protocol of the Registry Service that allows a RegistryClient to submit one or more repository items to the repository using the *ObjectManager* on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

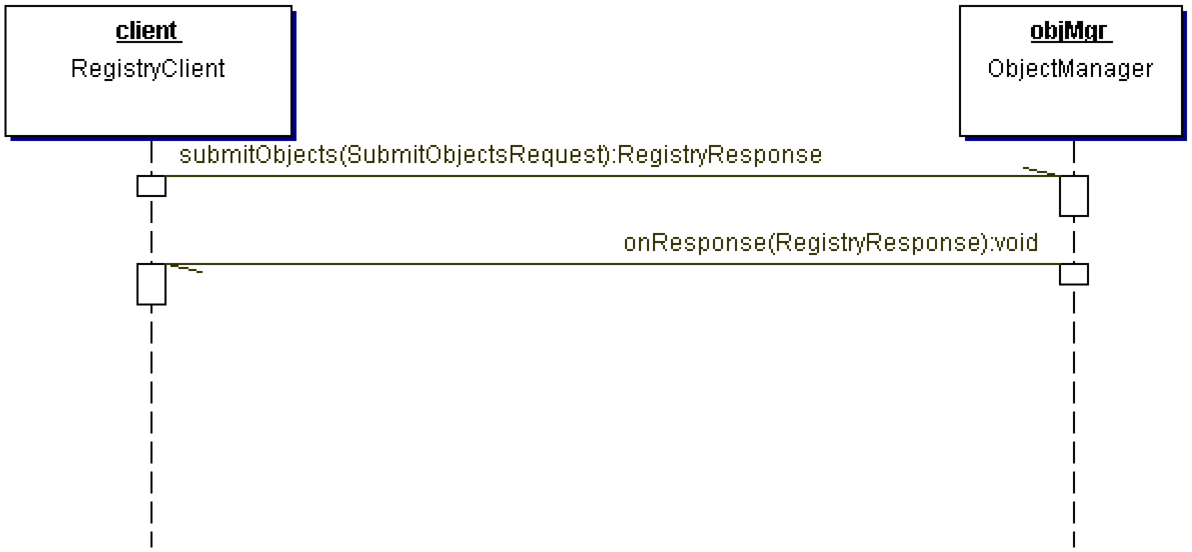

client
RegistryClient
objMgr
ObjectManager
submitObjects(SubmitObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 8: Submit Objects Sequence Diagram

For details on the schema for the *Business documents* shown in this process refer to Appendix B.

The SubmitObjectRequest message includes a RegistrEntryList element.

The RegistryEntryList element specifies one or more ExtrinsicObjects or other RegistryEntries such as Classifications, Associations, ExternalLinks, or Packages.

An ExtrinsicObject element provides required metadata about the content being submitted to the Registry as defined by [ebRIM]. Note that these standard ExtrinsicObject attributes are separate from the repository item itself, thus allowing the ebXML Registry to catalog objects of any object type.

7.3.1 Universally Unique ID Generation

As specified by [ebRIM], all objects in the registry have a unique id. The id must be a *Universally Unique Identifier (UUID)* and must conform to the to the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID].

(e.g. `urn:uuid:a2345678-1234-1234-123456789012`)

The registry usually generates this id. The client may optionally supply the **id** attribute for submitted objects. If the client supplies the **id** and it conforms to the format of a URN that specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the **id** for the object. In this case, the registry must honour a client-supplied **id** and use it as the **id** attribute of the object in the registry. If the **id** is found by the registry to not be globally unique, the registry must raise the error condition: InvalidIdError.

If the client does not supply an **id** for a submitted object then the registry must generate a universally unique **id**. Whether the client generates the **id** or whether the registry generates it, it must be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

7.3.2 ID Attribute And Object References

The id attribute of an object may be used by other objects to reference the first object. Such references are common both within the SubmitObjectsRequest as well as within the registry.

Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the SubmitObjectsRequest as well as to refer to an object within the registry. An object in the SubmitObjectsRequest that needs to be referred to within the request document may be assigned an id by the submitter so that it can be referenced within the request. The submitter may give the object a proper uuid URN, in which case the id is permanently assigned to the object within the registry. Alternatively, the submitter may assign an arbitrary id (not a proper uuid URN) as long as the id is unique within the request document. In this case the id serves as a linkage mechanism within the request document but must be ignored by the registry and replaced with a registry generated id upon submission.

When an object in a SubmitObjectsRequest needs to reference an object that is already in the registry, the request must contain an ObjectRef element whose id attribute is the id of the object in the registry. This id is by definition a proper uuid URN. An ObjectRef may be viewed as a proxy within the request for an object that is in the registry.

7.3.3 Audit Trail

The RS must create AuditableEvents object with eventType Created for each RegistryObject created via a SubmitObjects request.

7.3.4 Submitting Organization

The RS must create an Association of type SubmittedBy between the submitting organization and each RegistryObject created via a SubmitObjects request. (Submitting organization is determined from the organization attribute of the User who submits a SubmitObjects request.)

7.3.5 Error Handling

A SubmitObjects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of “success” back to the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure” back to the client. Failure occurs when one or more Error conditions are raised in the processing of the submitted objects. Warning messages do not result in failure of the request. The following business rules apply:

Table 5 Submit Objects Error Handling

Business Rule	Applies To	Error/Warning
`ID not unique`	`All Classes`	`Error`
`Not authorized`	`All Classes`	`Error`
`Referenced object not found.`	`Association, Classification, ClassificationNode, Organization`	`Error`
`Associations not allowed to connect to deprecated objects.`	`Association`	`Error`
`Object status, majorVersion and minorVersion are set by the RS, and ignored if supplied.`	`All Classes`	`Warning`

7.3.6 Sample SubmitObjectsRequest

The following example shows several different use cases in a single SubmitObjectsRequest. It does not show the complete SOAP or [ebMS] Message with the message header and additional payloads in the message for the repository items.

A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects that are being submitted. It may also contain any number of ObjectRefs to link objects being submitted to objects already within the registry.

```

<?xml version = "1.0" encoding = "UTF-8"?>
<SubmitObjectsRequest
  xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
  xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
  xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
  >

  <RegistryObjectList>

    <!--
    The following 3 objects package specified ExtrinsicObject in specified
      RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
      being  submitted
      -->

    <rim:RegistryPackage id = "acmePackage1" >
      <Name xmlns = "">
        <LocalizedString value = "RegistryPackage #1"/>
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's package #1"/>
      </Description>
    </rim:RegistryPackage>

    <rim:ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1" >
      <Name xmlns = "">
        <LocalizedString value = "Widget Profile" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's profile for selling widgets" />
      </Description>
    </rim:ExtrinsicObject>

    <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
= "acmePackage1" targetObject = "acmeCPP1" />

    <!--
      The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
      Where the RegistryPackage is being submitted and the ExtrinsicObject is
      already in registry
      -->

    <rim:RegistryPackage id = "acmePackage2" >
      <Name xmlns = "">
        <LocalizedString value = "RegistryPackage #2"/>
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's package #2"/>
      </Description>
    </rim:RegistryPackage>

    <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!--
      The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
      where the RegistryPackage and the ExtrinsicObject are already in registry
      -->

    <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
    <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>

    <!-- id is unspecified implying that registry must create a uuid for this object -->

    <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>

    <!--
      The following 3 objects externally link specified ExtrinsicObject using
      specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
      are being submitted
      -->

    <rim:ExternalLink id = "acmeLink1" >
      <Name xmlns = "">
        <LocalizedString value = "Link #1"/>
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's Link #1"/>
      </Description>
    </rim:ExternalLink>

    <rim:ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" >
      <Name xmlns = "">
        <LocalizedString value = "Sprockets Profile" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's profile for selling sprockets"/>
      </Description>
    </rim:ExtrinsicObject>

    <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>

    <!--
      The following 2 objects externally link specified ExtrinsicObject using specified
      ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
      is already in registry. Note that the targetObject points to an ObjectRef in a
      previous line
      -->

    <rim:ExternalLink id = "acmeLink2">
      <Name xmlns = "">
        <LocalizedString value = "Link #2"/>
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "ACME's Link #2"/>
      </Description>
    </rim:ExternalLink>

    <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
"ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>

    <!--
      The following 3 objects externally identify specified ExtrinsicObject using specified
      ExternalIdentifier, where the ExternalIdentifier is being submitted and the
      ExtrinsicObject is already in registry. Note that the targetObject points to an
      ObjectRef in a previous line
      -->

    <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
      <Name xmlns = "">
        <LocalizedString value = "DUNS"/>
      </Name>

      <Description xmlns = "">
        <LocalizedString value = "This is the DUNS scheme"/>
      </Description>
    </rim:ClassificationScheme>
    <rim:ExternalIdentifier id = "acmeDUNSId"  identificationScheme="DUNS-id" value = 
"13456789012">
      <Name xmlns = "">
        <LocalizedString value = "DUNS" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "DUNS ID for ACME"/>
      </Description>
    </rim:ExternalIdentifier>

    <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType = 
"ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>

    <!--
      The following show submission of a brand new classification scheme in its entirety
      -->
    <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
      <Name xmlns = "">
        <LocalizedString value = "Geography"/>
      </Name>

      <Description xmlns = "">
        <LocalizedString value = "This is a sample Geography scheme"/>
      </Description>

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

      <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" />
        </ClassificationNode>
      </ClassificationNode>
    </rim:ClassificationScheme>


    <!--
      The following show submission of a Automotive sub-tree of ClassificationNodes that 
      gets added to an existing classification scheme named 'Industry'
      that is already in the registry
      -->

    <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
    <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
123456789012">
      <Name xmlns = "">
        <LocalizedString value = "Automotive" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
      </Description>
    </rim:ClassificationNode>

    <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
      <Name xmlns = "">
        <LocalizedString value = "Parts Supplier" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "The Parts Supplier node under the Automotive node" />
      </Description>
    </rim:ClassificationNode>

    <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
      <Name xmlns = "">
        <LocalizedString value = "Engine Supplier" />
      </Name>
      <Description xmlns = "">
        <LocalizedString value = "The Engine Supplier node under the Automotive node" />
      </Description>
    </rim:ClassificationNode>

    <!--
      The following show submission of 2 Classifications of an object  that is already in
      the registry using 2 ClassificationNodes. One ClassificationNode
      is being submitted in this request (Japan) while the other is already in the registry.
      -->

    <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
1234-123456789012" classificationNode = "Japan-id">
      <Description xmlns = "">
        <LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
      </Description>
    </rim:Classification>

    <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
"urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
123456789012">
      <Description xmlns = "">
        <LocalizedString value = "Classifies object using a node in the registry" />
      </Description>
    </rim:Classification>

    <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
  </RegistryObjectList>
</SubmitObjectsRequest>
```

7.4 The Update Objects Protocol

This section describes the protocol of the Registry Service that allows a Registry Client to update one or more existing Registry Items in the registry on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

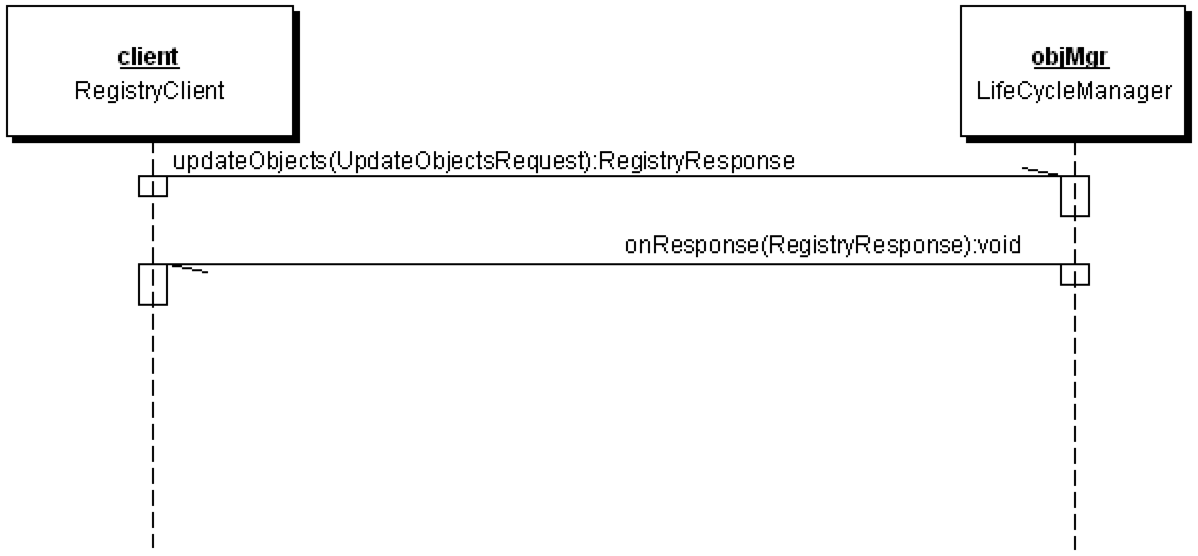

client
RegistryClient
objMgr
LifeCycleManager
updateObjects(UpdateObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 9: Update Objects Sequence Diagram

For details on the schema for the Business documents shown in this process refer to Appendix B. The UpdateObjectsRequest message includes a RegistryObjectList element. The RegistryObjectList element specifies one or more RegistryObjects. Each object in the list must be a current RegistryObject. RegistryObjects must include all attributes, even those the user does not intend to change. A missing attribute is interpreted as a request to set that attribute to NULL.

7.4.1 Audit Trail

The RS must create AuditableEvents object with eventType Updated for each RegistryObject updated via an UpdateObjects request.

7.4.2 Submitting Organization

The RS must maintain an Association of type SubmittedBy between the submitting organization and each RegistryObject updated via an UpdateObjects request. If an UpdateObjects request is accepted from a different submitting organization, then the RS must delete the original association object and create a new one. Of course, the AccessControlPolicy may prohibit this sort of update in the first place. (Submitting organization is determined from the organization attribute of the User who submits an UpdateObjects request.)

7.4.3 Error Handling

An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of "success" back to the client. In the event of failure, the registry sends a RegistryResponse with a status of "failure" back to the client. Failure occurs when one or more Error conditions are raised in the processing of the updated objects. Warning messages do not result in failure of the request. The following business rules apply:

Table 6: Update Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	All Classes	Error
Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
ContentURI cannot be changed via the UpdateObjects protocol, and is ignored if supplied.	ExtrinsicObject	Warning
Object status, majorVersion and minorVersion cannot be changed via the UpdateObjects protocol, ignored if supplied.	All Classes	Warning
RegistryEntries with stability = "Stable" should not be updated.	All Classes	Warning

7.5 The Add Slots Protocol

This section describes the protocol of the Registry Service that allows a client to add slots to a previously submitted registry entry using the ObjectManager. Slots provide a dynamic mechanism for extending registry entries as defined by [ebRIM].

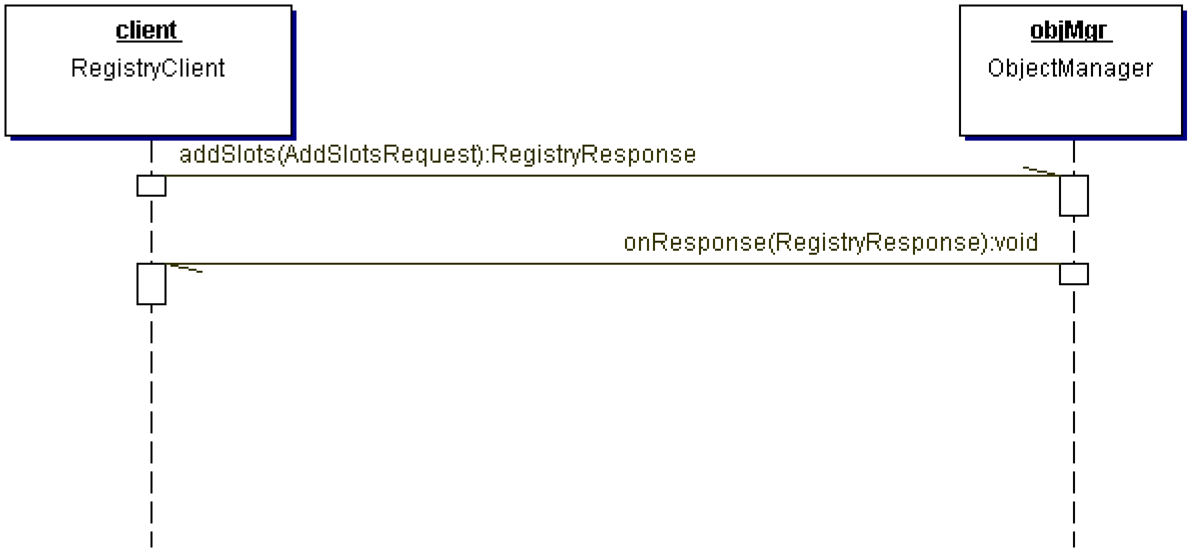

client
RegistryClient
objMgr
ObjectManager
addSlots(AddSlotsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 10: Add Slots Sequence Diagram

In the event of success, the registry sends a RegistryResponse with a status of “success” back to the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure” back to the client.

7.6 The Remove Slots Protocol

This section describes the protocol of the Registry Service that allows a client to remove slots to a previously submitted registry entry using the ObjectManager.

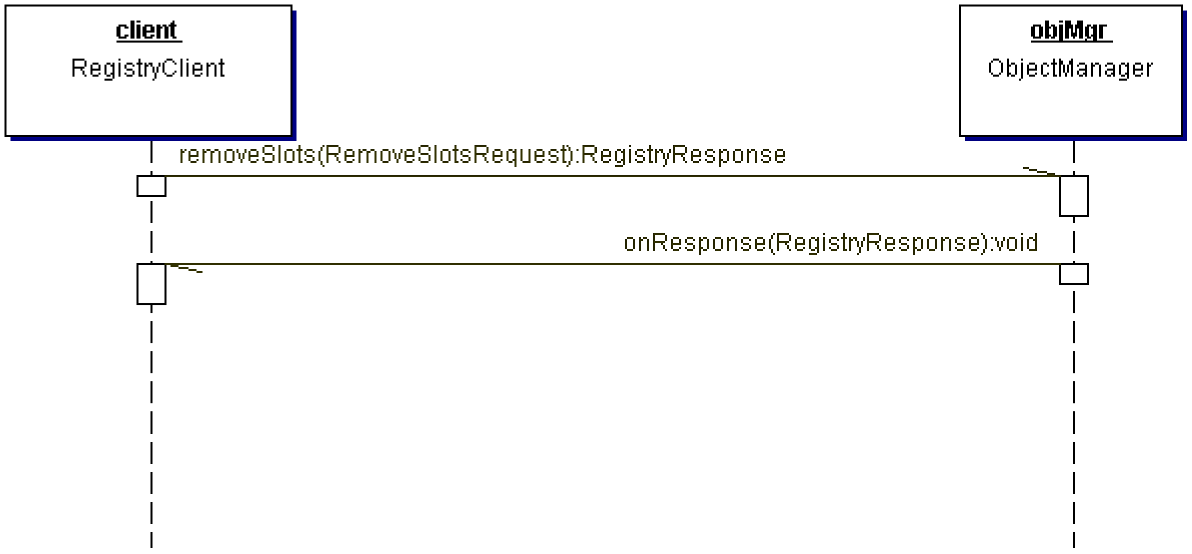

client
RegistryClient
objMgr
ObjectManager
removeSlots(RemoveSlotsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 11: Remove Slots Sequence Diagram

7.7 The Approve Objects Protocol

This section describes the protocol of the Registry Service that allows a client to approve one or more previously submitted repository items using the ObjectManager. Once a repository item is approved it will become available for use by business parties (e.g. during the assembly of new CPAs and Collaboration Protocol Profiles).

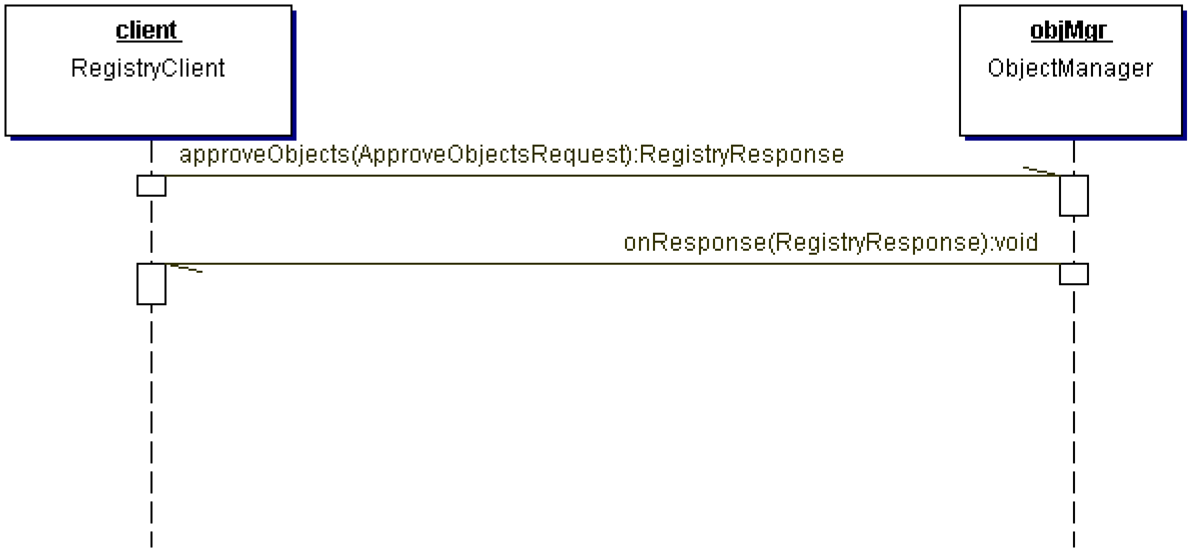

client
RegistryClient
objMgr
ObjectManager
approveObjects(ApproveObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 12: Approve Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix B.

7.7.1 Audit Trail

The RS must create AuditableEvents object with eventType Approved for each RegistryObject approved via an Approve Objects request.

7.7.2 Submitting Organization

The RS must maintain an Association of type SubmittedBy between the submitting organization and each RegistryObject updated via an ApproveObjects request. If an ApproveObjects request is accepted from a different submitting organization, then the RS must delete the original association object and create a new one. Of course, the AccessControlPolicy may prohibit this sort of ApproveObjects request in the first place. (Submitting organization is determined from the organization attribute of the User who submits an ApproveObjects request.)

7.7.3 Error Handling

An ApproveObjects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of “success” back to the client. In the event of failure, the registry sends a RegistryResponse with a status of “failure” back to the client. Failure occurs when one or more Error conditions are raised in the processing of the object reference list. Warning messages do not result in failure of the request. The following business rules apply:

Table 7: Approve Objects Error Handling

Business Rule	Applies To	Error/Warning
`Object not found`	`All Classes`	`Error`
`Not authorized`	`RegistryEntry Classes`	`Error`
`Only RegistryEntries may be "approved".`	`All Classes other than RegistryEntry classes`	`Error`

Object status is already "Approved".	RegistryEntry Classes	Warning

7.8 The Deprecate Objects Protocol

This section describes the protocol of the Registry Service that allows a client to deprecate one or more previously submitted repository items using the ObjectManager. Once an object is deprecated, no new references (e.g. *new* Associations, Classifications and ExternalLinks) to that object can be submitted. However, existing references to a deprecated object continue to function normally.

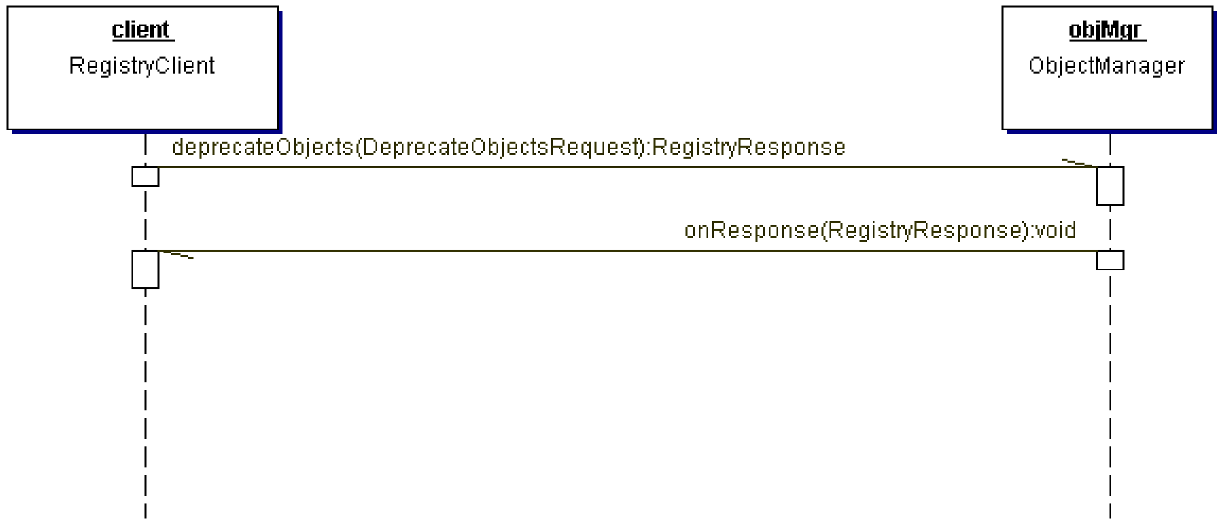

client
RegistryClient
objMgr
ObjectManager
deprecateObjects(DeprecateObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 13: Deprecate Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix B.

