Creating A Single Global Electronic Market

OASIS/ebXML Registry Services Specification v2.0

-Approved OASIS Standard

OASIS/ebXML Registry Technical Committee

April 2002

This page intentionally left blank.

1 Status of this Document

This document is an Approved OASIS Standard - April 2002.

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/documents/2.0/specs/ebRS.pdf

***Latest version*:**
http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf

2 OASIS/ebXML Registry Technical Committee

Prior to submission to the OASIS membership, the OASIS/ebXML Registry Technical Committee approved this document in its current form in December 2001. At the time of this approval the following were members of the OASIS/ebXML Registry Technical Committee.

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

2.1 Contributors

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

Len Gallagher, NIST
Sekhar Vajjhala, Sun Microsystems

Table of Contents

Table of Figures

Table of Tables

3 Introduction

3.1 Summary of Contents of Document

This document 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 resides in a repository for storage and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the Registry by a RegistryObject instance.

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

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

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* [ebBPSS]
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

This version of the Registry Services Specification is the second in a series of phased deliverables. Later versions of the document will include additional capability as deemed appropriate by the OASIS/ebXML Registry Technical Committee. It is assumed that:

Interoperability requirements dictate 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 LifeCycleManager 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 LifeCycleManager 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 version 2.0. Some of the actors are defined in Section 9.7. Note that the same entity may represent different actors. 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 have the same identity 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 MAY have the same identity as SO

		(RO)	
Registry Client	Registered User or Registered Guest		

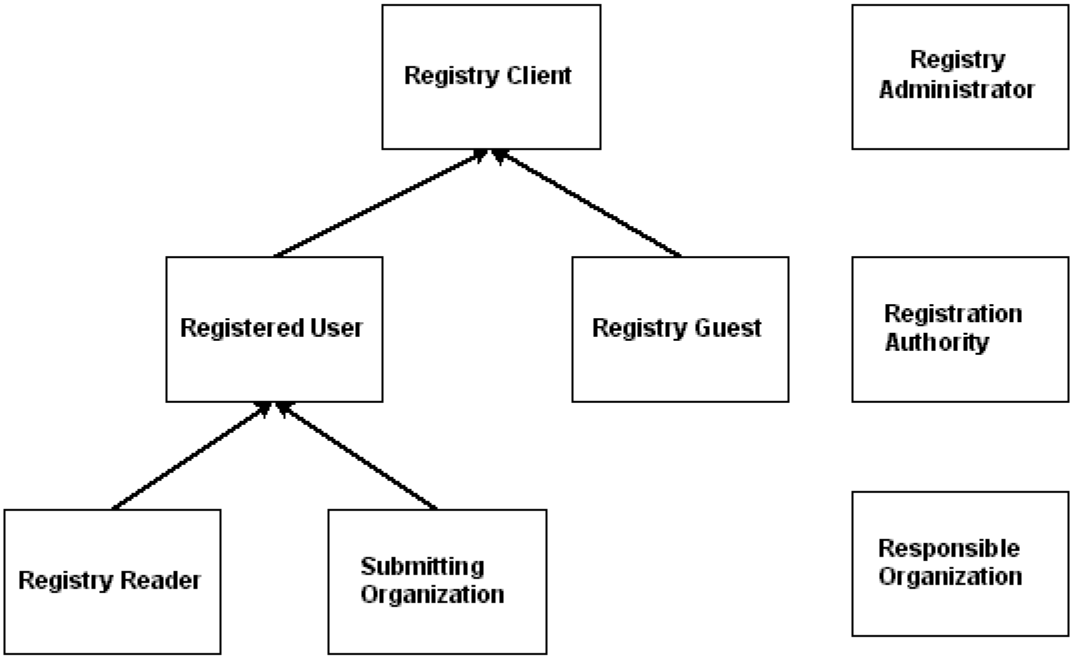


Figure 1: Actor Relationships

Note:

In the current version of the specification the following are true.

A Submitting Organization and a Responsible Organization are the same.

Registration of a user happens out-of-band, i.e, by means not specified in this specification.

A Registry Administrator and Registration Authority are the same.

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 5.5.1. An implementation is a conforming ebXML Registry Client if the implementation meets the conditions in Section 5.5.2. An implementation is a conforming ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to the conditions of Section 5.5.1 and Section 5.5.2. 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 Schema (Appendix B).
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 Schema (Appendix B).

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.