7.8.1 Audit Trail

The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject deprecated via a Deprecate Objects request.

7.8.2 Submitting Organization

The RS must maintain an Association of type SubmittedBy between the submitting organization and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request is accepted from a different submitting organization, then the RS must delete the original association object and create a new one. Of course, the AccessControlPolicy may prohibit this sort of Deprecate Objects request in the first place. (Submitting organization is determined from the organization attribute of the User who submits a Deprecate Objects request.)

7.8.3 Error Handling

A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of "success" back to the client. In the event of failure, the registry sends a RegistryResponse with a status of "failure" back to the client. Failure occurs when one or more Error conditions are raised in the processing of the object reference list. Warning messages do not result in failure of the request. The following business rules apply:

Table 8: Deprecate Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "deprecated".	All Classes other than RegistryEntry classes	Error
Object status is already "Deprecated".	RegistryEntry Classes	Warning

7.9 The Remove Objects Protocol

This section describes the protocol of the Registry Service that allows a client to remove one or more RegistryEntry instances and/or repository items using the ObjectManager.

The RemoveObjectsRequest message is sent by a client to remove RegistryEntry instances and/or repository items. The RemoveObjectsRequest element includes an XML attribute called *deletionScope* which is an enumeration that can have the values as defined by the following sections.

7.9.1 Deletion Scope DeleteRepositoryItemOnly

This deletionScope specifies that the request should delete the repository items for the specified registry entries but not delete the specified registry entries. This is useful in keeping references to the registry entries valid.

7.9.2 Deletion Scope DeleteAll

This deletionScope specifies that the request should delete both the RegistryEntry and the repository item for the specified registry entries. Only if all references (e.g. Associations, Classifications, ExternalLinks) to a RegistryEntry have been removed, can that RegistryEntry then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to remove a RegistryEntry while it still has references raises an error condition: InvalidRequestError.

The remove object protocol is expressed in UML notation as described in Appendix C.

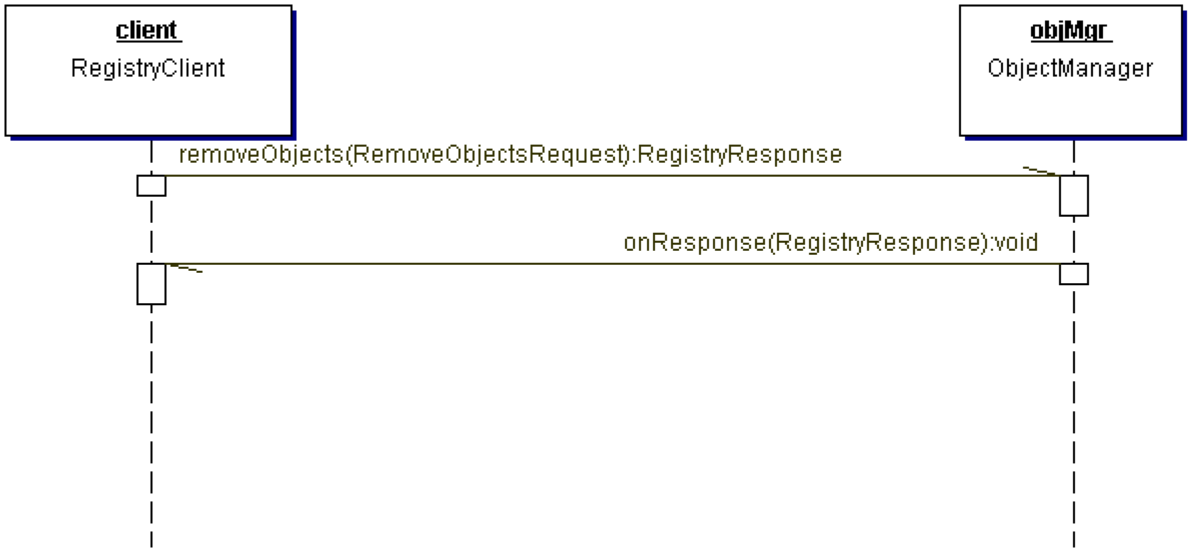

client
RegistryClient
objMgr
ObjectManager
removeObjects(RemoveObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 14: Remove Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix B.

7.9.3 Error Handling

A Remove Objects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of "success" back to the client. In the event of failure, the registry sends a RegistryResponse with a status of "failure" back to the client. Failure occurs when one or more Error conditions are raised in the processing of the object reference list. Warning messages do not result in failure of the request. The following business rules apply:

Table 9: Remove Objects Error Handling

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "removed".	All Classes other than RegistryEntry classes	Error

8 Query Management Service

This section describes the capabilities of the Registry Service that allow a client (QueryManagerClient) to search for or query different kind of registry objects in the ebXML Registry using the QueryManager interface of the Registry. The Registry supports the following query capabilities:

1. Filter Query
2. SQL Query

The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry implementation. The SQL Query mechanism is an optional feature and MAY be provided by a registry implementation. However, if a vendor provides an SQL query capability to an ebXML Registry it SHALL conform to this document. As such this capability is a normative yet optional capability.

In a future version of this specification, the W3C XQuery syntax may be considered as another query syntax.

The Registry will hold a self-describing capability profile that identifies all supported AdhocQuery options. This profile is described in Section **Error! Reference source not found.**

8.1 Ad Hoc Query Request/Response

A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry query mechanisms.

The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously back to the client. The AdhocQueryResponse returns a collection of objects whose element type depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate classes in [ebRIM] such as RegistryObject and RegistryEntry.

Any errors in the query request messages are indicated in the corresponding query response message.

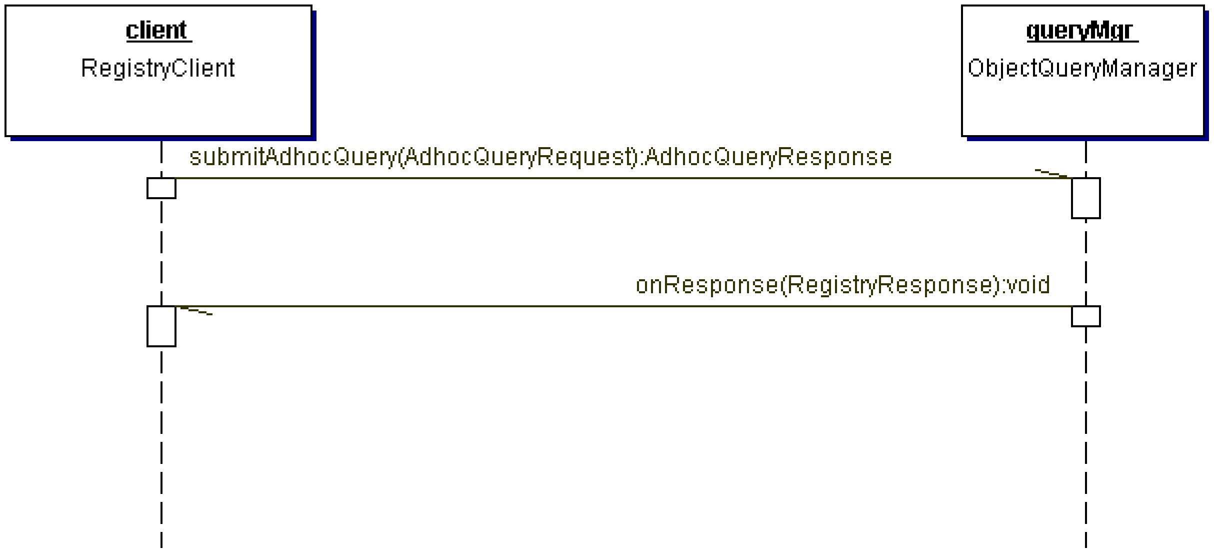

client
RegistryClient
queryMgr
ObjectQueryManager
submitAdhocQuery(AdhocQueryRequest):AdhocQueryResponse
onResponse(RegistryResponse):void

Figure 15: Submit Ad Hoc Query Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix B.2.

Definition

```

<element name="AdhocQueryRequest">
   <complexType>
      <sequence>
         <element ref="tns:ResponseOption" minOccurs="1" maxOccurs="1" />
         <choice minOccurs="1" maxOccurs="1">
            <element ref="FilterQuery" />
            <element ref="SQLQuery" />
         </choice>
      </sequence>
   </complexType>
</element>

<element name="AdhocQueryResponse">
   <complexType>
      <choice>
         <element ref="SQLQueryResult" />
         <element ref="FilterQueryResult" />
      </choice>
   </complexType>
</element>

```

8.1.1 Query Response Options

Purpose

A QueryManagerClient may specify what an ad hoc query must return within an AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest. ResponseOption element has an attribute "returnType” and its values are:

- ObjectRef - This option specifies that the AdhocQueryResponse must contain a collection of ObjectRef XML elements as defined in [RIM schema]. Purpose of this option is to return just the identifiers of the registry objects.

- RegistryObject - This option specifies that the AdhocQueryResponse must contain a collection of RegistryObject XML elements as defined in [RIM schema]. In this case all attributes of the registry objects are returned (objectType, name, description, …) in addition to id attribute.
- RegistryEntry - This option specifies that the AdhocQueryResponse must contain a collection of RegistryEntry XML elements as defined in [RIM schema], which correspond to RegistryEntry attributes.
- LeafClass - This option specifies that the AdhocQueryResponse must contain a collection of XML elements that correspond to leaf classes as defined in [RIM schema].
- LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse must contain a collection of ExtrinsicObject XML elements as defined in [RIM schema] accompanied with their repository items. Linking of ExtrinsicObject and its repository item is done via contentURI as explained in [XXX – Content Retrieval section].

ResponseOption element also has an attribute "returnComposedObjects". It specifies whether or not the whole hierarchy of composed objects are returned with the registry objects.

If "returnType" is higher then the RegistryObject option, then the highest option that satisfies the query is returned. This can be illustrated with a case when OrganizationQuery is asked to return LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then RegistryObject metadata will be returned.

Definition

```
<complexType name="ResponseOptionType">
   <attribute name="returnType" default="RegistryObject">
      <simpleType>
         <restriction base="NMTOKEN">
            <enumeration value="ObjectRef" />
            <enumeration value="RegistryObject" />
            <enumeration value="RegistryEntry" />
            <enumeration value="LeafClass" />
            <enumeration value="LeafClassWithRepositoryItem" />
         </restriction>
      </simpleType>
   </attribute>
   <attribute name="returnComposedObjects" type="boolean" default="false" />
</complexType>
<element name="ResponseOption" type="tns:ResponseOptionType" />
```

8.2 Filter Query Support

FilterQuery is an XML syntax that provides simple query capabilities for any ebXML conforming Registry implementation. Each query alternative is directed against a single class defined by the ebXML Registry Information Model (ebRIM).There are two types of filter queries depending on which classes are queried on.

- Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic queries that might return different subclasses of the class that is queried on. The result of such a query is a set of XML elements that correspond to instances of any class that satisfies the responseOption defined previously in Section 8.1.1. An example might be that RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances that satisfy the query. This implies that response might return XML elements that correspond to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact traversals of these classes. Responses to these queries are accordingly constrained.

A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.1.

Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated with Z. The ebRIM Binding for C might be as in Figure 16

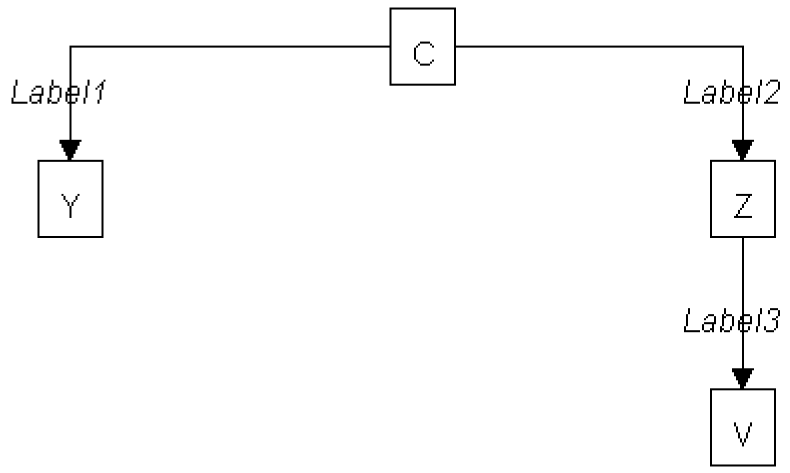


Figure 16: Example ebRIM Binding

Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to which ebRIM association is intended. The name of the query is determined by the root class, i.e. this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are limited to instances that are linked to their parent node by the identified association.

Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a restricted *predicate clause* over the attributes of a single class. Class methods that are defined in ebRIM and that return simple types constitute "visible attributes" that are valid choices for predicate clauses. Names of those attributes will be same as name of the corresponding method just without the prefix 'get'. For example, in case of "getLevelNumber" method the corresponding visible attribute is "levelNumber". The supported class filters are specified in Section 8.2.11 and the supported predicate clauses are defined in Section 8.2.12. A FilterQuery will be composed of elements that traverse the tree to determine which branches satisfy the designated class filters, and the query result will be the set of instances that support such a branch.

In the above example, the CQuery element will have three subelements, one a CFilter on the C class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on the Y class to eliminate branches from C to Y where the target of the association does not satisfy the YFilter, and a third to eliminate branches along a path from C through Z to V. The third element is called a *branch* element because it allows class filters on each class along the path from C to V. In general, a branch element will have subelements that are themselves class filters, other branch elements, or a full-blown query on the class in the path.

If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one branch, filter or query element on Y is allowed. However, if the association is one-to-many, then multiple branch, filter or query elements are allowed. This allows one to specify that an instance of C must have associations with multiple instances of Y before the instance of C is said to satisfy the branch element.

The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their respective superclass queries. Structures of XML elements that match the ebRIM classes are explained in [RIM Schema]. Names of Filters, Queries and Branches correspond to names in ebRIM whenever possible.

The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.10 below identify the virtual hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify the effect of that binding on query semantics.

8.2.1 FilterQuery

Purpose

To identify a set of queries that traverse specific registry class. Each alternative assumes a specific binding to ebRIM. The status is a success indication or a collection of warnings and/or exceptions.

Definition

```

<element name="FilterQuery">
   <complexType>
      <choice minOccurs="1" maxOccurs="1">
         <element ref="tns:RegistryObjectQuery" />
         <element ref="tns:RegistryEntryQuery" />
         <element ref="tns:AuditableEventQuery" />
         <element ref="tns:ClassificationNodeQuery" />
         <element ref="tns:ClassificationSchemeQuery" />
         <element ref="tns:RegistryPackageQuery" />
         <element ref="tns:ExtrinsicObjectQuery" />
         <element ref="tns:OrganizationQuery" />
         <element ref="tns:ServiceQuery" />
      </choice>
   </complexType>
</element>

<element name="FilterQueryResult">
   <complexType>
      <choice minOccurs="1" maxOccurs="1">
         <element ref="tns:RegistryObjectQueryResult" />
         <element ref="tns:RegistryEntryQueryResult" />
```

```
            <element ref="tns:AuditableEventQueryResult" />
            <element ref="tns:ClassificationNodeQueryResult" />
            <element ref="tns:ClassificationSchemeQueryResult" />
            <element ref="tns:RegistryPackageQueryResult" />
            <element ref="tns:ExtrinsicObjectQueryResult" />
            <element ref="tns:OrganizationQueryResult" />
            <element ref="tns:ServiceQueryResult" />
        </choice>
    </complexType>
</element>

```

Semantic Rules

1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
2. Each FilterQueryResult is a set of XML elements to identify each instance of the result set. Each XML attribute carries a value derived from the value of an attribute specified in the Registry Information Model [RIM Schema].
3. For each FilterQuery subelement there is only one corresponding FilterQueryResult subelement that must be returned as a response. Class name of the FilterQueryResult subelement has to match the class name of the FilterQuery subelement.
4. If an error condition is raised during any part of the execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to “failure” and no query result element is returned; instead, a RegistryErrorList element must be returned with its highestSeverity element set to “error”. At least one of the RegistryError elements in the RegistryErrorList will have its severity attribute set to “error”.
5. If no error conditions are raised during execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to “success” and an appropriate query result element must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of the RegistryErrorList is set to “warning” and the serverity attribute of each RegistryError is set to “warning”.

8.2.2 RegistryObjectQuery

Purpose

To identify a set of registry object instances as the result of a query over selected registry metadata.

ebRIM Binding

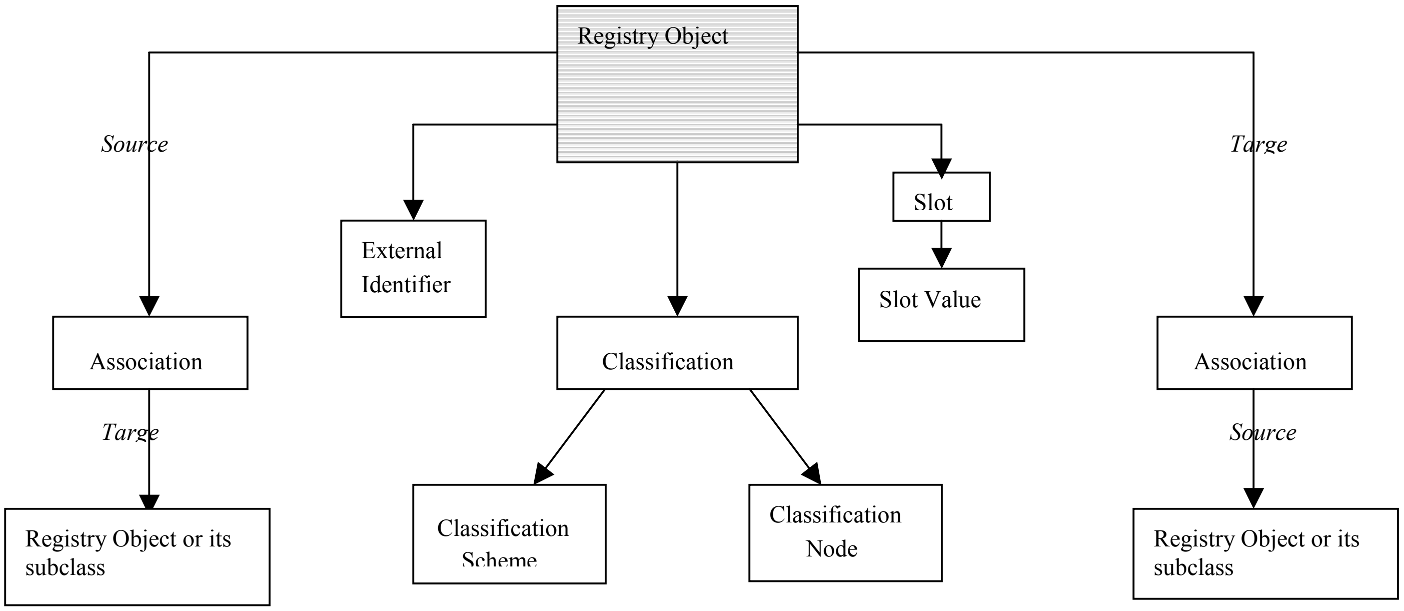


Figure 17: ebRim Binding for RegistryObjectQuery

Definition

```
<complexType name="RegistryObjectQueryType">
  <sequence>
    <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
    <element name="NameBranch" type="InternationalStringBranchType" minOccurs="0" maxOccurs="1" />
    <element name="DescriptionBranch" type="InternationalStringBranchType" minOccurs="0" maxOccurs="1" />
    <element ref="SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:ClassificationBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>
<element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />

<complexType name="LeafRegistryObjectListType">
  <choice minOccurs="0" maxOccurs="unbounded">
    <element ref="tns:ObjectRef" />
    <element ref="tns:Association" />
    <element ref="tns:AuditableEvent" />
    <element ref="tns:Classification" />
    <element ref="tns:ClassificationNode" />
    <element ref="tns:ClassificationScheme" />
    <element ref="tns:ExternalIdentifier" />
    <element ref="tns:ExternalLink" />
    <element ref="tns:ExtrinsicObject" />
    <element ref="tns:Organization" />
    <element ref="tns:RegistryPackage" />
    <element ref="tns:Service" />
    <element ref="tns:ServiceBinding" />
    <element ref="tns:SpecificationLink" />
    <element ref="tns:User" />
  </choice>
</complexType>

<complexType name="RegistryObjectListType">
  <complexContent>
```

```
      <extension base="tns:LeafRegistryObjectListType">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="tns:RegistryEntry" />
          <element ref="tns:RegistryObject" />
        </choice>
      </extension>
    </complexContent>
</complexType>
<element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />

<complexType name="InternationalStringBranchType">
  <sequence>
    <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>

<complexType name="AssociationBranchType">
  <sequence>
    <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
    <choice minOccurs="0" maxOccurs="1">
      <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ClassificationBranch" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
      <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
      <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="1" />
      <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="1" />
    </choice>
  </sequence>
</complexType>
<element name="SourceAssociationBranch" type="AssociationBranchType" />
<element name="TargetAssociationBranch" type="AssociationBranchType" />

<element name="ClassificationBranch">
  <complexType>
    <sequence>
      <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
      <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
    </sequence>
  </complexType>
</element>

<element name="SlotBranch">
  <complexType>
    <sequence>
      <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>

<element name="UserBranch">
  <complexType>
    <sequence>
      <element ref="tns:UserFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
      <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
    </sequence>
  </complexType>
</element>

<element name="ServiceBindingBranch">
  <complexType>
    <sequence>
      <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
      <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>

<element name="SpecificationLinkBranch">
  <complexType>
    <sequence>
      <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
      <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" minOccurs="0"
        maxOccurs="1" />
      <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" minOccurs="0"
        maxOccurs="1" />
    </sequence>
  </complexType>
</element>
```

Semantic Rules

1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The following steps will eliminate instances in RO that do not satisfy the conditions of the specified filters.

   a) If RO is empty then continue below.

   b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from RO. If RO is empty then continue below. If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a name then remove x from RO. If RO is empty then continue below; otherwise treat NameBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings that constitute the name of the registry object then remove x from RO. If RO is empty then continue below.

c) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a name then remove x from RO. If RO is empty then continue below; otherwise treat DescriptionBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings that constitute the description of the registry object then remove x from RO. If RO is empty then continue below.

d) If a SourceAssociationBranch element is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the source object of some Association instance, then remove x from RO. If RO is empty then continue below; otherwise, treat each SourceAssociationBranch element separately as follows:
If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be the set of all Association instances that have x as a source object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the source object. If AF is empty, then remove x from RO.

If RO is empty then continue below.

If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let ROT be the set of ClassificationScheme instances that satisfy the ClassificationSchemeQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Organization instances that satisfy the OrganizationQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If an AuditableEventQuery is specified within the SourceAssociationBranch, then let ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. RO is empty then continue below.

If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the set of Service instances that satisfy the ServiceQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set of User instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let u be the member of ROT. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationBranch is specified within the SourceAssociationBranch then let ROT be the set of Classification instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let cb be the member of ROT. If ClassificationFilter element is specified within the ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue below. if a ClassificationSchemeQuery element is specified within the ClassificationBranch then replace ROT by the set of remaining Classification instances in ROT whose defining classification scheme satisfies the ClassificationSchemeQuery. If ROT is empty, then remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery element is specified within the ClassificationBranch, then replace ROT by the set of remaining Classification instances in ROT for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery. If ROT is empty, then remove x from RO. If RO is empty then continue below..

If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let ROT be the set of ServiceBinding instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let sb be the member of ROT. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below. If a SpecificationLinkBranch is specified within the ServiceBindingBranch then consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in ROT. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below.

If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let ROT be the set of SpecificationLink instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below. Let sl be the member of ROT. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue below.

If a SourceAssociationBranch is specified within the SourceAssociationBranch, then let ROT be the set of RegistryObject instances that satisfy the SourceAssociationBranch and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

If a TargetAssociationBranch is specified within the SourceAssociationBranch, then let ROT be the set of RegistryObject instances that satisfy the TargetAssociationBranch and are the source object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue below.

e) If a TargetAssociationBranch element is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the target object of some Association instance, then remove x from RO. If RO is empty then continue below; otherwise, treat each TargetAssociationBranch element separately as follows:

If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be the set of all Association instances that have x as a target object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the target object. If AF is empty, then remove x from RO. If RO is empty then continue below.

If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below

If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS be the set of

RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let ROS be the set of ClassificationScheme instances that satisfy the ClassificationSchemeQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Organization instances that satisfy the OrganizationQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If an AuditableEventQuery is specified within the TargetAssociationBranch, then let ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. RO is empty then continue below.

If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the set of Service instances that satisfy the ServiceQuery and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set of User instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let u be the member of ROS. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a ClassificationBranch is specified within the TargetAssociationBranch then let ROS be the set of Classification instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let cb be the member of ROS. If ClassificationFilter element is specified within the ClassificationBranch, and if cb does not satisfy that filter, then remove cb from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue below. if a ClassificationSchemeQuery element is specified within the ClassificationBranch then replace ROS by the set of remaining Classification instances in ROS whose defining classification scheme satisfies the ClassificationSchemeQuery. If ROS is empty, then remove x from RO. If RO is empty then continue below. If a ClassificationNodeQuery element is specified within the ClassificationBranch, then replace ROS by the set of remaining Classification instances in ROS for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let ROS be the set of ServiceBinding instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let sb be the member of ROS. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below. If a SpecificationLinkBranch is specified within the ServiceBindingBranch then consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in ROS. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below.

If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let ROS be the set of SpecificationLink instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below. Let sl be the member of ROS. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue below

If a SourceAssociationBranch is specified within the TargetAssociationBranch, then let ROS be the set of RegistryObject instances that satisfy the SourceAssociationBranch and are the target object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

If a TargetAssociationBranch is specified within the TargetAssociationBranch, then let ROS be the set of RegistryObject instances that satisfy the TargetAssociationBranch and are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue below.

f) If a ClassificationBranch element is not specified, or if RO is empty, then continue below; otherwise, let x be a remaining registry object in RO. If x is not the classifiedObject of some Classification instance, then remove x from RO; otherwise, treat each ClassificationBranch element separately as follows:

If no ClassificationFilter is specified within the ClassificationBranch, then let CL be the set of all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then remove x from RO and continue below. Otherwise, if CL is not empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of remaining Classification instances in CL whose defining classification scheme satisfies the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and continue below. Otherwise, if CL remains not empty, and if a ClassificationNodeQuery is specified, then replace CL by the set of remaining Classification instances in CL for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then remove x from RO. If RO is empty then continue below.

g) If an ExternalIdentifierFilter element is not specified, or if RO is empty, then continue below; otherwise, let x be a remaining registry object in RO. If x is not linked to some ExternalIdentifier instance, then remove x from RO; otherwise, treat each ExternalIdentifierFilter element separately as follows: Let EI be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RO. If RO is empty then continue below.

h) If a SlotBranch element is not specified, or if RO is empty, then continue below; otherwise, let x be a remaining registry object in RO. If x is not linked to some Slot instance, then remove x from RO. If RO is empty then continue below; otherwise, treat each SlotBranch element separately as follows: If a SlotFilter is not specified within the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty, then remove x from RO and continue below. Otherwise, if SL remains not empty, and if a SlotValueFilter is specified, replace SL by the set of remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is empty, then remove x from RO.

2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, return RO as the result ofb the RegistryObjectQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the RegistryObjectQuery.

Examples

A client application needs all items that are classified by two different classification schemes, one based on "Industry" and another based on "Geography". Both schemes have been defined by ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The following query identifies registry entries for all registered items that are classified by Industry as any subnode of "Automotive" and by Geography as any subnode of "Asia/Japan".

```

<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryEntry"/>
  <FilterQuery>
    <RegistryObjectQuery>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "path">
              <StringClause stringPredicate = "equal">//Automotive</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "equal">urn:ebxml:cs:industry</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassifiedByBranch>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "path">
              <StringClause stringPredicate = "startswith">/Geography-id/Asia/Japan</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "equal">urn:ebxml:cs:geography</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassifiedByBranch>
    </RegistryObjectQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client application wishes to identify all RegistryObject instances that are classified by some internal classification scheme and have some given keyword as part of the description of one of the classification nodes of that classification scheme. The following query identifies all such RegistryObject instances. The query takes advantage of the knowledge that the classification scheme is internal, and thus that all of its nodes are fully described as ClassificationNode instances.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryObject"/>
  <FilterQuery>
    <RegistryObjectQuery>
      <ClassifiedByBranch>
        <ClassificationNodeQuery>
          <DescriptionBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
```

```
                  <StringClause stringPredicate = "equal">transistor</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </DescriptionBranch>
        </ClassificationNodeQuery>
      </ClassifiedByBranch>
    </RegistryObjectQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.3 RegistryEntryQuery

Purpose

To identify a set of registry entry instances as the result of a query over selected registry metadata.

ebRIM Binding

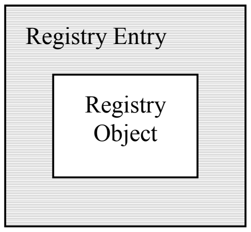


Figure 18:ebRIM Binding for RegistryEntryQuery

Definition

```
<complexType name="RegistryEntryQueryType">
    <complexContent>
        <extension base="tns:RegistryObjectQueryType">
            <sequence>
                <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
            </sequence>
        </extension>
    </complexContent>
</complexType>
<element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />

<element name="RegistryEntryQueryResult">
    <complexType>
        <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="rim:ObjectRef" />
            <element ref="rim:ClassificationScheme" />
            <element ref="rim:ExtrinsicObject" />
            <element ref="rim:RegistryEntry" />
            <element ref="rim:RegistryObject" />
            <element ref="rim:RegistryPackage" />
        </choice>
    </complexType>
</element>
```

Semantic Rules

1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
   a) If RE is empty then continue below.
   b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE. If RE is empty then continue below.
   c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited RegistryObjectQuery over RE as explained in section 8.2.2.
2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return RE as the result of the RegistryEntryQuery.
3. Return any accumulated warnings or exceptions as the StatusResult associated with the RegistryEntryQuery.

Examples

A client wishes to establish a trading relationship with XYZ Corporation and wants to know if they have registered any of their business documents in the Registry. The following query returns a set of registry entry identifiers for currently registered items submitted by any organization whose name includes the string "XYZ". It does not return any registry entry identifiers for superseded, replaced, deprecated, or withdrawn items.

```

<AdhocQueryRequest>
  <ResponseOption returnType = "ObjectRef"/>
  <FilterQuery>
    <RegistryEntryQuery>
      <SourceAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument = "associationType">
              <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
               <SimpleClause leftArgument = "value">
                 <StringClause stringPredicate = "contains">XYZ</StringClause>
               </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </OrganizationQuery>
      </SourceAssociationBranch>
      <RegistryEntryFilter>
        <Clause>
          <SimpleClause leftArgument = "status">
            <StringClause stringPredicate = "equal">Approved</StringClause>
          </SimpleClause>
        </Clause>
      </RegistryEntryFilter>
    </RegistryEntryQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client is using the United Nations Standard Product and Services Classification (UNSPSC) scheme and wants to identify all companies that deal with products classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "RegistryEntry"/>
  <FilterQuery>
    <RegistryEntryQuery>
      <ClassifiedByBranch>
        <ClassificationFilter>
          <Clause>
            <SimpleClause leftArgument = "code">
              <StringClause stringPredicate = "equal">321118</StringClause>
            </SimpleClause>
          </Clause>
        </ClassificationFilter>
        <ClassificationSchemeQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "equal">urn:org:un:spsc:cs2001</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </ClassificationSchemeQuery>
      </ClassifiedByBranch>
      <RegistryEntryFilter>
        <Clause>
          <CompoundClause connectivePredicate = "And">
            <Clause>
              <SimpleClause leftArgument = "objectType">
                <StringClause stringPredicate = "equal">CPP</StringClause>
              </SimpleClause>
            </Clause>
            <Clause>
              <SimpleClause leftArgument = "status">
                <StringClause stringPredicate = "equal">Approved</StringClause>
              </SimpleClause>
            </Clause>
          </CompoundClause>
        </Clause>
      </RegistryEntryFilter>
    </RegistryEntryQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.4 AuditableEventQuery

Purpose

To identify a set of auditable event instances as the result of a query over selected registry metadata.

ebRIM Binding

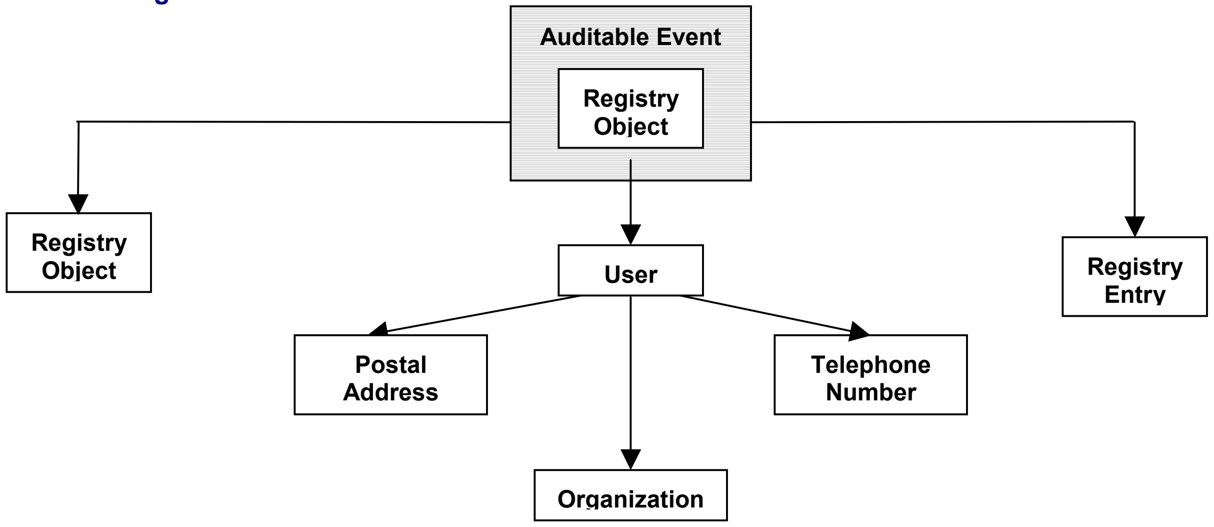


Figure 19: ebRim binding for AuditableEventQuery

Definition

```

<complexType name="AuditableEventQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:AuditableEventFilter" minOccurs="0" />
        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />

<element name="AuditableEventQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:AuditableEvent" />
    </choice>
  </complexType>
</element>

```

Semantic Rules

1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The following steps will eliminate instances in AE that do not satisfy the conditions of the specified filters.

   a) If AE is empty then continue below.
   b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from AE. If AE is empty then continue below.
   c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not an auditable event for some registry object in RO, then remove x from AE. If AE is empty then continue below.
   d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an auditable event for some registry entry in RE, then remove x from AE. If AE is empty then continue below.
   e) If an UserBranch element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove x from AE. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove x from AE. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove x from AE. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.9 then remove x from AE. If AE is empty then continue below.
   f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited RegistryObjectQuery over AE as explained in section 8.2.2.
2. If AE is empty, then raise the warning: ***auditable event query result is empty***; otherwise return AE as the result of the AuditableEventQuery.
3. Return any accumulated warnings or exceptions as the StatusResult associated with the AuditableEventQuery.

Examples

A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The client is now interested in all events since the beginning of the year that have impacted that item. The following query will return a set of AuditableEvent instances for all such events.

```

<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <AuditableEventQuery>
      <TargetAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument = "associationType">
              <StringClause stringPredicate = "equal">AuditedBy</StringClause>
            </SimpleClause>
          </Clause>
```

```
            </AssociationFilter>
            <RegistryEntryQuery>
               <NameBranch>
                  <LocalizedStringFilter>
                     <Clause>
                        <SimpleClause leftArgument = "value">
                           <StringClause stringPredicate = "equal">urn:path:myitem</StringClause>
                        </SimpleClause>
                     </Clause>
                  </LocalizedStringFilter>
               </NameBranch>
            </RegistryEntryQuery>
         </TargetAssociationBranch>
         <AuditableEventFilter>
            <Clause>
               <SimpleClause leftArgument = "timestamp">
                  <RationalClause logicalPredicate = "GE">
                     <IntClause>20000101</IntClause>
                  </RationalClause>
               </SimpleClause>
            </Clause>
         </AuditableEventFilter>
      </AuditableEventQuery>
   </FilterQuery>
</AdhocQueryRequest
```

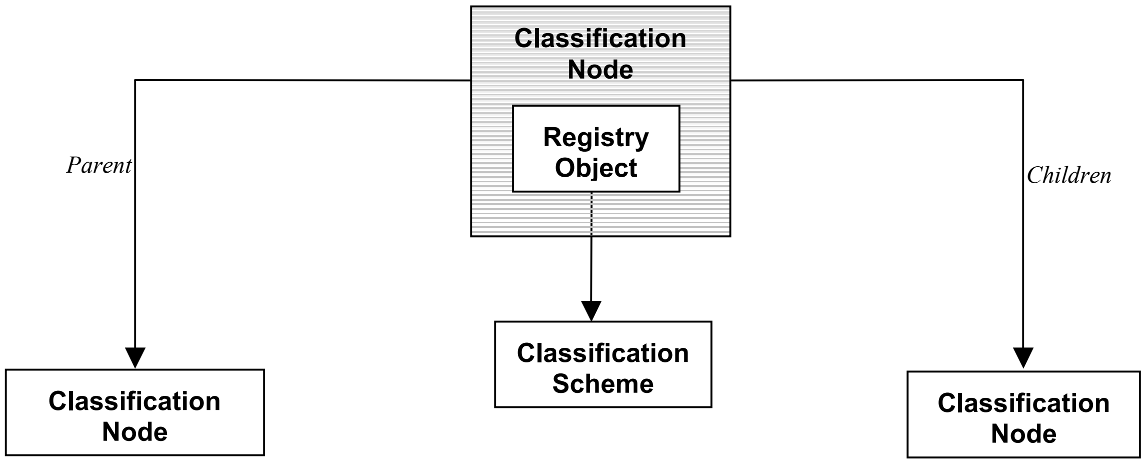
A client company has many registered objects in the Registry. The Registry allows events submitted by other organizations to have an impact on your registered items, e.g. new classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".

```
<AdhocQueryRequest>
   <ResponseOption returnType = "LeafClass"/>
   <FilterQuery>
      <AuditableEventQuery>
         <TargetAssociationBranch>
            <AssociationFilter>
               <Clause>
                  <SimpleClause leftArgument = "associationType">
                     <StringClause stringPredicate = "equal">AuditedBy</StringClause>
                  </SimpleClause>
               </Clause>
            </AssociationFilter>
            <RegistryEntryQuery>
               <SourceAssociationBranch>
                  <AssociationFilter>
                     <Clause>
                      <SimpleClause leftArgument = "associationType">
                       <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
                      </SimpleClause>
                     </Clause>
                  </AssociationFilter>
                  <OrganizationQuery>
                     <NameBranch>
                        <LocalizedStringFilter>
                         <Clause>
                          <SimpleClause leftArgument = "value">
                        <StringClause stringPredicate = "equal">myorg</StringClause>
                        </SimpleClause>
                      </Clause>
                    </LocalizedStringFilter>
                  </NameBranch>
                </OrganizationQuery>
              </SourceAssociationBranch>
            </RegistryEntryQuery>
          </TargetAssociationBranch>
          <UserBranch>
            <OrganizationQuery>
              <NameBranch>
                <LocalizedStringFilter>
                  <Clause>
                    <SimpleClause leftArgument = "value">
                      <StringClause stringPredicate = "-equal">myorg</StringClause>
                    </SimpleClause>
                  </Clause>
                </LocalizedStringFilter>
              </NameBranch>
            </OrganizationQuery>
          </UserBranch>
        </AuditableEventQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.5 ClassificationNodeQuery

Purpose

To identify a set of classification node instances as the result of a query over selected registry metadata.



ebRIM Binding

Figure 20: ebRim binding for ClassificationNodeQuery

Definition

```
<complexType name="ClassificationNodeQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
```

```
      <sequence>
        <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
        <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
          maxOccurs="1" />
        <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />

<element name="ClassificationNodeQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:ClassificationNode" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The following steps will eliminate instances in CN that do not satisfy the conditions of the specified filters.
   a) If CN is empty then continue below.
   b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove x from CN. If CN is empty then continue below.
   c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification node in CN. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in section 8.2.6, then remove x from CN. If CN is empty then continue below.
   d) If a ClassificationNodeParentBranch element is not specified, then go to the next step; otherwise, let x be a remaining classification node in CN and execute the following paragraph with n=x.

      Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base level node), then remove x from CN and go to the next step; otherwise, let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in the ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter, then remove x from CN. If CN is empty then continue below. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeParentBranch and if defining classification scheme of p does not satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue below.

      If another ClassificationNodeParentBranch element is directly contained within this ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.

e) If a ClassificationNodeChildrenBranch element is not specified, then continue below; otherwise, let x be a remaining classification node in CN. If x is not the parent node of some ClassificationNode instance, then remove x from CN and if CN is empty continue below; otherwise, treat each ClassificationNodeChildrenBranch element separately and execute the following paragraph with n = x.

   Let n be a classification node instance. If a ClassificationNodeFilter element is not specified within the ClassificationNodeChildrenBranch element then let CNC be the set of all classification nodes that have n as their parent node; otherwise, let CNC be the set of all classification nodes that satisfy the ClassificationNodeFilter and have n as their parent node. If CNC is empty, then remove x from CN and if CN is empty continue below; otherwise, let c be any member of CNC. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeChildrenBranch and if the defining classification scheme of c does not satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then remove x from CN. If CN is empty then continue below; otherwise, let y be an element of CNC and continue with the next paragraph.

   If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly contain another ClassificationNodeChildrenBranch element, then continue below; otherwise, repeat the previous paragraph with the new ClassificationNodeChildrenBranch element and with n = y.

f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited RegistryObjectQuery over CN as explained in section 8.2.2..

2. If CN is empty, then raise the warning: ***classification node query result is empty***; otherwise return CN as the result of the ClassificationNodeQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the ClassificationNodeQuery.

Path Filter Expression usage in ClassificationNodeFilter

The path filter expression is used to match classification nodes in ClassificationNodeFilter elements involving the path attribute of the ClassificationNode class as defied by the getPath method in [ebRIM].

The path filter expressions are based on a very small and proper sub-set of location path syntax of XPath.

The path filter expression syntax includes support for matching multiple nodes by using wild card syntax as follows:

- Use of '*' as a wildcard in place of any path element in the pathFilter
- Use of '//' syntax to denote any descendent of a node in the pathFilter

It is defined by the following BNF grammar:

```

pathFilter     ::= '/' schemeId nodePath
nodePath       ::= slashes nodeCode
               |    slashes '*'
               |    slashes nodeCode ( nodePath )?
Slashes ::= '/' | '//'

```

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

The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause is a PATH Filter expression.

This is illustrated in the following example that matches all second level nodes in ClassificationScheme with id ‘Geography-id’ and with code ‘Japan’:

```
<ClassificationNodeQuery>
  <ClassificationNodeFilter>
    <Clause>
      <SimpleClause leftArgument = "path">
        <StringClause stringPredicate = "equal">//Geography-id/*/Japan</StringClause>
      </SimpleClause>
    </Clause>
  </ClassificationNodeFilter>
</ClassificationNodeQuery>
```

Use Cases and Examples of Path Filter Expressions

The following table lists various use cases and examples using the sample Geography scheme below:

```
<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" />
```

Table 10: Path Filter Expressions for Use Cases

Use Case	PATH Expression	Description
Match all nodes in first level that have a specified value	/Geography-id/NorthAmerica	Find all first level nodes whose code is 'NorthAmerica'
Find all children of first level node whose code is “NorthAmerica”	/Geography-id/NorthAmerica/*	Match all nodes whose first level path element has code "NorthAmerica"
Match all nodes that have a specified value regardless of level	/ Geography-id//Japan	Find all nodes with code "Japan”
Match all nodes in the second level that have a specified value	/Geography-id/*/Japan	Find all second level nodes with code 'Japan'
Match all nodes in the 3rd level that have a specified value	/ Geography-id/*/*/Tokyo	Find all third level nodes with code 'Tokyo'

Examples

A client application wishes to identify all of the classification nodes in the first three levels of a classification scheme hierarchy. The client knows that the name of the underlying classification scheme is “urn:ebxml:cs:myscheme”. The following query identifies all nodes at the first three levels.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ClassificationNodeQuery>
      <ClassificationNodeFilter>
        <Clause>
          <SimpleClause leftArgument = "levelNumber">
            <RationalClause logicalPredicate = "LE">
             <IntClause>3</IntClause>
            </RationalClause>
          </SimpleClause>
        </Clause>
      </ClassificationNodeFilter>
      <ClassificationSchemeQuery>
        <NameBranch>
          <LocalizedStringFilter>
            <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "equal">urn:ebxml:cs:myscheme</StringClause>
                </SimpleClause>
            </Clause>
          </LocalizedStringFilter>
        </NameBranch>
      </ClassificationSchemeQuery>
    </ClassificationNodeQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

If, instead, the client wishes all levels returned, they could simply delete the ClassificationNodeFilter element from the query.

The following query finds all children nodes of a first level node whose code is NorthAmerica.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ClassificationNodeQuery>
      <ClassificationNodeFilter>
       <Clause>
        <SimpleClause leftArgument = "path">
          <StringClause stringPredicate = "equal">/Geography-id/NorthAmerica/*</StringClause>
        </SimpleClause>
       </Clause>
      </ClassificationNodeFilter>
    </ClassificationNodeQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

The following query finds all third level nodes with code of Tokyo.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
  <FilterQuery>
    <ClassificationNodeQuery>
      <ClassificationNodeFilter xmlns = "">
        <Clause>
          <SimpleClause leftArgument = "path">
```

```
                <StringClause stringPredicate = "equal">/Geography-id/*/*/Tokyo</StringClause>
              </SimpleClause>
            </Clause>
         </ClassificationNodeFilter>
      </ClassificationNodeQuery>
   </FilterQuery>
</AdhocQueryRequest>
```

8.2.6 ClassificationSchemeQuery

Purpose

To identify a set of classification scheme instances as the result of a query over selected registry metadata.

ebRIM Binding

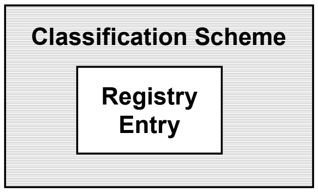


Figure 21: ebRIM Binding for ClassificationSchemeQuery

Definition

```

<complexType name="ClassificationSchemeQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
```

Semantic Rules

1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The following steps will eliminate instances in CS that do not satisfy the conditions of the specified filters.
   a) If CS is empty then continue below.
   b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter, then remove x from CS. If CS is empty then continue below.
   c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited RegistryEntryQuery over CS as explained in section 8.2.3.
2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise, return CS as the result of the ClassificationSchemeQuery.
3. Return any accumulated warnings or exceptions as the StatusResult associated with the ClassificationSchemeQuery.

8.2.7 RegistryPackageQuery

Purpose

To identify a set of registry package instances as the result of a query over selected registry metadata.

ebRIM Binding

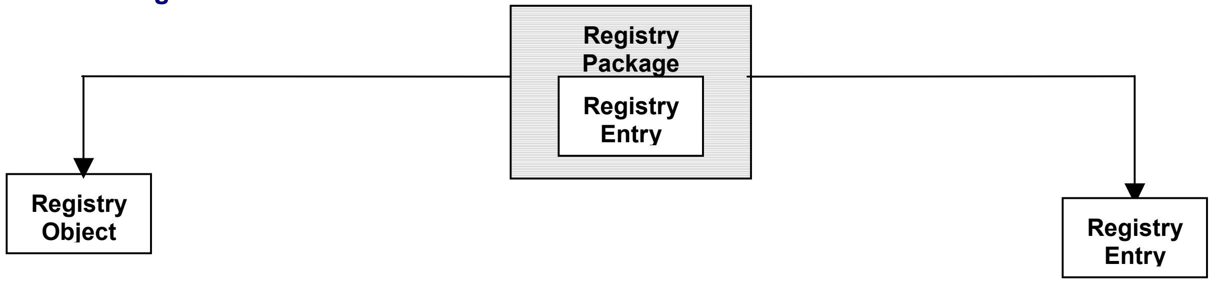


Figure 22: ebRim binding for RegistryPackageQuery

Definition

```

<complexType name="RegistryPackageQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />
            <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />
            <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />

<element name="RegistryPackageQueryResult">
   <complexType>
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="rim:ObjectRef" />
         <element ref="rim:RegistryEntry" />
         <element ref="rim:RegistryObject" />
         <element ref="rim:RegistryPackage" />
      </choice>
   </complexType>
</element>

```

Semantic Rules

1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The following steps will eliminate instances in RP that do not satisfy the conditions of the specified filters.
   a) If RP is empty then continue below.

   b) If a RegistryPackageFilter is not specified, then continue below; otherwise, let x be a registry package instance in RP. If x does not satisfy the RegistryPackageFilter then remove x from RP. If RP is empty then continue below.

   c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery element then let RO be the set of RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2 and let PO be the subset of RO that are members of the package x. If PO is empty, then remove x from RP. If RP is empty then continue below. If a RegistryEntryQuery element is directly contained in the RegistryPackageQuery element then let RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be the subset of RE that are members of the package x. If PE is empty, then remove x from RP.

   d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited RegistryEntryQuery over RP as explained in section 8.2.3.

2. If RP is empty, then raise the warning: ***registry package query result is empty***; otherwise return RP as the result of the RegistryPackageQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the RegistryPackageQuery.

Examples

A client application wishes to identify all package instances in the Registry that contain an Invoice extrinsic object as a member of the package.

```

<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <RegistryPackageQuery>
      <RegistryEntryQuery>
        <RegistryEntryFilter>
          <Clause>
            <SimpleClause leftArgument = "objectType">
              <StringClause stringPredicate = "equal">Invoice</StringClause>
            </SimpleClause>
          </Clause>
        </RegistryEntryFilter>
      </RegistryEntryQuery>
    </RegistryPackageQuery>
  </FilterQuery>
</AdhocQueryRequest>

```

A client application wishes to identify all package instances in the Registry that are not empty.

```

<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <RegistryPackageQuery/>
  </FilterQuery>
</AdhocQueryRequest>

```

A client application wishes to identify all package instances in the Registry that are empty. Since the RegistryPackageQuery is not set up to do negations, clients will have to do two separate RegistryPackageQuery requests, one to find all packages and another to find all non-empty packages, and then do the set difference themselves. Alternatively, they could do a more complex RegistryEntryQuery and check that the packaging association between the package and its members is non-existent.

Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an equivalent RegistryEntryQuery using appropriate "Source" and "Target" associations. However, the equivalent RegistryEntryQuery is often more complicated to write.

8.2.8 ExtrinsicObjectQuery

Purpose

To identify a set of extrinsic object instances as the result of a query over selected registry metadata.

ebRIM Binding

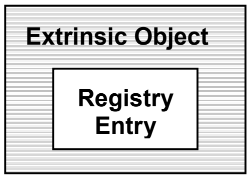


Figure 23:ebRIM Binding for ExtrinsicObjectQuery

Definition

```

<complexType name="ExtrinsicObjectQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />

<element name="ExtrinsicObjectQueryResult">
   <complexType>
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="rim:ObjectRef" />
         <element ref="rim:RegistryEntry" />
         <element ref="rim:RegistryObject" />
         <element ref="rim:ExtrinsicObject" />
      </choice>
   </complexType>
</element>

```

Semantic Rules

1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The following steps will eliminate instances in EO that do not satisfy the conditions of the specified filters.

   a) If EO is empty then continue below.

   b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from EO. If EO is empty then continue below.

   c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited RegistryEntryQuery over EO as explained in section 8.2.3.

2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, return EO as the result of the ExtrinsicObjectQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the ExtrinsicObjectQuery.

8.2.9 OrganizationQuery

Purpose

To identify a set of organization instances as the result of a query over selected registry metadata.

ebRIM Binding

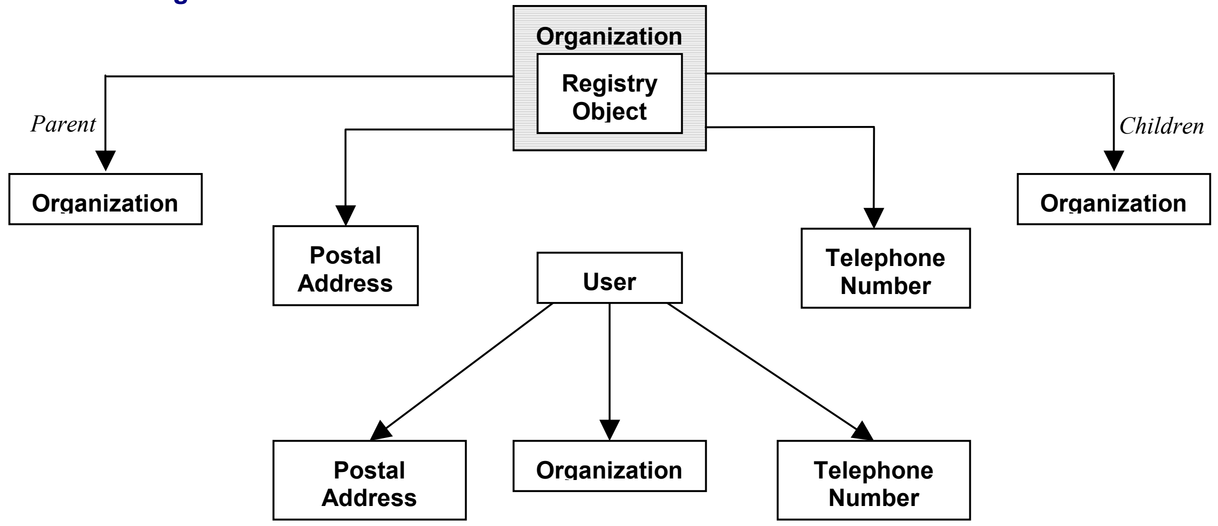


Figure 24: ebRim Binding for OrganizationQuery

Definition

```
<complexType name="OrganizationQueryType">
   <complexContent>
      <extension base="tns:RegistryObjectQueryType">
         <sequence>
            <element ref="tns:OrganizationFilter" minOccurs="0" maxOccurs="1" />
            <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
            <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
            <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
            <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0
               " maxOccurs="1" />
            <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
                maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

```
<element name="OrganizationQuery" type="tns:OrganizationQueryType" />

<element name="OrganizationQueryResult">
  <complexType>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element ref="rim:ObjectRef" />
      <element ref="rim:RegistryObject" />
      <element ref="rim:Organization" />
    </choice>
  </complexType>
</element>
```

Semantic Rules

1. Let ORG denote the set of all persistent Organization instances in the Registry. The following steps will eliminate instances in ORG that do not satisfy the conditions of the specified filters.

   a) If ORG is empty then continue below.

   b) If an OrganizationFilter element is not directly contained in the OrganizationQuery element, then go to the next step; otherwise let x be an organization instance in ORG. If x does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then continue below.

   c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is empty then continue below.

   d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then remove x from ORG. If ORG is empty then continue below.

   e) If a UserBranch element is not directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user instance that is affiliated with x. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then remove x from ORG. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in section 8.2.9 then remove x from ORG. If ORG is empty then continue below.

   f) If a OrganizationParentBranch element is not specified within the OrganizationQuery, then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the following paragraph with o = x:

Let o be an organization instance. If an OrganizationFilter is not specified within the OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the Organization hierarchy), then remove x from ORG; otherwise, let p be the parent organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG. If ORG is empty then continue below.

If another OrganizationParentBranch element is directly contained within this OrganizationParentBranch element, then repeat the previous paragraph with o = p.

g) If a OrganizationChildrenBranch element is not specified, then continue below; otherwise, let x be a remaining organization in ORG. If x is not the parent node of some organization instance, then remove x from ORG and if ORG is empty continue below; otherwise, treat each OrganizationChildrenBranch element separately and execute the following paragraph with n = x.

Let n be an organization instance. If an OrganizationFilter element is not specified within the OrganizationChildrenBranch element then let ORGC be the set of all organizations that have n as their parent node; otherwise, let ORGC be the set of all organizations that satisfy the OrganizationFilter and have n as their parent node. If ORGC is empty, then remove x from ORG and if ORG is empty continue below; otherwise, let c be any member of ORGC. If a PostalAddressFilter element is directly contained in the OrganizationChildrenBranch and if the postal address of c does not satisfy the PostalAddressFilter then remove c from ORGC. If ORGC is empty then remove x from ORG. If ORG is empty then continue below. If no TelephoneNumberFilter element is directly contained in the OrganizationChildrenBranch and if If any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of c then remove c from ORGC. If ORGC is empty then remove x from ORG. If ORG is empty then continue below; otherwise, let y be an element of ORGC and continue with the next paragraph.

If the OrganizationChildrenBranch element is terminal, i.e. if it does not directly contain another OrganizationChildrenBranch element, then continue below; otherwise, repeat the previous paragraph with the new OrganizationChildrenBranch element and with n = y.

h) Let ORG be the set of remaining Organization instances. Evaluate inherited RegistryObjectQuery over ORG as explained in section 8.2.2.

2. If ORG is empty, then raise the warning: *organization query result is empty*; otherwise return ORG as the result of the OrganizationQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the OrganizationQuery.

Examples

A client application wishes to identify a set of organizations, based in France, that have submitted a PartyProfile extrinsic object this year.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "True”/>
  <FilterQuery>
        <OrganizationQuery>
            <TargetAssociationBranch>
                <AssociationFilter>
                    <Clause>
                        <SimpleClause leftArgument = "associationType">
                            <StringClause stringPredicate = "equal">SubmittedBy</StringClause>
                        </SimpleClause>
                    </Clause>
                </AssociationFilter>
                <RegistryObjectQuery>
                    <RegistryObjectFilter>
                        <Clause>
                            <SimpleClause leftArgument = "objectType">
                                <StringClause stringPredicate = "equal">CPP</StringClause>
                            </SimpleClause>
                        </Clause>
                    </RegistryObjectFilter>
                    <SourceAssociationBranch>
                        <AssociationFilter>
                            <Clause>
                                <SimpleClause leftArgument = "associationType">
                                        <StringClause stringPredicate =
  "equal">AuditedBy</StringClause>
                                </SimpleClause>
                            </Clause>
                        </AssociationFilter>
                        <AuditableEventQuery>
                            <AuditableEventFilter>
                                <Clause>
                                        <SimpleClause leftArgument = "timestamp">
                                                <RationalClause logicalPredicate = "GE">
                                                        <IntClause>20010101</IntClause>
                                                </RationalClause>
                                        </SimpleClause>
                                </Clause>
                            </AuditableEventFilter>
                        </AuditableEventQuery>
                    </SourceAssociationBranch>
                </RegistryObjectQuery>
            </TargetAssociationBranch>
            <PostalAddressFilter>
                <Clause>
                    <SimpleClause leftArgument = "country">
                        <StringClause stringPredicate = "equal">France</StringClause>
                    </SimpleClause>
                </Clause>
            </PostalAddressFilter>
        </OrganizationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

A client application wishes to identify all organizations that have Corporation named XYZ as a parent.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
        <OrganizationQuery>
            <OrganizationParentBranch>
                <NameBranch>
                    <LocalizedStringFilter>
                        <Clause>
                            <SimpleClause leftArgument = "value">
                                <StringClause stringPredicate = "equal">XYZ</StringClause>
                            </SimpleClause>
                        </Clause>
                    </LocalizedStringFilter>
                </NameBranch>
            </OrganizationParentBranch>
        </OrganizationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.10 ServiceQuery

Purpose

To identify a set of service instances as the result of a query over selected registry metadata.

ebRIM Binding

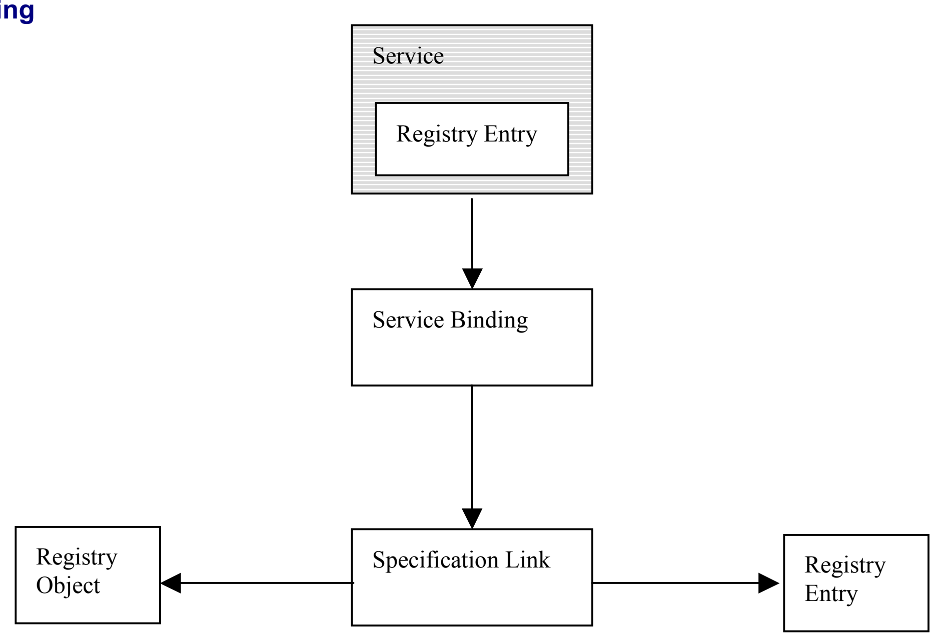


Figure 25:ebRIM Binding for ServiceQuery

Definition

```

<complexType name="ServiceQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:ServiceFilter" minOccurs="0"
               maxOccurs="1" />
            <element ref="tns:ServiceBindingBranch" minOccurs="0"
               maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="ServiceQuery" type="tns:ServiceQueryType" />

<element name="ServiceQueryResult">
   <complexType>
      <choice minOccurs="0" maxOccurs="unbounded">
         <element ref="rim:ObjectRef" />
         <element ref="rim:RegistryObject" />
         <element ref="rim:Service" />
      </choice>
   </complexType>
</element>

```

Semantic Rules

1. Let S denote the set of all persistent Service instances in the Registry. The following steps will eliminate instances in S that do not satisfy the conditions of the specified filters.

   a) If S is empty then continue below.

b) If a ServicetFilter is not specified then go to the next step; otherwise, let x be a service in S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then continue below.

c) If a ServiceBindingBranch is not specified then continue below; otherwise, consider each ServiceBindingBranch element separately as follows:

Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the member of SB. If a ServiceBindingFilter element is specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue below. If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then continue below; otherwise, consider each SpecificationLinkBranch element separately as follows:

Let sb be a remaining service binding in SB. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue below. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue below. If a RegistryEntryQuery element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove x from S. If S is empty then continue below.

d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery over AE as explained in section 8.2.3.

2. If S is empty, then raise the warning: *service query result is empty*; otherwise return S as the result of the ServiceQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the ServiceQuery.

Examples

8.2.11 Registry Filters

Purpose

To identify a subset of the set of all persistent instances of a given registry class.

Definition

```
<complexType name="FilterType">
   <sequence>
      <element ref="tns:Clause" />
   </sequence>
</complexType>
```

```
<element name="RegistryObjectFilter" type="tns:FilterType" />
<element name="RegistryEntryFilter" type="tns:FilterType" />
<element name="ExtrinsicObjectFilter" type="tns:FilterType" />
<element name="RegistryPackageFilter" type="tns:FilterType" />
<element name="OrganizationFilter" type="tns:FilterType" />
<element name="ClassificationNodeFilter" type="tns:FilterType" />
<element name="AssociationFilter" type="tns:FilterType" />
<element name="ClassificationFilter" type="tns:FilterType" />
<element name="ClassificationSchemeFilter" type="tns:FilterType" />
<element name="ExternalLinkFilter" type="tns:FilterType" />
<element name="ExternalIdentifierFilter" type="tns:FilterType" />
<element name="SlotFilter" type="tns:FilterType" />
<element name="AuditableEventFilter" type="tns:FilterType" />
<element name="UserFilter" type="tns:FilterType" />
<element name="SlotValueFilter" type="tns:FilterType" />
<element name="PostalAddressFilter" type="tns:FilterType" />
<element name="TelephoneNumberFilter" type="tns:FilterType" />
<element name="ServiceFilter" type="tns:FilterType" />
<element name="ServiceBindingFilter" type="tns:FilterType" />
<element name="SpecificationLinkFilter" type="tns:FilterType" />
<element name="LocalizedStringFilter" type="tns:FilterType" />

```

Semantic Rules

1. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryObject UML class defined in [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the Clause predicate.
3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True* for the Clause predicate.
4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause predicate.
5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for the Clause predicate.
6. For every OrganizationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Organization or PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The OrganizationFilter returns a set of identifiers for Organization instances whose attribute values evaluate to *True* for the Clause predicate.

7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose attribute values evaluate to *True* for the Clause predicate.

8. For every AssociationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Association UML class defined in [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a set of identifiers for Association instances whose attribute values evaluate to *True* for the Clause predicate.

9. For every ClassificationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Classification UML class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter returns a set of identifiers for Classification instances whose attribute values evaluate to *True* for the Clause predicate.

10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances whose attribute values evaluate to *True* for the Clause predicate.

11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalLink UML class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the Clause predicate.

12. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose attribute values evaluate to *True* for the Clause predicate.

13. For every SlotFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot instances whose attribute values evaluate to *True* for the Clause predicate.

14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to *True* for the Clause predicate.

15. For every UserFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User instances whose attribute values evaluate to *True* for the Clause predicate.

16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one SlotValue instance for each "value" in the "values" list of a Slot instance. The visible attribute of SlotValue is"value". It is a character string. The dynamic instances of SlotValue are derived from the "values" attribute defined in ebRIM for a Slot instance. For every SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify the "value" attribute of the SlotValue class just defined. If not, raise exception: *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose "value" attribute evaluates to *True* for the Clause predicate.
17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the PostalAddress UML class defined in [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True* for the Clause predicate.
18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the TelephoneNumber UML class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*. The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances whose attribute values evaluate to *True* for the Clause predicate.
19. For every ServiceFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM]. If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for Service instances whose attribute values evaluate to *True* for the Clause predicate.
20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to *True* for the Clause predicate.
21. For every SpecificationLinkFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the SpecificationLink UML class defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose attribute values evaluate to *True* for the Clause predicate.
22. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the LocalizedString UML class defined in [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter returns a set of identifiers for LocalizedString instances whose attribute values evaluate to *True* for the Clause predicate.

8.2.12 XML Clause Constraint Representation

NOTE: The Filter Query proposal makes no changes to this Section, however, it still needs to be updated to reflect XML Schema changes.

Purpose

The simple XML FilterQuery utilizes a formal XML structure based on *Predicate Clauses*. Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to simply as ***Clauses*** in this specification.

Conceptual UML Diagram

The following is a conceptual diagram outlining the Clause base structure. It is expressed in UML for visual depiction.

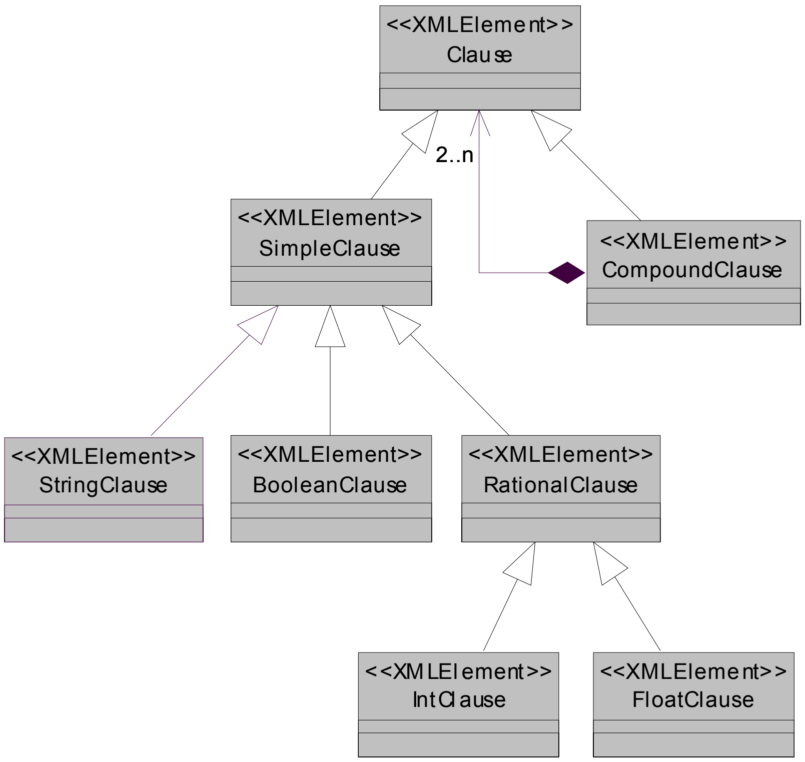

<<XMLElement>> Clause
2..n
<<XMLElement>> SimpleClause
<<XMLElement>> CompoundClause
<<XMLElement>> StringClause
<<XMLElement>> BooleanClause
<<XMLElement>> RationalClause
<<XMLElement>> IntClause
<<XMLElement>> FloatClause

Figure 26: The Clause base structure

Semantic Rules

Predicates and *Arguments* are combined into a "LeftArgument - Predicate - RightArgument" format to form a *Clause*. There are two types of Clauses: *SimpleClauses* and *CompoundClauses*.

<u>SimpleClauses</u>

A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be extended. SimpleClause is extended to support BooleanClause, StringClause, and RationalClause (abstract).

BooleanClause implicitly defines the predicate as ‘equal to’, with the right argument as a boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-compare operations and a right argument as the element’s text data. Rational number support is provided through a common RationalClause providing an enumeration of appropriate rational number compare operations, which is further extended to IntClause and FloatClause, each with appropriate signatures for the right argument.

<u>CompoundClauses</u>

A CompoundClause contains two or more Clauses (Simple or Compound) and a connective predicate. This provides for arbitrarily complex Clauses to be formed.

Definition

```
<!ELEMENT Clause ( SimpleClause | CompoundClause )>
```

```
<!ELEMENT SimpleClause
   ( BooleanClause | RationalClause | StringClause )>
<!ATTLIST SimpleClause
   leftArgument   CDATA #REQUIRED >

<!ELEMENT CompoundClause ( Clause, Clause+ )>
<!ATTLIST CompoundClause
   connectivePredicate ( And | Or ) #REQUIRED>

<!ELEMENT BooleanClause EMPTY >
<!ATTLIST BooleanClause
   booleanPredicate ( True | False ) #REQUIRED>

<!ELEMENT RationalClause ( IntClause | FloatClause )>
<!ATTLIST RationalClause
   logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >

<!ELEMENT IntClause ( #PCDATA )
<!ATTLIST IntClause
   e-dtype NMTOKEN #FIXED 'int' >

<!ELEMENT FloatClause ( #PCDATA )>
<!ATTLIST FloatClause
   e-dtype NMTOKEN #FIXED 'float' >

<!ELEMENT StringClause ( #PCDATA )>
<!ATTLIST StringClause
   stringPredicate
      ( contains | -contains |
        startswith | -startswith |
        equal | -equal
        endswith | -endswith ) #REQUIRED >
```

Examples

Simple BooleanClause: "Smoker" = True

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <SimpleClause leftArgument="Smoker">
      <BooleanClause booleanPredicate="True"/>
   </SimpleClause>
</Clause>
```

Simple StringClause: "Smoker" contains "mo"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <SimpleClause leftArgument="Smoker">
      <StringClause stringcomparepredicate="contains">
         mo
      </StringClause>
   </SimpleClause>
</Clause>
```

Simple IntClause: "Age" >= 7

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <SimpleClause leftArgument="Age">
      <RationalClause logicalPredicate="GE">
         <IntClause e-dtype="int">7</IntClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

Simple FloatClause: "Size" = 4.3

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <SimpleClause leftArgument="Size">
      <RationalClause logicalPredicate="E">
         <FloatClause e-dtype="float">4.3</FloatClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

Compound with two Simples (("Smoker" = False)AND("Age" =< 45))

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
            <BooleanClause booleanPredicate="False"/>
         </SimpleClause>
      </Clause>
      <Clause>
         <SimpleClause leftArgument="Age">
            <RationalClause logicalPredicate="EL">
               <IntClause e-dtype="int">45</IntClause>
            </RationalClause>
         </SimpleClause>
      </Clause>
   </CompoundClause>
</Clause>
```

Coumpound with one Simple and one Compound

(("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
            <BooleanClause booleanPredicate="False"/>
         </SimpleClause>
      </Clause>
      <Clause>
         <CompoundClause connectivePredicate="Or">
            <Clause>
               <SimpleClause leftArgument="Age">
                  <RationalClause logicalPredicate="EL">
                     <IntClause e-dtype="int">45</IntClause>
                  </RationalClause>
               </SimpleClause>
            </Clause>
            <Clause>
               <SimpleClause leftArgument="American">
                  <BooleanClause booleanPredicate="True"/>
               </SimpleClause>
            </Clause>
         </CompoundClause>
      </Clause>
   </CompoundClause>
</Clause>
```

8.3 SQL Query Support

The Registry may optionally support an SQL based query capability that is designed for Registry clients that demand more advanced query capability. The optional SQLQuery element in the AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query language.

The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of the “SELECT” statement of Entry level SQL defined by ISO/IEC 9075:1992, Database Language SQL [SQL], extended to include `<sql invoked routines>` (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined in template form in Appendix D.3. The syntax of the Registry query language is defined by the BNF grammar in D.1.

Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use relational databases in a Registry implementation.

8.3.1 SQL Query Syntax Binding To [ebRIM]

SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as described in the following sections.

8.3.1.1 Class Binding

A subset of the class names defined in [ebRIM] map to table names that may be queried by an SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an SQL query.

The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3 is as follows:

- Classes that have concrete instances are mapped to relational tables. In addition entity classes (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- The intermediate classes in the inheritance hierarchy, namely RegistryObject and RegistryEntry, map to relational views.

- The names of relational tables and views are the same as the corresponding [ebRIM] class name. However, the name binding is case insensitive.
- Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in Appendix D.3 where the column definitions are based on a subset of attributes defined for that class in [ebRIM]. The attributes that map to columns include the inherited attributes for the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed which column definitions.

An SQLQuery against a table not defined in Appendix D.3 may raise an error condition: InvalidQueryException.

The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn definitions.

8.3.1.2 Primitive Attributes Binding

Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same way as column names in SQL. Again the exact attribute names are defined in the class definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is therefore valid for a query to contain attribute names that do not exactly match the case defined in [ebRIM].

8.3.1.3 Reference Attribute Binding

A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject class returns a reference to an instance of an AccessControlPolicy object.

In such cases the reference maps to the `id` attribute for the referenced object. The name of the resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data type for the column is VARCHAR(64) as defined in Appendix D.3.

When a reference attribute value holds a null reference, it maps to a null value in the SQL binding and may be tested with the <null specification> ("IS [NOT] NULL" syntax) as defined by [SQL].

Reference attribute binding is a special case of a primitive attribute mapping.

8.3.1.4 Complex Attribute Binding

A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type TelephoneNumber, Contact, PersonName etc. in class Organization and class User.

The SQL query schema does not map complex attributes as columns in the table for the class for which the attribute is defined. Instead the complex attributes are mapped to columns in the table for the domain class that represents the data type for the complex attribute (e.g. TelephoneNumber). A column links the row in the domain table to the row in the parent table (e.g. User). An additional column named 'attribute_name' identifies the attribute name in the parent class, in case there are multiple attributes with the same complex attribute type.

This mapping also easily allows for attributes that are a collection of a complex type. For example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier and an attribute_name.

8.3.1.5 Binding of Methods Returning Collections

Several of the [ebRIM] classes define methods in addition to attributes, where these methods return collections of references to instances of classes defined by [ebRIM]. For example, the getPackages method of the ManagedObject class returns a Collection of references to instances of Packages that the object is a member of.

Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in Appendix D.3 such that these stored procedures return a collection of `id` attribute values. The returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

These stored procedures may be used as the right-hand-side of an SQL IN clause to test for membership of an object in such collections of references.

8.3.2 Semantic Constraints On Query Syntax

This section defines simplifying constraints on the query syntax that cannot be expressed in the BNF for the query syntax. These constraints must be applied in the semantic analysis of the query.

1. Class names and attribute names must be processed in a case insensitive manner.
2. The syntax used for stored procedure invocation must be consistent with the syntax of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
3. For this version of the specification, the SQL select column list consists of exactly one column, and must always be `t.id`, where t is a table reference in the FROM clause.
4. Join operations must be restricted to simple joins involving only those columns that have an index defined within the normative SQL schema. This constraint is to prevent queries that may be computationally too expensive.

8.3.3 SQL Query Results

The result of an SQL query resolves to a collection of objects within the registry. It never resolves to partial attributes. The objects related to the result set may be returned as an ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the responseOption parameter specified by the client on the AdHocQueryRequest. The entire result set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 1.1.

8.3.4 Simple Metadata Based Queries

The simplest form of an SQL query is based upon metadata attributes specified for a single class within [ebRIM]. This section gives some examples of simple metadata based queries.

For example, to get the collection of ExtrinsicObjects whose name contains the word 'Acme' and that have a version greater than 1.3, the following query must be submitted:

```
SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
        eo.id = nm.parent AND
        eo.majorVersion >= 1 AND
        (eo.majorVersion >= 2 OR  eo.minorVersion > 3);
```

Note that the query syntax allows for conjugation of simpler predicates into more complex queries as shown in the simple example above.

8.3.5 RegistryObject Queries

The schema for the SQL query defines a special view called RegistryObject that allows doing a polymorphic query against all RegistryObject instances regardless of their actual concrete type or table name.

The following example is the similar to that in Section 8.3.4 except that it is applied against all RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id for all qualifying RegistryObject instances whose name contains the word 'Acme' and whose description contains the word "bicycle".

```
SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND
       d.value LIKE '%bicycle%' AND
       ro.id = nm.parent AND ro.id = d.parent;
```

8.3.6 RegistryEntry Queries

The schema for the SQL query defines a special view called RegistryEntry that allows doing a polymorphic query against all RegistryEntry instances regardless of their actual concrete type or table name.

The following example is the same as Section 8.3.4 except that it is applied against all RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a version greater than 1.3.

```
SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
       re.id = nm.parent AND
       re.majorVersion >= 1 AND
       (re.majorVersion >= 2 OR  re.minorVersion > 3);
```

8.3.7 Classification Queries

This section describes the various classification related queries that must be supported.

8.3.7.1 Identifying ClassificationNodes

Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may also be identified as a path attribute that specifies an XPATH expression [XPT] from a root classification node to the specified classification node in the XML document that would represent the ClassificationNode tree including the said ClassificationNode.

8.3.7.2 Getting ClassificationSchemes

To get the collection of ClassificationSchemes the following query predicate must be supported:

```
SELECT scheme.id FROM ClassificationScheme scheme;
```

The above query returns all ClassificationSchemes. Note that the above query may also specify additional predicates (e.g. name, description etc.) if desired.

8.3.7.3 Getting Children of Specified ClassificationNode

To get the children of a ClassificationNode given the ID of that node the following style of query must be supported:

```
SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

The above query returns all ClassificationNodes that have the node specified by <id> as their parent attribute.

8.3.7.4 Getting Objects Classified By a ClassificationNode

To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the following style of query must be supported:

```
SELECT id FROM ExtrinsicObject
WHERE
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                     WHERE path = '/Geography/Asia/Japan'))
  AND
   id IN (SELECT classifiedObject FROM Classification
          WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                     WHERE path = '/Industry/Automotive'))
```

The above query gets the collection of ExtrinsicObjects that are classified by the Automotive Industry and the Japan Geography. Note that according to the semantics defined for GetClassifiedObjectsRequest, the query will also contain any objects that are classified by descendents of the specified ClassificationNodes.

8.3.7.5 Getting Classifications That Classify an Object

To get the collection of Classifications that classify a specified Object the following style of query must be supported:

```
SELECT id FROM Classification c
       WHERE c.classifiedObject = <id>;
```

8.3.8 Association Queries

This section describes the various Association related queries that must be supported.

8.3.8.1 Getting All Association With Specified Object As Its Source

To get the collection of Associations that have the specified Object as its source, the following query must be supported:

```
SELECT id FROM Association WHERE sourceObject = <id>
```

8.3.8.2 Getting All Association With Specified Object As Its Target

To get the collection of Associations that have the specified Object as its target, the following query must be supported:

```
SELECT id FROM Association WHERE targetObject = <id>
```

8.3.8.3 Getting Associated Objects Based On Association Attributes

To get the collection of Associations that have specified Association attributes, the following queries must be supported:

Select Associations that have the specified name.

```
SELECT id FROM Association WHERE name = <name>
```

Select Associations that have the specified association type, where association type is a string containing the corresponding field name described in [ebRIM].

```
SELECT id FROM Association WHERE
       associationType = <associationType>
```

8.3.8.4 Complex Association Queries

The various forms of Association queries may be combined into complex predicates. The following query selects Associations that have a specific sourceObject, targetObject and associationType:

```
SELECT id FROM Association WHERE
       sourceObject = <id1> AND
       targetObject = <id2> AND
       associationType = <associationType>;
```

8.3.9 Package Queries

To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));
```

8.3.9.1 Complex Package Queries

The following query gets all Packages that a specified object belongs to, that are not deprecated and where name contains "RosettaNet."

```
SELECT id FROM Package p, Name n WHERE
       p.id IN (RegistryObject_packages(<id>)) AND
       nm.value LIKE '%RosettaNet%'  AND nm.parent = p.id AND
       p.status <> ‘Deprecated’
```

8.3.10 ExternalLink Queries

To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is specified:

```
SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
```

To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is specified:

```
SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
```

8.3.10.1 Complex ExternalLink Queries

The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that contain the word ‘legal’ in their description and have a URL for their externalURI.

```
SELECT id FROM ExternalLink WHERE
       id IN (RegistryObject_externalLinks(<id>)) AND
       description LIKE ‘%legal%’ AND
       externalURI LIKE ‘%http://%’
```

8.3.11 Audit Trail Queries

To get the complete collection of AuditableEvent objects for a specified ManagedObject, the following query is specified:

```
SELECT id FROM AuditableEvent WHERE  registryObject = <id>
```

8.4 Content Retrieval

A client retrieves content via the Registry by sending the GetContentRequest to the ObjectQueryManager. The GetContentRequest specifies a list of Object references for Objects that need to be retrieved. The ObjectQueryManager returns the specified content by sending a GetContentResponse message to the ObjectQueryManagerClient class of the client. If there are no errors encountered, the GetContentResponse message includes the specified content as additional payloads within the message. In addition to the GetContentResponse payload, there is one additional payload for each content that was requested. If there are errors encountered, the RegistryResponse payload includes an error and there are no additional content specific payloads.

8.4.1 Identification Of Content Payloads

Since the GetContentResponse message may include several repository items as additional payloads, it is necessary to have a way to identify each payload in the message. To facilitate this identification, the Registry must do the following:

- Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mime-part that contains the corresponding repository item for the ExtrinsicObject
- In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the repository item in the Reference element for that object in the Manifest element of the ebXMLHeader.

8.4.2 GetContentResponse Message Structure

The following message fragment illustrates the structure of the GetContentResponse Message that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs for the requested objects.

```

Content-type: multipart/related; boundary="Boundary"; type="text/xml";

--BoundarY
Content-ID: <GetContentRequest@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:eb= 'http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
<SOAP-ENV:Header>

...ebMS header goes here if using ebMS

</SOAP-ENV:Header>
<SOAP-ENV:Body>

...ebMS manifest gooes here if using ebMS

<?xml version="1.0" encoding="UTF-8"?>

<GetContentRequest>
   <ObjectRefList>
      <ObjectRef id=”d8163dfb-f45a-4798-81d9-88aca29c24ff” …/>
      <ObjectRef id=”212c3a78-1368-45d7-acc9-a935197e1e4f” …/>
   </ObjectRefList>
</GetContentRequest>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--Boundary
Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<CPP>
.....
</CPP>
```

```


--Boundary--
Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f
Content-Type: text/xml

<CPP>
.....
</CPP>

--Boundary—

```

9 Registry Security

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [ebRIM]. Security glossary terms can be referenced from RFC 2828.

9.1 Security Concerns

The security risks broadly stem from the following concerns. After a description of these concerns and potential solutions, we identify the concerns that we address in the current specificiation

1. Is the content of the registry (data) trustworthy?
    a) How to make sure "what is in the registry" is "what is put there" by a submitting organization? This concern can be addressed by ensuring that the publisher is authenticated using digital signature (Source Integrity), message is not corrupted during transfer using digital signature (Data Integrity), and the data is not altered by unauthorized subjects based on access control policy (Authorization)
    b) How to protect data while in transmission?
    Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data Confidentiality that can be addressed by encrypting the data in transmission. How to protect against a replay attack.
    c) Is the content up to date? The versioning as well as any time stamp processing, when done securely will ensure the "latest content" is guaranteed to be the latest content. Authorization with access control policy could solve this problem.
    d) How to ensure only bona fide responsible organizations add contents to registry? Ensuring Source Integrity (as in 1a).
    e) How to ensure that bona fide publishers add contents to registry only at authorized locations? (System Integrity
    f) What if the publishers deny modifying certain content after-the-fact? To prevent this (Nonrepudiation) audit trails may be kept which contain signed message digests.
    g) What if the reader denies getting information from the registry?
2. How to provide selective access to registry content? The broad answer is, by using an access control policy – applies to (a), (b), and (c) directly.
    a) How does a submitting organization restrict access to the content to only specific registry readers?
    b) How can a submitting organization allow some "partners" (fellow publishers) to modify content?
    c) How to provide selective access to partners the registry usage data?
    d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw failure of the registry security components? The solution to this problem is by having System Integrity.
    e) Data confidentiality of RegistryObject

3. How do we make "who can see what" policy itself visible to limited parties, even excluding the administrator (self & confidential maintenance of access control policy). By making sure there is an access control policy for accessing the policies themselves.

4. How to transfer credentials? The broad solution is to use credentials assertion (such as being worked on in SAML). Currently, Registry does not support the notion of a session. Therefore, some of these concerns are not releveant to the current specification.
   a) How to transfer credentials (authorization/authentication) to federated registries?
   b) How do aggregators get credentials (authorization/authentication) transferred to them?
   c) How to store credentials through a session?

In the current version of this specification, we address concern 1(d) above. We have used a minimalist approach with respect to the access control concern in item 2 above. Essentially, "any *known* entity (Submitting Organization) can publish content and *anyone* can view published content." The Registry information model has been designed to allow more sophisticated security policies in future versions of this specification.

9.2 Integrity of Registry Content

It is assumed that most business registries do not have the resources to validate the veracity of the content submitted to them. The minimal integrity that the Registry must provide is to ensure that content submitted by a Submitting Organization (SO) is maintained in the Registry without any tampering *within* the Registry. Furthermore, the Registry must make it possible to identify the SO for any Registry content unambiguously.

9.2.1 Message Payload Signature

Integrity of Registry content requires that all submitted content be signed by the Registry client. The signature on the submitted content ensures that:

- The content has not been tampered within the Registry.
- The content's veracity can be ascertained by its association with a specific submitting organization

This section specifies the requirements for generation, packaging and validation of payload signatures. A payload signature is packaged with the payload. Therefore the requirements apply regardless of whether the Registry Client and the Registry Operator communicate over vanilla SOAP with Attachments.

ebXML Messaging Service ([ebMS]) does not specify the generation, validation and packaging of payload signatures. The specification of payload signatures is left upto the application (such as [ebRS]). So the requirements on the payload signatures augment the [ebMS] specification.

Use Case

This Use Case illustrates the use of header and payload signatures.

- RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the content along with the payload signature to the Registry.
- RC2 (Registry Client 2) retrieves RC1's content from the Registry.
- RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves the content, the response from the Registry Operator to RC2 contains the following:
  - Payload containing the content that has been published by RC1.
  - RC1's payload signature (represented by a ds:Signature element) over RC1's published content.

- Either the key for validating RC1’s payload signature in ds:Signature element (using the KeyInfo element as specified in [XMLDSIG]) or RC1’s identity so RC2 can obtain the validation key for signature (e.g. retrieve a certificate containing the public key for RC1).
- A ds:Signature element containing the header signature. Note that the Registry Operator not RC1 generates this signature.

9.2.2 Payload Signature Requirements

9.2.2.1 Payload Signature Packaging Requirements

A payload signature is represented by a ds:Signature element. The payload signature must be packaged with the payload as specified here. This packaging assumes that the payload is always signed.

- The payload and its signature must be enclosed in a MIME multipart message with a Content-Type of multipart/Related.
- The first body part must contain the XML signature as specified in the section “Payload Signature Generation Requirements”.
- The second body part must be the content

The packaging of the payload signature is as follows:

```

MIME-Version: 1.0
Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
Content-Description: ebXML Message

-- MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: http://claiming-it.com/claim061400a.xml

<?xml version='1.0' encoding="utf-8"?>
<SOAP-ENV: Envelope>
  …
 SOAP-ENV: Envelope>

--MIME_boundary
Content-Type: multipart/Related; boundary=PAYLOAD_boundary

--PAYLOAD_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: payload1
<ds:Signature>
  …. Payload signature
</ds: Signature>

--PAYLOAD_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: payload2
<SubmitObjectsRequest>…</SubmitObjectsRequest>
--MIME_boundary

```

9.2.2.2 Payload Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in this section. Note: the "ds" name space reference is to http://www.w3.org/2000/09/xmldsig#

- ds:SignatureMethod must be present. For same reasons as noted in Section "Message Header Requirements", the client must sign using the following Algorithm attribute: http://www.w3.org/2000/09/xmldsig/#dsa-sha1
- The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. . The following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
  http://www.w3.org/TR/2001/REC-xml-c14n-2001315
- One ds:Reference element to reference the payload that needs to be signed must be created.. The ds:Reference element:
  - Must identify the payload to be signed using the URI attribute of the ds:Reference element.
  - Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be support the following digest algorithm:
    http://www.w3.org/2000/09/xmldsig/#sha1
  - Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

The ds:SignedValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element may be present But when present, the ds:KeyInfo field is subject to the requirements stated in the "KeyDistrbution and KeyInfo element" section of this document.

9.2.2.3 Message Payload Signature Validation

The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

9.2.2.4 Payload Signature Example

The following example shows the format of the payload signature:

```

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<ds:SignedInfo>
             <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>
             <ds:CanonicalizationMethod>
                 Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
            </ds:CanonicalizationMethod>
           <ds:Reference URI=#Payload1>
                     <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
                   <ds:DigestValue> ... </ds:DigestValue>
           </ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>

```

9.3 Authentication

The Registry must be able to authenticate the identity of the Principal associated with client requests. *Authentication* is required to identify the ownership of content as well as to identify what "privileges" a Principal can be assigned with respect to the specific objects in the Registry.

The Registry must perform Authentication on a per request basis. From a security point of view, all messages are independent and there is no concept of a session encompassing multiple messages or conversations. Session support may be added as an optimization feature in future versions of this specification.

It is important to note that the registry can only guarantee data integrity and it may be used for Authentication knowing that it is vulnerable to replay types of attacks. The mechanism we describe in this section would fail to correctly authenticate a Submitting Organization in case of replay attacks. True support for authentication requires timestamps or nonce that are signed

9.3.1 Message Header Signature

Message headers are signed to provide data integrity while the message is in transit. Note that the signature within the message header also signs the digests of the payloads.

Header Signature Requirements

Message headers can be signed and are referred to as a header signature. This section specifies the requirements for generation, packaging and validation of a header signature. These requirements apply when the Registry Client and Registry Operator communicate using vanilla SOAP with Attachments. When ebXML MS is used for communication, then the [ebMS] specifies the generation, packaging and validation of XML signatures in the SOAP header. Therefore the header signature requirements do not apply when the ebXML MS is used for communication. However, payload signature generation requirements (specified elsewhere in this document) do apply whether vanilla SOAP with Attachments or ebXML MS is used for communication.

9.3.1.1 Packaging Requirements

A header signature is represented by a ds:Signature element. The ds:Signature element generated must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature element in the SOAP header field is shown below.

```

MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
Content-Description: ebXML Message

-- MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: http://claiming-it.com/claim061400a.xml

<?xml version='1.0' encoding="utf-8"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header>
        <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            …signature over soap envelope
        </ds:Signature>
    </SOAP-ENV: Header>
    <SOAP-ENV: Body>
        …
    </SOAP-ENV: Body>
</SOAP-ENV: Envelope>

```

9.3.1.2 Header Signature Generation Requirements

The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in this section. A ds:Signature element contains:

- ds:SignedInfo
- ds:SignatureValue
- ds:KeyInfo

The ds:SignedInfo element must be generated as follows:

1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute, a client must be capable of signing using only the following Algorithm attribute: http://www.w3.org/2000/09/xmldsig/#dsa-sha1 This algorithm is being chosen because all XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
2. The ds:SignatureMethod elment must contain a ds:CanonicalizationMethod element. The following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:

   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation. This signs the entire ds:Reference element and:
   - Must include the following ds:Transform:
     http://www.w3.org/2000/09/xmldsig#enveloped-signature
     This ensures that the signature (which is embedded in the <SOAP-ENV:Header> element) is not included in the signature calculation.
   - Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) . The URI attribute must be “”.
   - Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support the following digest algorithm: http://www.w3.org/2000/09/xmldsig/#sha1
   - Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

The ds:SignedValue must be generated as specified in [XMLDSIG].

The ds:KeyInfo element may be present But when present, it is subject to the requirements stated in the “KeyDistrbution and KeyInfo element” section of this document.

9.3.1.3 Header Signature Validation Requirements

The ds:Signature element for the ebXML message header must be validated by the recipient as specified by [XMLDSIG].

9.3.1.4 Header Signature Example

The following example shows the format of a header signature:

```

<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>
        <ds:CanonicalizationMethod>
            Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
        </ds:CanonicalizationMethod>
        <ds:Reference URI= “”>
            <ds:Transform>
                http://www.w3.org/2000/09/xmldsig#enveloped-signature
            </ds:Transform>
            <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
            <ds:DigestValue> ... </ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>

```

9.4 Key Distribution and KeyInfo Element

To validate a signature, the recipient of the signature needs the validation key corresponding to the signer's key. The following use cases need to be handled:

- Registry Operator needs the validation key of the Registry Client to validate the signature
- Registry Client needs the validation key of the Registry Operator to validate the Registry's signature.
- Registry Client RC1 needs the validation key of Registry Client (RC2) to validate the content signed by RC1.

[XMLDSIG] provides a ds:*KeyInfo* element that can be used to pass the recipient information for retrieving the validation key. ds:*KeyInfo* is an optional element as specified in [XMLDSIG]. This field together with the procedures outlined in this section is used to securely pass the validation key to a recipient. ds:Keyinfo can be used to pass information such as keys, certificates, names etc. The intended usage of KeyInfo field for ebXML is as follows:

- Pass a X509 Certificate. This recipient extracts the X509 Certificate and the public key from the certificate.

The following assumptions are also made:

1. A Certificate is associated both with the Registry Operator and a Registry Client.
2. A Registry Client registers its certificate with the Registry Operator. The mechanism used for this is not specified here.
3. A Registry Client obtains the Registry Operator's certificate and stores it in its own local key store. The mechanism is not specified here.

9.4.1 Using ds:KeyInfo Field

Two typical usage scenarios are described below, and the contents of the ds:KeyInfo field based on these use cases are deduced from these scenarios.

Scenario 1

1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
4. Registry Operator retrieves the certificate from the KeyInfo field in the header signature It establishes trust in the certificate by comparing it with the certificate that RC must already have registered.
5. Registry Operator validates the header signature using the public key from the certificate.
6. Registry Operator validates the payload signature by repeating steps 4 and 5 using the certificate from the KeyInfo field of the payload signature. This is only required if the Registry Operator wants to ensure that contents have not been modified on the network.

Scenario 2

1. RC1 signs the payload and SOAP envelope using its private key and publishes to the Registry.
2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
3. The certificate of RC2 is passed to the Registry in the KeyInfo field of the payload signature.
4. RC2 retrieves content from the Registry.
5. Registry Operator signs the SOAP envelope using its private key. Registry Operator sends RC1's content and the RC1's signature (signed by RC1).
6. Registry Operator need not send its certificate in the KeyInfo field sinceRC2 is assumed to have obtained the Registry Operator's certificate out of band and installed it in its local key store.
7. RC2 obtains Registry Operator's certificate out of its local key store and verifies the Registry Operator's signature.
8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates the signature on the payload.

Use Case Summary

Based on the above use cases, a Registry Client and Registry must support the following:

- X509Certificate element.
  - This is a child element of X509Data, which in turn is a child element of KeyInfo. This can be used to pass the certificate to the recipient. X509Certificate element contains a base64-encoded certificate.

9.5 Confidentiality

9.5.1 On-the-wire Message Confidentiality

It is suggested but not required that message payloads exchanged between clients and the Registry be encrypted during transmission. Payload encryption must abide by any restrictions set forth in [SEC].

9.5.2 Confidentiality of Registry Content

In the current version of this specification, there are no provisions for confidentiality of Registry content. All content submitted to the Registry may be discovered and read by *any* client. Therefore, the Registry must be able to decrypt any submitted content after it has been received and prior to storing it in its repository. This implies that the Registry and the client have an a priori agreement regarding encryption algorithm, key exchange agreements, etc. This service is not addressed in this specification.

9.6 Authorization

The Registry must provide an authorization mechanism based on the information model defined in [ebRIM]. In this version of the specification the authorization mechanism is based on a default Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of this specification will allow for custom Access Control Policies to be defined by the Submitting Organization. The authorization is going to be applied on a specific set of privileges. A privelege is the ability to carry a specific action.

9.6.1 Actions

Life Cycle Actions

submitObjects
updateObjects
addSlots
removeSlots
approveObjects
deprecateObjects
removeObjects

Read Actions

The various getXXX() methods in QueryManagement Service.

9.7 Access Control

The Registry must create a default AccessControlPolicy object that grants the default permissions to Registry users based upon their assigned role. The following table defines the Permissions granted by the Registry to the various pre-defined roles for Registry users based upon the default AccessControlPolicy.

Table 11: Default Access Control Policies

Role	Permissions
ContentOwner	Access to *all* methods on Registry Objects that are owned by the ContentOwner.
RegistryAdministrator	Access to *all* methods on *all* Registry Objects
RegistryGuest	Access to *all* read-only (getXXX) methods on *all* Registry Objects (read-only access to all content).

The following list summarizes the default role-based AccessControlPolicy:

- The Registry must implement the default AccessControlPolicy and associate it with all Objects in the Registry
- Anyone can publish content, but needs to be authenticated
- Anyone can access the content without requiring authentication
- The ContentOwner has access to all methods for Registry Objects owned by them
- The RegistryAdministrator has access to all methods on all Registry Objects
- Unauthenticated clients can access all read-only (getXXX) methods
- At the time of content submission, the Registry must assign the default ContentOwner role to the Submitting Organization (SO) as authenticated by the credentials in the submission message. In the current version of this specification, the Submitting Organization will be the DN as identified by the certificate
- Clients that browse the Registry need not use certificates. The Registry must assign the default RegistryGuest role to such clients.

Appendix A Web Service Architecture

A.1 Registry Service Abstract Specification

```
<?xml version = "1.0" encoding = "UTF-8"?>
<definitions name = "RegistryService"
     targetNamespace = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
     xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
     xmlns:xsd1 = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
     xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
     xmlns = "http://schemas.xmlsoap.org/wsdl/">

     <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
"http://schemas.xmlsoap.org/wsdl/ file:///c:/jsews/ebxmlrr-spec/misc/schema/wsdl.xsd"-->
     <documentation>$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/services/Registry.wsdl,v 1.9
2001/11/13 00:51:07 farrukh_najmi Exp $
This is the the normative abstract WSDL service definition for the OASIS ebXML Registry
services.</documentation>

     <import namespace = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0" location =
"http://www.oasis-open.org/ebxml/registry/2.0/schemas/Registry.xsd"/>
     <!-- Commonly re-used Messages -->

     <message name = "RegistryResponse">
            <part name = "RegistryResponse" element = "xsd1:RegistryResponse"/>
            <!--This part is optional and contains the mime/multippart containing content for a
GetContentRequest-->

            <part name = "content" type = "xsd:any"/>
     </message>
     <!-- Messages used by QueryManager -->

     <message name = "GetContentsRequest">
            <part name = "GetContentsRequest" element = "xsd1:GetContentsRequest"/>
     </message>

     <message name = "SubmitAdhocQueryRequest">
            <part name = "SubmitAdhocQueryRequest" element = "xsd1:SubmitAdhocQueryRequest"/>
     </message>

     <!-- Messages used by LifeCycleManager -->
     <message name = "AddSlotsRequest">
            <part name = "AddSlotsRequest" element = "xsd1:AddSlotsRequest"/>
     </message>
     <message name = "ApproveObjectsRequest">
            <part name = "ApproveObjectsRequest" element = "xsd1:ApproveObjectsRequest"/>
     </message>
     <message name = "DeprecateObjectsRequest">
            <part name = "DeprecateObjectsRequest" element = "xsd1:DeprecateObjectsRequest"/>
     </message>
     <message name = "RemoveObjectsRequest">
            <part name = "RemoveObjectsRequest" element = "xsd1:RemoveObjectsRequest"/>
     </message>
     <message name = "RemoveSlotsRequest">
            <part name = "RemoveSlotsRequest" element = "xsd1:RemoveSlotsRequest"/>
     </message>
     <message name = "SubmitObjectsRequest">
            <part name = "SubmitObjectsRequest" element = "xsd1:SubmitObjectsRequest"/>
            <!--This part is the mime/multippart containing content-->
            <part name = "content" type = "xsd:any"/>
     </message>

     <portType name = "QueryManagerPortType">
            <documentation>Maps to the QueryManager interface of Registry Services
spec.</documentation>

            <operation name = "getContents">
                   <input message = "tns:GetContentsRequest"/>
                   <output message = "tns:RegistryResponse"/>
            </operation>

            <operation name = "submitAdhocQuery">
                    <input message = "tns:SubmitAdhocQueryRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
     </portType>

     <portType name = "LifeCycleManagerPortType">
            <documentation>Maps to the LifeCycleManager interface of Registry Services
spec.</documentation>
            <operation name = "addSlots">
                    <input message = "tns:AddSlotsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
            <operation name = "approveObjectsRequest">
                    <input message = "tns:ApproveObjectsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
            <operation name = "deprecateObjectsRequest">
                    <input message = "tns:DeprecateObjectsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
            <operation name = "removeObjectsRequest">
                    <input message = "tns:RemoveObjectsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
            <operation name = "removeSlotsRequest">
                    <input message = "tns:RemoveSlotsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
            <operation name = "submitObjectsRequest">
                    <input message = "tns:SubmitObjectsRequest"/>
                    <output message = "tns:RegistryResponse"/>
            </operation>
     </portType>
</definitions>
```

A.2 Registry Service SOAP Binding

```
<?xml version = "1.0" encoding = "UTF-8"?>
<definitions name = "RegistryServiceSOAPBinding"
     targetNamespace = "urn:oasis:names:tc:ebxml-regrep:soapbinding:wsdl:2.0"
     xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:soapbinding:wsdl:2.0"
     xmlns:registry = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
     xmlns:soap = "http://schemas.xmlsoap.org/wsdl/soap/"
     xmlns:wsdl = "http://schemas.xmlsoap.org/wsdl/"
     xmlns:mime = "http://schemas.xmlsoap.org/wsdl/mime/"
     xmlns = "http://schemas.xmlsoap.org/wsdl/">




     <!--xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance" xsi:schemaLocation =
"http://schemas.xmlsoap.org/wsdl/ file:///C:/jsews/ebxmlrr-spec/misc/schema/wsdl.xsd"-->
     <documentation>$Header: /cvsroot/ebxmlrr/ebxmlrr-
spec/misc/services/RegistrySOAPBinding.wsdl,v 1.11 2001/11/20 12:29:17 farrukh_najmi Exp $
This is the the normative concrete SOAP/HTTP binding for the OASIS ebXML Registry
services.</documentation>

     <!--Import the definition of the abstract OASIS ebXML Registry services-->
     <import namespace = "urn:oasis:names:tc:ebxml-regrep:services:wsdl:2.0"
            location = "http://www.oasis-open.org/ebxml/registry/1.1/services/Registry.wsdl"/>

     <!--The SOAP bindings to the abstract services follow-->

     <binding name = "QueryManagerSoapBinding" type = "tns:QueryManagerPortType">
            <!--
            transport attribute below specifies use of http transport for SOAP binding.
            Currently this is the only normative definition of transport for SOAP binding.
             -->
            <soap:binding style = "document" transport =
"http://schemas.xmlsoap.org/soap/http"/>

            <operation name = "getContent">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:ObjectQueryManager:getContent"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>

                    <output>
                            <mime:multipartRelated>
                                    <mime:part>
                                            <soap:body parts = "RegistryResponse" use =
"literal"/>
                                    </mime:part>
                                    <mime:part>
                                            <mime:content part = "content" type = "*/*"/>
                                    </mime:part>
                            </mime:multipartRelated>
                    </output>
            </operation>

            <operation name = "submitAdhocQuery">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:QueryManager:submitAdhocQueries"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>
    </binding>

    <binding name = "LifeCycleManagerSoapBinding" type = "tns:LifeCycleManagerPortType">
            <soap:binding style = "document" transport =
"http://schemas.xmlsoap.org/soap/http"/>

            <operation name = "addSlots">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:LifeCycleManager:addSlots"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

            <operation name = "approveObjects">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:LifeCycleManager:approveObjects"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

            <operation name = "deprecateObjects">
                    <!--Need undeprecateObjects??-->

                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:LifeCycleManager:deprecateObjects"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

            <operation name = "removeObjects">
```

```
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:QueryManager:removeObjects"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

            <operation name = "removeSlots">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:QueryManager:removeSlots"/>
                    <input>
                            <soap:body use = "literal"/>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

            <operation name = "submitObjects">
                    <soap:operation soapAction =
"uri:oasis:ebxml:registry:services:QueryManager:submitObjects"/>
                    <input>
                            <mime:multipartRelated>
                                    <mime:part>
                                            <soap:body parts = "SubmitObjectsRequest" use =
"literal"/>
                                    </mime:part>
                                    <mime:part>
                                            <mime:content part = "content" type = "*/*"/>
                                    </mime:part>
                            </mime:multipartRelated>
                    </input>
                    <output>
                            <soap:body use = "literal"/>
                    </output>
            </operation>

     </binding>
     <!--The concrete services bound to the SOAP bidning follows-->

     <service name = "RegistryService">
            <documentation>The QueryManager service of OASIS ebXML registry version
1.1</documentation>
            <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
                    <soap:address location = "http://your_URL_to_your_QueryManager"/>
            </port>
            <port name = "LifeCycleManagerSOAPBinding" binding =
"tns:LifeCycleManagerSOAPBinding">
                    <soap:address location = "http://your_URL_to_your_QueryManager"/>
            </port>
     </service>
</definitions>
```

Appendix B ebXML Registry Schema Definitions

B.1 RIM Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
   xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
  >
  <annotation>
    <documentation xml:lang = "en">The schema for OASIS ebXML Registry Information
Model</documentation>
  </annotation>
  <!--$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/schema/rim.xsd,v 1.18 2001/11/24 01:45:44
farrukh_najmi Exp $-->



  <!--Begin information model mapping from ebRIM.-->

  <complexType name = "AssociationType1">
    <annotation>
      <documentation xml:lang = "en">
Association is the mapping of the same named interface in ebRIM.
It extends RegistryObject.

An Association specifies references to two previously submitted
registry entrys.

The sourceObject is id of the sourceObject in association
The targetObject is id of the targetObject in association
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <attribute name = "associationType" use = "required" type = "string"/>
        <attribute name = "sourceObject" use = "required" type = "IDREF"/>
        <attribute name = "targetObject" use = "required" type = "IDREF"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "Association" type = "tns:AssociationType1"/>
  <complexType name = "AuditableEventType">
    <annotation>
      <documentation xml:lang = "en">
Mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <attribute name = "eventType" use = "required">
          <simpleType>
            <restriction base = "NMTOKEN">
```

```
              <enumeration value = "Created"/>
              <enumeration value = "Deleted"/>
              <enumeration value = "Deprecated"/>
              <enumeration value = "Updated"/>
              <enumeration value = "Versioned"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name = "registryObject" use = "required" type = "IDREF"/>
        <attribute name = "timestamp" use = "required" type = "string"/>
        <attribute name = "user" use = "required" type = "IDREF"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "AuditableEvent" type = "tns:AuditableEventType"/>

  <complexType name = "ClassificationType">
    <annotation>
      <documentation xml:lang = "en">
Classification is the mapping of the same named interface in ebRIM.
It extends RegistryObject.

A Classification specifies references to two registry entrys.

The classifiedObject is id of the Object being classified.
The classificationNode is id of the ClassificationNode classying the object
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <attribute name = "classificationScheme" use = "optional" type = "IDREF"/>
        <attribute name = "classifiedObject" use = "required" type = "IDREF"/>
        <attribute name = "classificationNode" use = "required" type = "IDREF"/>
        <attribute name = "nodeRepresentation" use = "optional" type = "string"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "Classification" type = "tns:ClassificationType"/>

  <complexType name = "ClassificationNodeType">
    <annotation>
      <documentation xml:lang = "en">
ClassificationNode is the mapping of the same named interface in ebRIM.
It extends RegistryObject.

ClassificationNode is used to submit a Classification tree to the Registry.

The parent attribute is the id to the parent node. code is an optional code value for a
ClassificationNode
often defined by an external taxonomy (e.g. NAICS)
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">

        <sequence>
          <element ref="tns:ClassificationNode" minOccurs="0" maxOccurs="unbounded" />
        </sequence>
        <attribute name = "parent" type = "IDREF"/>
        <attribute name = "code" type = "string"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ClassificationNode" type = "tns:ClassificationNodeType"/>

  <complexType name = "ClassificationSchemeType">
    <annotation>
      <documentation xml:lang = "en">
ClassificationScheme is the mapping of the same named interface in ebRIM.
It extends RegistryEntry.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryEntryType">
        <sequence>
          <element ref = "tns:ClassificationNode" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
        <attribute name = "isInternal" use = "required" type = "boolean"/>
        <attribute name = "nodeType" use = "required">
          <simpleType>
            <restriction base = "NMTOKEN">
              <enumeration value = "UniqueCode"/>
              <enumeration value = "EmbeddedPath"/>
              <enumeration value = "NonUniqueCode"/>
            </restriction>
          </simpleType>
        </attribute>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ClassificationScheme" type = "tns:ClassificationSchemeType"/>
  <complexType name = "ExternalIdentifierType">
    <annotation>
      <documentation xml:lang = "en">
ExternalIdentifier is the mapping of the same named interface in ebRIM.
It extends RegistryObject.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <attribute name = "identificationScheme" use = "required" type = "IDREF"/>
        <attribute name = "value" use = "required" type = "string"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ExternalIdentifier" type = "tns:ExternalIdentifierType"/>
  <complexType name = "ExternalLinkType">
    <annotation>
      <documentation xml:lang = "en">
ExternalLink is the mapping of the same named interface in ebRIM.
It extends RegistryObject.
      </documentation>
    </annotation>
    <complexContent>
```

```
      <extension base = "tns:RegistryObjectType">
        <attribute name = "externalURI" type = "anyURI"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ExternalLink" type = "tns:ExternalLinkType"/>
  <complexType name = "ExtrinsicObjectType">
    <annotation>
      <documentation xml:lang = "en">
ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.
It inherits RegistryEntryAttributes
      </documentation>
    </annotation>
    <!-- Need clarity on use of contentURI in both directions -->
    <complexContent>
      <extension base = "tns:RegistryEntryType">
        <attribute name = "contentURI" use = "required" type = "anyURI"/>
        <attribute name = "mimeType" type = "string"/>
        <attribute name = "opaque" use = "optional" type = "boolean"/>
      </extension>
    </complexContent>
  </complexType>

  <!--Following element decl nneds to be lower case but using upper camel case for backward
compatibility-->

  <element name = "ExtrinsicObject" type = "tns:ExtrinsicObjectType"/>
  <element name = "Address" type = "tns:PostalAddressType"/>
  <complexType name = "OrganizationType">
    <annotation>
      <documentation xml:lang = "en">
Mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <sequence maxOccurs = "unbounded">
          <element ref = "tns:Address"/>
          <element ref = "tns:TelephoneNumber"/>
        </sequence>
        <attribute name = "parent" type = "IDREF"/>
        <attribute name = "primaryContact" use = "required" type = "IDREF"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "Organization" type = "tns:OrganizationType"/>
  <complexType name = "SlotType1">
    <sequence minOccurs = "0">
      <element ref = "tns:ValueList"/>
    </sequence>
    <attribute name = "name" use = "required" type = "string"/>
    <attribute name = "slotType" use = "optional" type = "string"/>
  </complexType>
  <element name = "Slot" type = "tns:SlotType1"/>
  <complexType name = "ValueListType">
    <sequence maxOccurs = "unbounded">
      <element name = "Value" type = "string"/>
    </sequence>
  </complexType>
  <element name = "ValueList" type = "tns:ValueListType"/>
  <complexType name = "PersonNameType">
    <annotation>
      <documentation xml:lang = "en">
Mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <sequence minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "tns:Slot"/>
    </sequence>
    <attribute name = "firstName" use = "required" type = "string"/>
    <attribute name = "middleName" type = "string"/>
    <attribute name = "lastName" use = "required" type = "string"/>
  </complexType>
  <element name = "PersonName" type = "tns:PersonNameType"/>

  <complexType name = "PostalAddressType">
    <annotation>
      <documentation xml:lang = "en">
Mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <sequence minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "tns:Slot"/>
    </sequence>
    <attribute name = "city" use = "optional" type = "string"/>
    <attribute name = "country" use = "optional" type = "string"/>
    <attribute name = "postalCode" use = "optional" type = "string"/>
    <attribute name = "state" use = "optional" type = "string"/>
    <attribute name = "street" use = "optional" type = "string"/>
    <attribute name = "streetNumber" use = "optional" type = "string"/>
  </complexType>
  <element name = "PostalAddress" type = "tns:PostalAddressType"/>

  <complexType name = "RegistryEntryType">
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <attribute name = "expiration" use = "optional" type = "date"/>
        <attribute name = "majorVersion" default = "1" type = "integer"/>
        <attribute name = "minorVersion" default = "0" type = "integer"/>
        <attribute name = "stability" use = "optional">
          <simpleType>
            <restriction base = "NMTOKEN">
              <enumeration value = "Dynamic"/>
              <enumeration value = "DynamicCompatible"/>
              <enumeration value = "Static"/>
            </restriction>
          </simpleType>
        </attribute>

        <attribute name = "status">
          <simpleType>
            <restriction base = "NMTOKEN">
              <enumeration value = "Submitted"/>
              <enumeration value = "Approved"/>
              <enumeration value = "Deprecated"/>
              <enumeration value = "Withdrawn"/>
            </restriction>
          </simpleType>
        </attribute>
        <attribute name = "userVersion" use = "optional" type = "string"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "RegistryEntry" type = "tns:RegistryEntryType"/>

  <complexType name = "InternationalStringType">
    <sequence minOccurs = "0" maxOccurs = "unbounded">
      <element name = "LocalizedString" type = "tns:LocalizedStringType"/>
    </sequence>
  </complexType>
  <element name = "InternationalString" type = "tns:InternationalStringType"/>

  <complexType name = "LocalizedStringType">
    <attribute name = "lang" use = "optional" default = "en-us" form = "qualified" type = "language"/>
    <attribute name = "charset" use = "optional" default = "UTF-8" />
    <attribute name = "value" use = "required" type = "string"/>
  </complexType>

  <complexType name = "RegistryObjectType">
    <annotation>
      <documentation xml:lang = "en">
id may be empty. If specified it may be in urn:uuid format or be in some
arbitrary format. If id is empty registry must generate globally unique id.

If id is provided and in proper UUID syntax (starts with urn:uuid:)
registry will honour it.

If id is provided and is not in proper UUID syntax then it is used for
linkage within document and is ignored by the registry. In this case the
registry generates a UUID for id attribute.

id must not be null when object is being retrieved from the registry.

      </documentation>
    </annotation>
    <sequence minOccurs = "0" maxOccurs = "1">
      <element name = "Name" type = "tns:InternationalStringType" minOccurs = "0"/>
      <element name = "Description" type = "tns:InternationalStringType" minOccurs = "0"/>
      <element ref = "tns:Slot" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:Classification" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:ExternalIdentifier" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "accessControlPolicy" use = "optional" type = "IDREF"/>
    <attribute name = "id" type = "ID"/>
    <attribute name = "objectType" use = "optional" type = "string"/>
  </complexType>
  <element name = "RegistryObject" type = "tns:RegistryObjectType"/>
  <complexType name = "RegistryPackageType">
    <annotation>
      <documentation xml:lang = "en">
RegistryPackage is the mapping of the same named interface in ebRIM.
```

```
It extends RegistryEntry.

A RegistryPackage is a named collection of objects.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryEntryType">
        <sequence>
          <element ref = "tns:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "RegistryPackage" type = "tns:RegistryPackageType"/>
  <complexType name = "ServiceType">
    <complexContent>
      <extension base = "tns:RegistryEntryType">
        <sequence minOccurs = "0" maxOccurs = "unbounded">
          <element ref = "tns:ServiceBinding"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "Service" type = "tns:ServiceType"/>
  <complexType name = "ServiceBindingType">
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <sequence minOccurs = "0" maxOccurs = "unbounded">
          <element ref = "tns:SpecificationLink"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ServiceBinding" type = "tns:ServiceBindingType"/>
  <complexType name = "SpecificationLinkType">
    <complexContent>
      <extension base = "tns:RegistryObjectType"/>
    </complexContent>
  </complexType>
  <element name = "SpecificationLink" type = "tns:SpecificationLinkType"/>

  <!--??Need to fix TelephoneNumbers at the cost of backward compatibility-->

  <complexType name = "TelephoneNumberType">
    <annotation>
      <documentation xml:lang = "en">
TelephoneNumber is the mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <attribute name = "areaCode" use = "required" type = "string"/>
    <attribute name = "contryCode" use = "required" type = "string"/>
    <attribute name = "extension" type = "string"/>
    <attribute name = "number" use = "required" type = "string"/>
    <attribute name = "url" type = "anyURI"/>
  </complexType>
  <element name = "TelephoneNumber" type = "tns:TelephoneNumberType"/>
  <element name = "FaxNumber" type = "tns:TelephoneNumberType"/>
  <element name = "MobileTelephoneNumber" type = "tns:TelephoneNumberType"/>
  <element name = "PagerNumber" type = "tns:TelephoneNumberType"/>
  <complexType name = "TelephoneNumberListType">
    <sequence>
      <element ref = "tns:TelephoneNumber" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
  </complexType>
  <complexType name = "UserType">
    <annotation>
      <documentation xml:lang = "en">
Mapping of the same named interface in ebRIM.
      </documentation>
    </annotation>
    <complexContent>
      <extension base = "tns:RegistryObjectType">
        <sequence maxOccurs = "unbounded">
          <element ref = "tns:Address"/>
          <element ref = "tns:PersonName"/>
          <element ref = "tns:TelephoneNumber"/>
        </sequence>
        <attribute name = "organization" type = "IDREF"/>
        <attribute name = "email" type = "string"/>
        <attribute name = "url" type = "anyURI"/>
      </extension>
    </complexContent>
  </complexType>
  <element name = "User" type = "tns:UserType"/>
  <complexType name = "ObjectRefType">
    <annotation>
      <documentation xml:lang = "en">
Use to reference an Object by its id.
Specifies the id attribute of the object as its id attribute.
id attribute in ObjectAttributes is exactly the same syntax and semantics as
id attribute in RegistryObject.
      </documentation>
    </annotation>
    <attribute name = "id" type = "ID"/>
  </complexType>
  <element name = "ObjectRef" type = "tns:ObjectRefType"/>

  <element name = "ObjectRefList">
    <annotation>
      <documentation xml:lang = "en">A list of ObjectRefs</documentation>
    </annotation>
    <complexType>
      <sequence minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "tns:ObjectRef"/>
      </sequence>
    </complexType>
  </element>


  <complexType name = "LeafRegistryObjectListType">
    <choice minOccurs = "0" maxOccurs = "unbounded">
      <element ref = "tns:ObjectRef"/>
      <element ref = "tns:Association"/>
      <element ref = "tns:AuditableEvent"/>
      <element ref = "tns:Classification"/>
      <element ref = "tns:ClassificationNode"/>
      <element ref = "tns:ClassificationScheme"/>
      <element ref = "tns:ExternalIdentifier"/>
      <element ref = "tns:ExternalLink"/>
      <element ref = "tns:ExtrinsicObject"/>
      <element ref = "tns:Organization"/>
      <element ref = "tns:RegistryPackage"/>
      <element ref = "tns:Service"/>
      <element ref = "tns:ServiceBinding"/>
      <element ref = "tns:SpecificationLink"/>
      <element ref = "tns:User"/>
    </choice>
  </complexType>
  <element name = "LeafRegistryObjectList" type = "tns:LeafRegistryObjectListType"/>

  <complexType name = "RegistryObjectListType">
    <complexContent>
      <extension base = "tns:LeafRegistryObjectListType">
        <choice minOccurs = "0" maxOccurs = "unbounded">
          <element ref = "tns:RegistryEntry"/>
          <element ref = "tns:RegistryObject"/>
        </choice>
      </extension>
    </complexContent>
  </complexType>
  <element name = "RegistryObjectList" type = "tns:RegistryObjectListType"/>


</schema>
```

B.2 Query Schema

```

<?xml version = "1.0" encoding = "UTF-8"?>

<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
   xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0">

  <!-- Import the rim.xsd file with XML schema mappaing from RIM -->
  <import namespace = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0" schemaLocation = "./rim.xsd"/>

  <complexType name = "ResponseOptionType">
    <attribute name = "returnType" default="RegistryObject">
      <simpleType>
        <restriction base = "NMTOKEN">
          <enumeration value = "ObjectRef"/>
          <enumeration value = "RegistryObject"/>
          <enumeration value = "RegistryEntry"/>
          <enumeration value = "LeafClass"/>
          <enumeration value = "LeafClassWithRepositoryItem"/>
        </restriction>
      </simpleType>
    </attribute>

    <attribute name = "returnComposedObjects" type="boolean" default="false"/>

  </complexType>
  <element name = "ResponseOption" type = "tns:ResponseOptionType"/>

  <element name = "AdhocQueryRequest">
    <annotation>
      <documentation xml:lang = "en">
An Ad hoc query request specifies a query string as defined by [RS] in the queryString attribute

      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "tns:ResponseOption" minOccurs="1" maxOccurs="1" />
        <choice minOccurs="1" maxOccurs="1">
          <element ref = "tns:FilterQuery"/>
          <element ref = "tns:SQLQuery"/>
        </choice>
      </sequence>
    </complexType>
  </element>
  <element name = "SQLQuery" type = "string"/>

  <element name = "AdhocQueryResponse">
    <annotation>
      <documentation xml:lang = "en">
The response includes a RegistryObjectList which has zero or more
RegistryObjects that match the query specified in AdhocQueryRequest.

      </documentation>
    </annotation>
    <complexType>
      <choice>
        <element ref = "tns:SQLQueryResult"/>
        <element ref = "tns:FilterQueryResult"/>
      </choice>
    </complexType>
  </element>

  <element name = "SQLQueryResult" type = "rim:RegistryObjectListType"/>

  <element name = "FilterQuery">
    <complexType>
      <choice minOccurs = "1" maxOccurs = "1">
        <element ref = "tns:RegistryObjectQuery"/>
        <element ref = "tns:RegistryEntryQuery"/>
        <element ref = "tns:AuditableEventQuery"/>
        <element ref = "tns:ClassificationNodeQuery"/>
        <element ref = "tns:ClassificationSchemeQuery"/>
        <element ref = "tns:RegistryPackageQuery"/>
        <element ref = "tns:ExtrinsicObjectQuery"/>
        <element ref = "tns:OrganizationQuery"/>
        <element ref = "tns:ServiceQuery"/>
      </choice>
    </complexType>
  </element>

  <complexType name = "RegistryObjectQueryType">
    <sequence>
      <element ref = "tns:RegistryObjectFilter" minOccurs = "0"  maxOccurs="1"/>
      <element name = "NameBranch" type = "tns:InternationalStringBranchType" minOccurs = "0" maxOccurs
= "1"/>
      <element name = "DescriptionBranch" type = "tns:InternationalStringBranchType" minOccurs = "0"
maxOccurs = "1"/>
      <element ref = "tns:SourceAssociationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:TargetAssociationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:ClassificationBranch" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:ExternalIdentifierFilter" minOccurs = "0" maxOccurs = "unbounded"/>
      <element ref = "tns:SlotBranch" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
  </complexType>
  <element name = "RegistryObjectQuery" type = "tns:RegistryObjectQueryType"/>

  <complexType name = "InternationalStringBranchType">
    <sequence>
      <element ref = "tns:LocalizedStringFilter" minOccurs = "0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <complexType name = "RegistryEntryQueryType">
    <complexContent>
      <extension base = "tns:RegistryObjectQueryType">
        <sequence>
          <element ref = "tns:RegistryEntryFilter" minOccurs = "0" maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "RegistryEntryQuery" type = "tns:RegistryEntryQueryType"/>

  <complexType name = "ClassificationSchemeQueryType">
    <complexContent>
      <extension base = "tns:RegistryEntryQueryType">
        <sequence>
          <element ref = "tns:ClassificationSchemeFilter" minOccurs = "0" maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ClassificationSchemeQuery" type = "tns:ClassificationSchemeQueryType"/>

  <complexType name = "AuditableEventQueryType">
    <complexContent>
      <extension base = "tns:RegistryObjectQueryType">
        <sequence>
          <element ref = "tns:AuditableEventFilter" minOccurs = "0"/>
          <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs = "1"/>
          <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs = "1"/>
          <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs = "1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "AuditableEventQuery" type = "tns:AuditableEventQueryType"/>

  <complexType name = "ClassificationNodeQueryType">
    <complexContent>
      <extension base = "tns:RegistryObjectQueryType">
        <sequence>
          <element ref = "tns:ClassificationNodeFilter" minOccurs = "0" maxOccurs="1"/>
          <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
          <element name = "ClassificationNodeParentBranch" type="tns:ClassificationNodeQueryType"
minOccurs = "0" maxOccurs="1"/>
          <element name = "ClassificationNodeChildrenBranch" type="tns:ClassificationNodeQueryType"
minOccurs = "0" maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ClassificationNodeQuery" type = "tns:ClassificationNodeQueryType"/>

  <complexType name = "RegistryPackageQueryType">
    <complexContent>
      <extension base = "tns:RegistryEntryQueryType">
        <sequence>
          <element ref = "tns:RegistryPackageFilter" minOccurs = "0" maxOccurs="1"/>
          <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs = "unbounded"/>
          <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs = "unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "RegistryPackageQuery" type = "tns:RegistryPackageQueryType"/>

  <complexType name = "ExtrinsicObjectQueryType">
    <complexContent>
      <extension base = "tns:RegistryEntryQueryType">
        <sequence>
          <element ref = "tns:ExtrinsicObjectFilter" minOccurs = "0" maxOccurs="1"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ExtrinsicObjectQuery" type = "tns:ExtrinsicObjectQueryType"/>

  <complexType name = "OrganizationQueryType">
    <complexContent>
      <extension base = "tns:RegistryObjectQueryType">
        <sequence>
          <element ref = "tns:OrganizationFilter" minOccurs = "0" maxOccurs="1"/>
          <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
          <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="unbounded"/>
          <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs = "1"/>
          <element name = "OrganizationParentQuery" type="tns:OrganizationQueryType" minOccurs = "0"
maxOccurs="1" />
          <element name = "OrganizationChildrenQuery" type="tns:OrganizationQueryType" minOccurs = "0"
maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "OrganizationQuery" type = "tns:OrganizationQueryType"/>

  <complexType name = "ServiceQueryType">
    <complexContent>
      <extension base = "tns:RegistryEntryQueryType">
        <sequence>
          <element ref = "tns:ServiceFilter" minOccurs = "0" maxOccurs="1"/>

          <element ref = "tns:ServiceBindingBranch" minOccurs = "0" maxOccurs="unbounded"/>

        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name = "ServiceQuery" type = "tns:ServiceQueryType"/>

  <element name = "ServiceBindingBranch">
    <complexType>
      <sequence>
        <element ref = "tns:ServiceBindingFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:SpecificationLinkBranch" minOccurs = "0" maxOccurs="unbounded"/>

      </sequence>
    </complexType>
  </element>

  <element name = "SpecificationLinkBranch">
    <complexType>
      <sequence>
        <element ref = "tns:SpecificationLinkFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
      </sequence>
    </complexType>
  </element>

  <element name = "FilterQueryResult" >
    <complexType>
      <choice minOccurs = "1" maxOccurs = "1">
        <element ref = "tns:RegistryObjectQueryResult"/>
        <element ref = "tns:RegistryEntryQueryResult"/>
        <element ref = "tns:AuditableEventQueryResult"/>
        <element ref = "tns:ClassificationNodeQueryResult"/>
        <element ref = "tns:ClassificationSchemeQueryResult"/>
        <element ref = "tns:RegistryPackageQueryResult"/>
        <element ref = "tns:ExtrinsicObjectQueryResult"/>
        <element ref = "tns:OrganizationQueryResult"/>
        <element ref = "tns:ServiceQueryResult"/>
      </choice>
    </complexType>
  </element>

  <element name = "RegistryObjectQueryResult" type = "rim:RegistryObjectListType"/>

  <element name = "RegistryEntryQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:ClassificationScheme"/>
        <element ref = "rim:ExtrinsicObject"/>
        <element ref = "rim:RegistryEntry"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:RegistryPackage"/>
      </choice>
    </complexType>
  </element>

  <element name = "AuditableEventQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:AuditableEvent"/>
      </choice>
    </complexType>
  </element>

  <element name = "ClassificationNodeQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:ClassificationNode"/>
      </choice>
    </complexType>
  </element>

  <element name = "ClassificationSchemeQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:RegistryEntry"/>
        <element ref = "rim:ClassificationScheme"/>
      </choice>
    </complexType>
  </element>

  <element name = "RegistryPackageQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryEntry"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:RegistryPackage"/>
      </choice>
    </complexType>
  </element>

  <element name = "ExtrinsicObjectQueryResult">
    <complexType>
```

```
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:RegistryEntry"/>
        <element ref = "rim:ExtrinsicObject"/>
      </choice>
    </complexType>
  </element>

  <element name = "OrganizationQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:Organization"/>
      </choice>
    </complexType>
  </element>

  <element name = "ServiceQueryResult">
    <complexType>
      <choice minOccurs = "0" maxOccurs = "unbounded">
        <element ref = "rim:ObjectRef"/>
        <element ref = "rim:RegistryObject"/>
        <element ref = "rim:RegistryEntry"/>
        <element ref = "rim:Service"/>
      </choice>
    </complexType>
  </element>

  <complexType name = "AssociationBranchType">
    <sequence>
      <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs="1"/>
      <choice minOccurs = "0" maxOccurs="1">
        <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:AuditableEventQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryPackageQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ExtrinsicObjectQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ServiceQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ExternalLinkFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ExternalIdentifierFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:UserBranch" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:SourceAssociationBranch" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:TargetAssociationBranch" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassifiedByBranch" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ServiceBindingBranch" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:SpecificationLinkBranch" minOccurs = "0" maxOccurs="1"/>
      </choice>
    </sequence>
  </complexType>
  <element name = "SourceAssociationBranch" type="tns:AssociationBranchType" />
  <element name = "TargetAssociationBranch" type="tns:AssociationBranchType" />

  <element name = "ClassificationBranch">
    <complexType>
      <sequence>
        <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
      </sequence>
    </complexType>
  </element>

  <element name = "ClassifiedByBranch">
    <complexType>
      <sequence>
        <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
      </sequence>
    </complexType>
  </element>

  <element name = "SlotBranch">
    <complexType>
      <sequence>
        <element ref = "tns:SlotFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:SlotValueFilter" minOccurs = "0" maxOccurs = "unbounded"/>
      </sequence>
    </complexType>
  </element>

  <element name = "OrganizationParentBranch">
    <complexType>
      <sequence>
        <element ref = "tns:OrganizationFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="1"/>
        <element ref = "tns:OrganizationParentBranch" minOccurs = "0"/>
      </sequence>
    </complexType>
  </element>

  <element name = "UserBranch">
          <complexType>
                  <sequence>
                          <element ref = "tns:UserFilter" minOccurs = "0" maxOccurs="1"/>
                          <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
                          <element ref = "tns:TelephoneNumberFilter" minOccurs = "0"
maxOccurs="unbounded"/>
                          <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
                  </sequence>
          </complexType>
  </element>


  <complexType name = "FilterType">
    <sequence>
```

```
      <element ref = "tns:Clause"/>
    </sequence>
  </complexType>


  <element name = "RegistryObjectFilter" type = "tns:FilterType"/>
  <element name = "RegistryEntryFilter" type = "tns:FilterType"/>
  <element name = "ExtrinsicObjectFilter" type = "tns:FilterType"/>
  <element name = "RegistryPackageFilter" type = "tns:FilterType"/>
  <element name = "OrganizationFilter" type = "tns:FilterType"/>
  <element name = "ClassificationNodeFilter" type = "tns:FilterType"/>
  <element name = "AssociationFilter" type = "tns:FilterType"/>
  <element name = "ClassificationFilter" type = "tns:FilterType"/>
  <element name = "ClassificationSchemeFilter" type = "tns:FilterType"/>
  <element name = "ExternalLinkFilter" type = "tns:FilterType"/>
  <element name = "ExternalIdentifierFilter" type = "tns:FilterType"/>
  <element name = "SlotFilter" type = "tns:FilterType"/>
  <element name = "AuditableEventFilter" type = "tns:FilterType"/>
  <element name = "UserFilter" type = "tns:FilterType"/>
  <element name = "SlotValueFilter" type = "tns:FilterType"/>
  <element name = "PostalAddressFilter" type = "tns:FilterType"/>
  <element name = "TelephoneNumberFilter" type = "tns:FilterType"/>
  <element name = "ServiceFilter" type = "tns:FilterType"/>
  <element name = "ServiceBindingFilter" type = "tns:FilterType"/>
  <element name = "SpecificationLinkFilter" type = "tns:FilterType"/>
  <element name = "LocalizedStringFilter" type = "tns:FilterType"/>

  <element name = "Clause">
    <annotation>
      <documentation xml:lang = "en">
The following lines define the XML syntax for Clause.

      </documentation>
    </annotation>
    <complexType>
      <choice>
        <element ref = "tns:SimpleClause"/>
        <element ref = "tns:CompoundClause"/>
      </choice>
    </complexType>
  </element>
  <element name = "SimpleClause">
    <complexType>
      <choice>
        <element ref = "tns:BooleanClause"/>
        <element ref = "tns:RationalClause"/>
        <element ref = "tns:StringClause"/>
      </choice>
      <attribute name = "leftArgument" use = "required" type = "string"/>
    </complexType>
  </element>
  <element name = "CompoundClause">
    <complexType>
      <sequence>
        <element ref = "tns:Clause"/>
        <element ref = "tns:Clause" maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "connectivePredicate" use = "required">
        <simpleType>
          <restriction base = "NMTOKEN">
            <enumeration value = "And"/>
            <enumeration value = "Or"/>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>
  </element>
  <element name = "BooleanClause">
    <complexType>
      <attribute name = "booleanPredicate" use = "required" type = "boolean"/>
    </complexType>
  </element>
  <element name = "RationalClause">
    <complexType>
      <choice>
        <element ref = "tns:IntClause"/>
        <element ref = "tns:FloatClause"/>
      </choice>
      <attribute name = "logicalPredicate" use = "required">
        <simpleType>
          <restriction base = "NMTOKEN">
            <enumeration value = "LE"/>
            <enumeration value = "LT"/>
            <enumeration value = "GE"/>
            <enumeration value = "GT"/>
            <enumeration value = "EQ"/>
            <enumeration value = "NE"/>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>
  </element>
  <element name = "IntClause" type = "integer"/>
  <element name = "FloatClause" type = "float"/>
  <element name = "StringClause">
    <complexType>
      <simpleContent>
        <extension base = "string">
          <attribute name = "stringPredicate" use = "required">
            <simpleType>
              <restriction base = "NMTOKEN">
                <enumeration value = "contains"/>
                <enumeration value = "-contains"/>
                <enumeration value = "startswith"/>
                <enumeration value = "-startswith"/>
                <enumeration value = "equal"/>
                <enumeration value = "-equal"/>
                <enumeration value = "endswith"/>
                <enumeration value = "-endswith"/>
              </restriction>
            </simpleType>
          </attribute>
        </extension>
      </simpleContent>
    </complexType>
  </element>

  <element name = "GetContentRequest">
    <annotation>
      <documentation xml:lang = "en">
Gets the actual content (not metadata) specified by the ObjectRefList
      </documentation>
    </annotation>

    <complexType>
      <sequence>
        <element ref = "rim:ObjectRefList"/>
      </sequence>
    </complexType>
  </element>

  <element name = "GetContentResponse">
    <annotation>
      <documentation xml:lang = "en">
The GetObjectsResponse will have no sub-elements if there were no errors.
The actual contents will be in the other payloads of the message.
      </documentation>
    </annotation>
    <complexType/>
  </element>

</schema>
```

B.3 Registry Services Interface Schema

```
<?xml version = "1.0" encoding = "UTF-8"?>
<!--Generated by XML Authority. Conforms to w3c http://www.w3.org/2001/XMLSchema-->
<!--$Header: /cvsroot/ebxmlrr/ebxmlrr-spec/misc/schema/rs.xsd,v 1.17 2001/11/23 18:52:12 farrukh_najmi
Exp $-->

<schema xmlns = "http://www.w3.org/2001/XMLSchema"
   targetNamespace = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
   xmlns:tns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
   xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
   xmlns:query = "urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0"
  >
  <annotation>
    <documentation xml:lang = "en">The schema for OASIS ebXML Registry Services</documentation>
  </annotation>

  <!-- Import the rim.xsd file with XML schema mappaing from RIM -->
  <import namespace="urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0" schemaLocation="./rim.xsd"/>

  <!-- Import the query.xsd file with XML schema for query related schema -->
  <import namespace="urn:oasis:names:tc:ebxml-regrep:query:xsd:2.0" schemaLocation="./query.xsd"/>



  <element name = "RequestAcceptedResponse">
    <annotation>
      <documentation xml:lang = "en">
```

```
Mapping of the same named interface in ebRS.
      </documentation>
    </annotation>
    <complexType/>
  </element>

  <element name = "SubmitObjectsRequest">
    <annotation>
      <documentation xml:lang = "en">
The SubmitObjectsRequest allows one to submit a list of RegistryObject elements. Each RegistryEntry
element provides metadata for a single submitted object.  Note that the repository item being submitted
is in a separate document that is not in this DTD. The ebXML Messaging Services Specfication defines
packaging, for submission, of the metadata of a repository item with the repository item itself. The
value of the contentURI attribute of the ExtrinsicObject element must be the same as the xlink:href
attribute within the Reference element within the Manifest element of the MessageHeader.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "rim:LeafRegistryObjectList"/>
      </sequence>
    </complexType>
  </element>

  <element name = "UpdateObjectsRequest">
    <annotation>
      <documentation xml:lang = "en">
The UpdateObjectsRequest allows one to update a list of RegistryObject elements. Each RegistryEntry
element provides metadata for a single submitted object.  Note that the repository item being submitted
is in a separate document that is not in this DTD. The ebXML Messaging Services Specfication defines
packaging, for submission, of the metadata of a repository item with the repository item itself. The
value of the contentURI attribute of the ExtrinsicObject element must be the same as the xlink:href
attribute within the Reference element within the Manifest element of the MessageHeader.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "rim:LeafRegistryObjectList"/>
      </sequence>
    </complexType>
  </element>

  <element name = "AddSlotsRequest">
    <complexType>
      <sequence>
        <element ref = "rim:ObjectRef" minOccurs="1" maxOccurs="1"/>
        <element ref = "rim:Slot" minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name = "RemoveSlotsRequest">
    <annotation>
      <documentation xml:lang = "en"> Only need name in Slot within SlotList </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "rim:ObjectRef" minOccurs="1" maxOccurs="1"/>
        <element ref = "rim:Slot" minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <element name = "ApproveObjectsRequest">
    <annotation>
      <documentation xml:lang = "en">
The ObjectRefList is the list of
refs to the registry entrys being approved.
      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "rim:ObjectRefList"/>
      </sequence>
    </complexType>
  </element>
  <element name = "DeprecateObjectsRequest">
    <annotation>
      <documentation xml:lang = "en">
The ObjectRefList is the list of
refs to the registry entrys being deprecated.

      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "rim:ObjectRefList"/>
      </sequence>
    </complexType>
  </element>
  <element name = "RemoveObjectsRequest">
    <annotation>
      <documentation xml:lang = "en">
The ObjectRefList is the list of
refs to the registry entrys being removed

      </documentation>
    </annotation>

    <complexType>
      <sequence>
        <element ref = "rim:ObjectRefList"/>
      </sequence>
      <attribute name = "deletionScope" use = "optional">
        <simpleType>
          <restriction base = "NMTOKEN">
            <enumeration value = "DeleteAll"/>
            <enumeration value = "DeleteRepositoryItemOnly"/>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>
  </element>



  <element name = "RegistryProfile">
    <annotation>
```

```
      <documentation xml:lang = "en">
Describes the capability profile for the registry and what optional features
are supported

      </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "tns:OptionalFeaturesSupported"/>
      </sequence>
      <attribute name = "version" use = "required" type = "string"/>
    </complexType>
  </element>

  <element name = "OptionalFeaturesSupported">
    <complexType>
      <attribute name = "sqlQuery" default = "false" type = "boolean"/>
      <attribute name = "xQuery" default = "false" type = "boolean"/>
    </complexType>
  </element>

  <simpleType name = "ErrorType">
    <restriction base = "NMTOKEN">
      <enumeration value = "Warning"/>
      <enumeration value = "Error"/>
    </restriction>
  </simpleType>
  <element name = "RegistryErrorList">
    <annotation>
      <documentation xml:lang = "en"> End FilterQuery DTD </documentation>
      <documentation xml:lang = "en"> Begin RegistryError definition </documentation>
      <documentation xml:lang = "en"> The RegistryErrorList is derived from the ErrorList element from
the ebXML Message Service Specification </documentation>
    </annotation>
    <complexType>
      <sequence>
        <element ref = "tns:RegistryError" maxOccurs = "unbounded"/>
      </sequence>
      <attribute name = "highestSeverity" use = "optional" type = "tns:ErrorType"/>
    </complexType>
  </element>
  <element name = "RegistryError">
    <complexType>
      <simpleContent>
        <extension base = "string">
          <attribute name = "codeContext" use = "required" type = "string"/>
          <attribute name = "errorCode" use = "required" type = "string"/>
          <attribute name = "severity" use = "optional" type = "tns:ErrorType"/>
          <attribute name = "location" type = "string"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
  <element name = "RegistryResponse">
    <complexType>
      <sequence>
        <choice minOccurs = "0">
```

```
          <element ref = "query:AdhocQueryResponse"/>
          <element ref = "query:GetContentResponse"/>
        </choice>
        <element ref = "tns:RegistryErrorList" minOccurs = "0"/>
      </sequence>
      <attribute name = "status" use = "required">
        <simpleType>
          <restriction base = "NMTOKEN">
            <enumeration value = "success"/>
            <enumeration value = "warning"/>
            <enumeration value = "failure"/>
            <enumeration value = "unavailable"/>
          </restriction>
        </simpleType>
      </attribute>
    </complexType>
  </element>

  <element name = "RootElement">
    <annotation>
      <documentation xml:lang = "en"> The contrived root node </documentation>
    </annotation>
    <complexType>
      <choice>
        <element ref = "tns:SubmitObjectsRequest"/>
        <element ref = "tns:UpdateObjectsRequest"/>
        <element ref = "tns:ApproveObjectsRequest"/>
        <element ref = "tns:DeprecateObjectsRequest"/>
        <element ref = "tns:RemoveObjectsRequest"/>
        <element ref = "query:AdhocQueryRequest"/>
        <element ref = "query:GetContentRequest"/>
        <element ref = "tns:AddSlotsRequest"/>
        <element ref = "tns:RemoveSlotsRequest"/>
        <element ref = "tns:RegistryResponse"/>
        <element ref = "tns:RegistryProfile"/>
      </choice>
    </complexType>
  </element>
</schema>
```

Appendix C Interpretation of UML Diagrams

This section describes in *abstract terms* the conventions used to define ebXML business process description in UML.

C.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP]) required to implement an ebXML Registry Services and clients. See on page **Error! Bookmark not defined.** for an example. The UML class diagram contains:

1. A collection of UML interfaces where each interface represents a Service Interface for a Registry service.
2. Tabular description of methods on each interface where each method represents an Action (as defined by [ebCPP]) within the Service Interface representing the UML interface.
3. Each method within a UML interface specifies one or more parameters, where the type of each method argument represents the ebXML message type that is exchanged as part of the Action corresponding to the method. Multiple arguments imply multiple payload documents within the body of the corresponding ebXML message.

C.2 UML Sequence Diagram

A UML sequence diagram is used to specify the business protocol representing the interactions between the UML interfaces for a Registry specific ebXML business process. A UML sequence diagram provides the necessary information to determine the sequencing of messages, request to response association as well as request to error response association as described by [ebCPP].

Each sequence diagram shows the sequence for a specific conversation protocol as method calls from the requestor to the responder. Method invocation may be synchronous or asynchronous based on the UML notation used on the arrow-head for the link. A half arrow-head represents asynchronous communication. A full arrow-head represents synchronous communication.

Each method invocation may be followed by a response method invocation from the responder to the requestor to indicate the ResponseName for the previous Request. Possible error response is indicated by a conditional response method invocation from the responder to the requestor. See Figure 8 on page 26 for an example.

Appendix D SQL Query

D.1 SQL Query Syntax Specification

This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax conforms to the <query specification>, modulo the restrictions identified below:

1. A **<select list>** may contain at most one **<select sublist>.**
2. In a **<select list>** must be is a single column whose data type is UUID, from the table in the **<from clause>.**
3. A **<derived column>** may not have an **<as clause>**.
4. **<table expression>** does not contain the optional **<group by clause>** and **<having clause>** clauses.
5. A **<table reference>** can only consist of **<table name>** and **<correlation name>.**
6. A **<table reference>** does not have the optional AS between **<table name>** and **<correlation name>.**
7. There can only be one **<table reference>** in the **<from clause>**.
8. Restricted use of sub-queries is allowed by the syntax as follows. The **<in predicate>** allows for the right hand side of the **<in predicate>** to be limited to a restricted **<query specification>** as defined above.
9. A **<search condition>** within the **<where clause>** may not include a **<query expression>**.
10. Simple joins are allowed only if they are based on indexed columns within the relational schema.
11. The SQL query syntax allows for the use of **<sql invoked routines>** invocation from [SQL/PSM] as the RHS of the **<in predicate>**.

D.2 Non-Normative BNF for Query Syntax Grammar

The following BNF exemplifies the grammar for the registry query syntax. It is provided here as an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative syntax. For the normative syntax rules see Appendix D.1.

```
/*******************************************************************
 * The Registry Query (Subset of SQL-92) grammar starts here
 *******************************************************************/

RegistryQuery = SQLSelect [";"]

SQLSelect = "SELECT" ["DISTINCT"] SQLSelectCols "FROM" SQLTableList [ SQLWhere ]

SQLSelectCols =  ID

SQLTableList = SQLTableRef

SQLTableRef = ID

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*
```

```
SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*

SQLNotExpr = [ "NOT" ] SQLCompareExpr

SQLCompareExpr =
    (SQLColRef "IS") SQLIsClause
  | SQLSumExpr [ SQLCompareExprRight ]


SQLCompareExprRight =
    SQLLikeClause
  | SQLInClause
  | SQLCompareOp SQLSumExpr

SQLCompareOp =
    "="
  | "<>"
  | ">"
  | ">="
  | "<"
  | "<="

SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"

SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*

SQLLValueElement = "NULL" | SQLSelect

SQLIsClause =  SQLColRef "IS" [ "NOT" ] "NULL"

SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern

SQLPattern = STRING_LITERAL

SQLLiteral =
    STRING_LITERAL
  | INTEGER_LITERAL
  | FLOATING_POINT_LITERAL

SQLColRef = SQLLvalue

SQLLvalue =  SQLLvalueTerm

SQLLvalueTerm = ID ( "."  ID )*

SQLSumExpr = SQLProductExpr (( "+" | "-" ) SQLProductExpr )*

SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*

SQLUnaryExpr = [ ( "+" | "-") ] SQLTerm

SQLTerm = "(" SQLOrExpr ")"
  | SQLColRef
  | SQLLiteral

INTEGER_LITERAL = (["0"-"9"])+

FLOATING_POINT_LITERAL =
          (["0"-"9"])+ "." (["0"-"9"])+ (EXPONENT)?
        | "." (["0"-"9"])+ (EXPONENT)?
        | (["0"-"9"])+ EXPONENT
        | (["0"-"9"])+ (EXPONENT)?

EXPONENT = ["e","E"] (["+","-"])? (["0"-"9"])+

STRING_LITERAL: "'" (~["'"])* ( "''" (~["'"])* )* "'"

ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
LETTER = ["A"-"Z", "a"-"z"]
DIGIT = ["0"-"9"]
```

D.3 Relational Schema For SQL Queries

```
CREATE TABLE Association (

--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                          VARCHAR(64) NOT NULL PRIMARY KEY,
--??Should be String32 in RIM not LongName
  objectType           VARCHAR(32) DEFAULT 'Association' CHECK (objectType = 'Association'),

--Association attributes
  associationType      VARCHAR(128) NOT NULL,
  sourceObject         VARCHAR(64) NOT NULL,
  targetObject         VARCHAR(64) NOT NULL
);

CREATE TABLE AuditableEvent (

--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                          VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'AuditableEvent' CHECK (objectType =
'AuditableEvent'),

--AuditableEvent attributes
  eventType                   VARCHAR(128) NOT NULL,

--??What about keyword conflicts. Adding '_' suffix for now
  timeStamp_           TIMESTAMP NOT NULL,
  user_                       VARCHAR(64) NOT NULL
);

CREATE TABLE Classification (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                          VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'Classification' CHECK (objectType =
'Classification'),

--Classification attributes.
  classificationNode          VARCHAR(64),
  classificationScheme        VARCHAR(64),
  classifiedObject                   VARCHAR(64) NOT NULL,

  --??What is the right String length for nodeRepresentation
  nodeRepresentation          VARCHAR(256)

);


CREATE TABLE ClassificationNode (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                          VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'ClassificationNode' CHECK (objectType =
'ClassificationNode'),

--ClassificationNode attributes
  code                               VARCHAR(64),

  --??Need to allow for null parent in RIM. Currently this is not so.
  parent                             VARCHAR(64)
);

CREATE TABLE ClassificationScheme (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                          VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'ClassificationScheme' CHECK (objectType =
'ClassificationScheme'),

--RegistryEntry attributes
  expirationDate       TIMESTAMP,
  majorVersion         INT DEFAULT 0 NOT NULL,
  minorVersion         INT DEFAULT 1 NOT NULL,

  --??stability and status in RIM1.1 are LongName. Should be ShortName
  stability                    VARCHAR(128),
  status                       VARCHAR(128) NOT NULL,
  userVersion          VARCHAR(64),

--ClassificationScheme attributes: currently none defined
  isInternal           BOOLEAN DEFAULT false NOT NULL,
  nodeType                     VARCHAR(32) DEFAULT 'UniqueCode' NOT NULL
);

CREATE TABLE ExternalIdentifier (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                           VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'ExternalIdentifier' CHECK (objectType =
'ExternalIdentifier'),

--ExternalIdentifier attributes
  registryObject       VARCHAR(64) NOT NULL,
  identificationScheme         VARCHAR(64) NOT NULL,
  value                        VARCHAR(64) NOT NULL
);

CREATE TABLE ExternalLink (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                           VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'ExternalLink' CHECK (objectType = 'ExternalLink'),

--ExternalLink attributes
  externalURI          VARCHAR(256) NOT NULL
);

CREATE TABLE ExtrinsicObject (

--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                           VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'text/xml',

--RegistryEntry attributes
  expirationDate       TIMESTAMP,
  majorVersion         INT DEFAULT 0 NOT NULL,
  minorVersion         INT DEFAULT 1 NOT NULL,

  --??stability and status in RIM1.1 are LongName. Should be ShortName
  stability                    VARCHAR(128),
  status                       VARCHAR(128) NOT NULL,
  userVersion          VARCHAR(64),

--ExtrinsicObject attributes
  contentURI           VARCHAR(256),
  isOpaque                     BOOLEAN DEFAULT false NOT NULL,
  mimeType                     VARCHAR(128) NOT NULL
);

CREATE TABLE Name (

--LocalizedString attributes flattened for Name
  charset                      VARCHAR(32),
  lang                         VARCHAR(32) NOT NULL,
  value                        VARCHAR(128) NOT NULL,

--The RegistryObject id for teh parent RegistryObject for which this is a Name
  parent                       VARCHAR(64) NOT NULL,

  PRIMARY KEY (parent, lang, value)

);

CREATE TABLE Description (

--LocalizedString attributes flattened for Description
  charset                      VARCHAR(32),
  lang                         VARCHAR(32) NOT NULL,
  value                        VARCHAR(128) NOT NULL,

--The RegistryObject id for teh parent RegistryObject for which this is a Name
  parent                        VARCHAR(64) NOT NULL,

  PRIMARY KEY (parent, lang, value)

);

CREATE TABLE Organization (

--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'Organization' CHECK (objectType = 'Organization'),

--Organization attributes

--Organization.address attribute is in PostalAddress table

  parent                        VARCHAR(64),

--primary contact for Organization, points to a User.
  primaryContact       VARCHAR(64) NOT NULL

--Organization.telephoneNumbers attribute is in TelephoneNumber table
);

CREATE TABLE RegistryPackage (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'RegistryPackage' CHECK (objectType =
'RegistryPackage'),

--RegistryEntry attributes
  expirationDate       TIMESTAMP,
  majorVersion         INT DEFAULT 0 NOT NULL,
  minorVersion         INT DEFAULT 1 NOT NULL,

  --??stability and status in RIM1.1 are LongName. Should be ShortName
  stability                     VARCHAR(128),
  status                        VARCHAR(128) NOT NULL,
  userVersion          VARCHAR(64)

--RegistryPackage attributes: currently none defined
);

CREATE TABLE PostalAddress (

  city                          VARCHAR(64),
  country                       VARCHAR(64),
  postalCode           VARCHAR(64),
  state                         VARCHAR(64),
  street                        VARCHAR(64),
  streetNumber         INTEGER,

--The parent object that this is an address for
  parent                        VARCHAR(64) PRIMARY KEY NOT NULL

);

CREATE TABLE Service (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'Service' CHECK (objectType = 'Service')

--Service attributes: currently none defined
);

CREATE TABLE ServiceBinding (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'ServiceBinding' CHECK (objectType = 
'ServiceBinding'),

--ServiceBinding attributes
  service                       VARCHAR(64) NOT NULL
);

--Multiple rows of Slot make up a single Slot
CREATE TABLE Slot (

  name                          VARCHAR(128) NOT NULL,

--??Do we need this or should we get rid of it from RIM
  slotType                      VARCHAR(128),
  value                         VARCHAR(64),

--The parent RegistryObject that this is a Slot for
  parent                        VARCHAR(64) NOT NULL,

  PRIMARY KEY (parent, name)
);


CREATE TABLE SpecificationLink (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'SpecificationLink' CHECK (objectType = 
'SpecificationLink'),

--SpecificationLink attributes
--??Need to tell Sally to add these two attribute to RIM
  service                       VARCHAR(64) NOT NULL,
  serviceBinding       VARCHAR(64) NOT NULL
);

CREATE TABLE TelephoneNumber (
  areaCode                      VARCHAR(4),
  countryCode          VARCHAR(4),
  extension                     VARCHAR(8),
  number                        VARCHAR(8),
  phoneType                     VARCHAR(128),

--??Remove from RIM and here
  url                           VARCHAR(256),

  parent                        VARCHAR(64) NOT NULL PRIMARY KEY
);

CREATE TABLE User_ (
--Object Attributes
  accessControlPolicy  VARCHAR(64),
  id                            VARCHAR(64) NOT NULL PRIMARY KEY,
  objectType           VARCHAR(32) DEFAULT 'User' CHECK (objectType = 'User'),

--User attributes

--address is in PostalAddress table

  email                         VARCHAR(128) NOT NULL,
  organization         VARCHAR(64) NOT NULL,

--personName flattened
  personName_firstName VARCHAR(64),
  personName_middleName       VARCHAR(64),
  personName_lastName  VARCHAR(64),


--telephoneNumbers is in TelephoneNumber table

  url                           VARCHAR(256)

);

CREATE VIEW RegistryObject (
--Object Attributes
  accessControlPolicy,
  id,
  objectType

) AS
 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM Association
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM AuditableEvent
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM Classification
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ClassificationNode
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ClassificationScheme
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ExternalIdentifier
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ExternalLink
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ExtrinsicObject
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM Organization
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM RegistryPackage
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM Service
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM ServiceBinding
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM SpecificationLink
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType

 FROM User_
;

CREATE VIEW RegistryEntry (
--Object Attributes
  accessControlPolicy,
  id,
  objectType,

--RegistryEntry attributes
  expirationDate,
  majorVersion,
  minorVersion,
  stability,
  status,
  userVersion

) AS
 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType,

--RegistryEntry attributes
  expirationDate,
  majorVersion,
  minorVersion,
  stability,
  status,
  userVersion

 FROM ClassificationScheme
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType,

--RegistryEntry attributes
  expirationDate,
  majorVersion,
  minorVersion,
  stability,
  status,
  userVersion

 FROM ExtrinsicObject
 UNION

 SELECT
--Object Attributes
  accessControlPolicy,
  id,
  objectType,


--RegistryEntry attributes
  expirationDate,
  majorVersion,
  minorVersion,
  stability,
  status,
  userVersion

 FROM RegistryPackage;

--Following is a partial list of indexes. Will need to add more.

--id index
CREATE INDEX id_ASSOCIATION_index ON ASSOCIATION(id);
CREATE INDEX id_AuditableEvent_index ON AuditableEvent(id);
CREATE INDEX id_Classification_index ON Classification(id);
CREATE INDEX id_ClassificationNode_index ON ClassificationNode(id);
CREATE INDEX id_ClassificationScheme_index ON ClassificationScheme(id);
CREATE INDEX id_ExternalIdentifier_index ON ExternalIdentifier(id);
CREATE INDEX id_ExternalLink_index ON ExternalLink(id);
CREATE INDEX id_ExtrinsicObject_index ON ExtrinsicObject(id);
CREATE INDEX id_Organization_index ON Organization(id);
CREATE INDEX id_RegistryPackage_index ON RegistryPackage(id);
CREATE INDEX id_Service_index ON Service(id);
CREATE INDEX id_ServiceBinding_index ON ServiceBinding(id);
CREATE INDEX id_SpecificationLink_index ON SpecificationLink(id);
CREATE INDEX id_User_index ON User_(id);

--name index
CREATE INDEX value_Name_index ON Name(value);
CREATE INDEX lang_value_Name_index ON Name(lang, value);

--description index
CREATE INDEX value_Description_index ON Description(value);
CREATE INDEX lang_value_Description_index ON Description(lang, value);

--Indexes on Association
```

```

CREATE INDEX sourceObject_Association_index ON Association(sourceObject);
CREATE INDEX targetObject_Association_index ON Association(targetObject);
CREATE INDEX associationType_Association_index ON Association(associationType);


--Indexes on Classification

CREATE INDEX classifiedObject_Classification_index ON Classification(classifiedObject);
CREATE INDEX classificationNode_Classification_index ON Classification(classificationNode);

--Indexes on ClassificationNode

CREATE INDEX parent_ClassificationNode_index ON ClassificationNode(parent);
CREATE INDEX code_ClassificationNode_index ON ClassificationNode(code);

--Indexes on ExternalIdentifier

CREATE INDEX registryObject_ExternalIdentifier_index ON ExternalIdentifier(registryObject);

--Indexes on ExternalLink

CREATE INDEX externalURI_ExternalLink_index ON ExternalLink(externalURI);

--Indexes on ExtrinsicObject

CREATE INDEX status_ExtrinsicObject_index ON ExtrinsicObject(status);

--Indexes on Organization

CREATE INDEX parent_Organization_index ON Organization(parent);

--Indexes on PostalAddress

CREATE INDEX parent_PostalAddress_index ON PostalAddress(parent);
CREATE INDEX city_PostalAddress_index ON PostalAddress(city);
CREATE INDEX country_PostalAddress_index ON PostalAddress(country);
CREATE INDEX postalCode_PostalAddress_index ON PostalAddress(postalCode);


--Indexes on ServiceBinding

CREATE INDEX service_ServiceBinding_index ON ServiceBinding(service);

--Indexes on Slot

CREATE INDEX parent_Slot_index ON Slot(parent);
CREATE INDEX name_Slot_index ON Slot(name);

--Indexes on SpecificationLink

CREATE INDEX service_SpecificationLink_index ON SpecificationLink(service);
CREATE INDEX serviceBinding_SpecificationLink_index ON SpecificationLink(serviceBinding);

--Indexes on TelephoneNumber

CREATE INDEX parent_TelephoneNumber_index ON TelephoneNumber(parent);

--Indexes on User

CREATE INDEX organization_User_index ON User_(organization);
CREATE INDEX personName_lastName_User_index ON User_(personName_lastName);

--Defines Stored procedures that map to RIM class methods

--Procedures on RegistryObject
CREATE PROCEDURE RegistryObject_associatedObjects(registryEntryId) {
--Must return a collection of UUIDs for related RegistryObject instances
}

CREATE PROCEDURE RegistryObject_auditTrail(registryEntryId) {
--Must return an collection of UUIDs for AuditableEvents related to the RegistryObject.
--Collection must be in ascending order by timestamp
}

```

```
CREATE PROCEDURE RegistryObject_externalLinks(registryEntryId) {
--Must return a collection of UUIDs for ExternalLinks annotating this RegistryObject.
}

CREATE PROCEDURE RegistryObject_externalIdentifiers(registryEntryId) {
--Must return a collection of UUIDs for ExternalIdentifiers for this RegistryObject.
}

CREATE PROCEDURE RegistryObject_classifications(registryEntryId) {
--Must return a collection of UUIDs for Classifications classifying this RegistryObject.
}

CREATE PROCEDURE RegistryObject_packages(registryEntryId) {
--Must return a collection of UUIDs for Packages that this RegistryObject belongs to.
}


--Procedures on RegistryPackage
CREATE PROCEDURE RegistryPackage_memberObjects(packageId) {
--Must return a collection of UUIDs for RegistryObjects that are memebers of this Package.
}

--Procedures on ExternalLink
CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
--Must return a collection of UUIDs for objects in this relationship
}

--Procedures on ClassificationNode
CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
--Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
```

Appendix E Non-normative Content Based Ad Hoc Queries

The Registry SQL query capability supports the ability to search for content based not only on metadata that catalogs the content but also the data contained within the content itself. For example it is possible for a client to submit a query that searches for all Collaboration Party Profiles that define a role named “seller” within a RoleName element in the CPP document itself. Currently content-based query capability is restricted to XML content.

E.1.1 Automatic Classification of XML Content

Content-based queries are indirectly supported through the existing classification mechanism supported by the Registry.

A submitting organization may define logical indexes on any XML schema or DTD when it is submitted. An instance of such a logical index defines a link between a specific attribute or element node in an XML document tree and a ClassificationNode in a classification scheme within the registry.

The registry utilizes this index to automatically classify documents that are instances of the schema at the time the document instance is submitted. Such documents are classified according to the data contained within the document itself.

Such automatically classified content may subsequently be discovered by clients using the existing classification-based discovery mechanism of the Registry and the query facilities of the ObjectQueryManager.

[Note] `This approach is conceptually similar to the way databases support indexed retrieval. DBAs define indexes on tables in the schema. When data is added to the table, the data gets automatically indexed.`

E.1.2 Index Definition

This section describes how the logical indexes are defined in the SubmittedObject element defined in the Registry DTD. The complete Registry DTD is specified in Appendix A.

A SubmittedObject element for a schema or DTD may define a collection of ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList is ignored if the content being submitted is not of the SCHEMA objectType.

The ClassificationIndex element inherits the attributes of the base class RegistryObject in [ebRIM]. It then defines specialized attributes as follows:

1. classificationNode: This attribute references a specific ClassificationNode by its ID.
2. contentIdentifier: This attribute identifies a specific data element within the document instances of the schema using an XPATH expression as defined by [XPT].

E.1.3 Example Of Index Definition

To define an index that automatically classifies a CPP based upon the roles defined within its RoleName elements, the following index must be defined on the CPP schema or DTD:

```
<ClassificationIndex
        classificationNode='id-for-role-classification-scheme'
        contentIdentifier='/Role//RoleName'
/>
```

E.1.4 Proposed XML Definition

```
<!--
A ClassificationIndexList is specified on ExtrinsicObjects of objectType
'Schema' to define an automatic Classification of instance objects of the
schema using the specified classificationNode as parent and a
ClassificationNode created or selected by the object content as selected
by the contentIdentifier
-->
<!ELEMENT ClassificationIndex EMPTY>
<!ATTLIST ClassificationIndex
         %ObjectAttributes;
         classificationNode IDREF #REQUIRED
         contentIdentifier CDATA #REQUIRED
>

<!-- ClassificationIndexList contains new ClassificationIndexes  -->
<!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

E.1.5 Example of Automatic Classification

Assume that a CPP is submitted that defines two roles as “seller” and “buyer." When the CPP is submitted it will automatically be classified by two ClassificationNodes named “buyer” and “seller” that are both children of the ClassificationNode (e.g. a node named Role) specified in the classificationNode attribute of the ClassificationIndex. If either of the two ClassificationNodes named “buyer” and “seller” did not previously exist, the ObjectManager would automatically create these ClassificationNodes.

Appendix F Security Implementation Guideline

This section provides a suggested blueprint for how security processing may be implemented in the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have different implementations as long as they support the default security roles and authorization rules described in this document.

F.1 Authentication

1. As soon as a message is received, the first work is the authentication. A principal object is created.
2. If the message is signed, it is verified (including the validity of the certificate) and the DN of the certificate becomes the identity of the principal. Then the Registry is searched for the principal and if found, the roles and groups are filled in.
3. If the message is not signed, an empty principal is created with the role RegistryGuest. This step is for symmetry and to decouple the rest of the processing.
4. Then the message is processed for the command and the objects it will act on.

F.2 Authorization

For every object, the access controller will iterate through all the AccessControlPolicy objects with the object and see if there is a chain through the permission objects to verify that the requested method is permitted for the Principal. If any of the permission objects which the object is associated with has a common role, or identity, or group with the principal, the action is permitted.

F.3 Registry Bootstrap

When a Registry is newly created, a default Principal object should be created with the identity of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message signed by the Registry Admin will get all the privileges.

When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the default AccessControlPolicy. This includes the creation of the necessary Permission instances as well as the Privilges and Privilege attributes.

F.4 Content Submission – Client Responsibility

The Registry client has to sign the contents before submission – otherwise the content will be rejected.

F.5 Content Submission – Registry Responsibility

1. As with any other request, the client will first be authenticated. In this case, the Principal object will get the DN from the certificate.
2. As per the request in the message, the RegistryEntry will be created.
3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
4. If a principal with the identity of the SO is not available, an identity object with the SO's DN is created.

5. A principal with this identity is created.

F.6 Content Delete/Deprecate – Client Responsibility

The Registry client has to sign the payload (not entire message) before submission, for authentication purposes; otherwise, the request will be rejected

F.7 Content Delete/Deprecate – Registry Responsibility

1. As with any other request, the client will first be authenticated. In this case, the Principal object will get the DN from the certificate. As there will be a principal with this identity in the Registry, the Principal object will get all the roles from that object
2. As per the request in the message (delete or deprecate), the appropriate method in the RegistryObject class will be accessed.
3. The access controller performs the authorization by iterating through the Permission objects associated with this object via the singleton default AccessControlPolicy.
4. If authorization succeeds then the action will be permitted. Otherwise an error response is sent back with a suitable AuthorizationException error message.

Appendix G Native Language Support (NLS)

G.1 Definitions

Although this section discusses only character set and language, the following terms have to be defined clearly.

G.1.1 Coded Character Set (CCS):

CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

G.1.2 Character Encoding Scheme (CES):

CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are ISO-2022, UTF-8.

G.1.3 Character Set (charset):

- charset is a set of rules for mapping from a sequence of octets to a sequence of characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- A list of registered character sets can be found at [IANA].

G.2 NLS And Request / Response Messages

For the accurate processing of data in both registry client and registry services, it is essential to know which character set is used. Although the body part of the transaction may contain the charset in xml encoding declaration, registry client and registry services shall specify charset parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a text/xml entity is received with the charset parameter omitted, MIME processors and XML processors MUST use the default charset value of "us-ascii". For example:

```
        Content-Type: text/xml; charset=ISO-2022-JP
```

Also, when an application/xml entity is used, the charset parameter is optional, and registry client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which directly address this contingency.

If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

G.3 NLS And Storing of RegistryObject

This section provides NLS guidelines on how a registry should store ***RegistryObject*** instances.

A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple locales. Thus there is no language or character set associated with a specific RegistryObject instance.

A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows. Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed as LocalizedString sub-elements within the XML element representing the I18N capable attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a specific locale. Each LocalizedString element has a charset and lang attribute as well as a value attribute of type string.

G.3.1 Character Set of *LocalizedString*

This is basically an implementation issue because the actual character set that the ***LocalizedString*** is stored with, does not affect the interface. However, it is highly recommended to use UTF-8 or UTF-16 for maximum inter-operability.

G.3.2 Language Information of *LocalizedString*

The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

G.4 NLS And Storing of Repository Items

This section provides NLS guidelines on how a registry should store repository items.

While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is always associated with a single repository item. The repository item may be in a single locale or may be in multiple locales. This specification does not specify the repository item.

G.4.1 Character Set of Repository Items

The MIME **`Content-Type`** mime header for the mime multi-part containing the repository item MAY contain a "**`charset`**" attribute that specifies the character set used by the repository item. For example:

```
Content-Type: text/xml; charset="UTF-8"
```

It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset of a repository item must be preserved as it is originally specified in the transaction.

G.4.2 Language information of repository item

The Content-language mime header for the mime bodypart containing the repository item may specify the language for a locale specific repository item. The value of the Content-language mime header property must conform to [RFC 1766].

This document currently specifies only the method of sending the information of character set and language, and how it is stored in a registry. However, the language information may be used as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation procedure, like registry client is asking a favorite language for messages from registry services, could be another functionality for the future revision of this document.

Appendix H Terminology Mapping

While every attempt has been made to use the same terminology used in other works there are some terminology differences. The following table shows the terminology mapping between this specification and that used in other specifications and working groups.

Table 12: Terminology Mapping Table

This Document	**OASIS**	**ISO 11179**
"repository item"	RegisteredObject	
RegistryEntry	RegistryEntry	Administered Component
ExternalLink	RelatedData	N/A
Object.id	regEntryId, orgId, etc.	
ExtrinsicObject.uri	objectURL	
ExtrinsicObject.objectType	defnSource, objectType	
RegistryEntry.name	commonName	
Object.description	shortDescription, Description	
ExtrinsicObject.mimeType	objectType="mime" fileType="<mime type>"	
Versionable.majorVersion	userVersion only	
Versionable.minorVersion	userVersion only	
RegistryEntry.status	registrationStatus	

References

[Bra97] Keywords for use in RFCs to Indicate Requirement Levels.

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

[TA] ebXML Technical Architecture
http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf

[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

[ebRIM] ebXML Registry Information Model
http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf

[ebBPM] ebXML Business Process Specification Schema
http://www.ebxml.org/specdrafts/Busv2-0.pdf

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

[ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps
http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html

[CTB] Context table informal document from Core Components

[ebMS] ebXML Messaging Service Specification, Version 0.21
http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf

[ERR] ebXML TRP Error Handling Specification
http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf

[SEC] ebXML Risk Assessment Technical Report, Version 3.6
http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html

[XPT] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[SQL] Structured Query Language (FIPS PUB 127-2)
http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
(SQL/PSM) [ISO/IEC 9075-4:1996]

[IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

[RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html

[RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html

[RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html

[RFC 3023] ETF (Internet Engineering Task Force). RFC 3023:

XML Media Types, ed. M. Murata. 2001.
ftp://ftp.isi.edu/in-notes/rfc3023.txt

[REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
http://www.w3.org/TR/REC-xml

[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

[WSDL]W3C Note. Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[SOAP11]W3C Note. Simple Object Access Protocol, May 2000,
http://www.w3.org/TR/SOAP

[SOAPAt]W3C Note: SOAP with Attachments, Dec 2000,
http://www.w3.org/TR/SOAP-attachments

[XMLDSIG] XML-Signature Syntax and Processing,
http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

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.

Contact Information

Team Leader

Name:	Lisa Carnahan
Company:	
Street:	
City, State, Postal Code:	
Country:	USA
Phone:	
Email:	lisa.carnahan@????

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:	Anne A. Fischer
Company:	Drummond Group, Inc.
Street:	4700 Bryant Irvin Ct., Suite 303
City, State, Postal Code:	Fort Worth, Texas 76107-7645
Country:	USA
Phone:	817-371-2367
Email:	anne@drummondgroup.com

Copyright Statement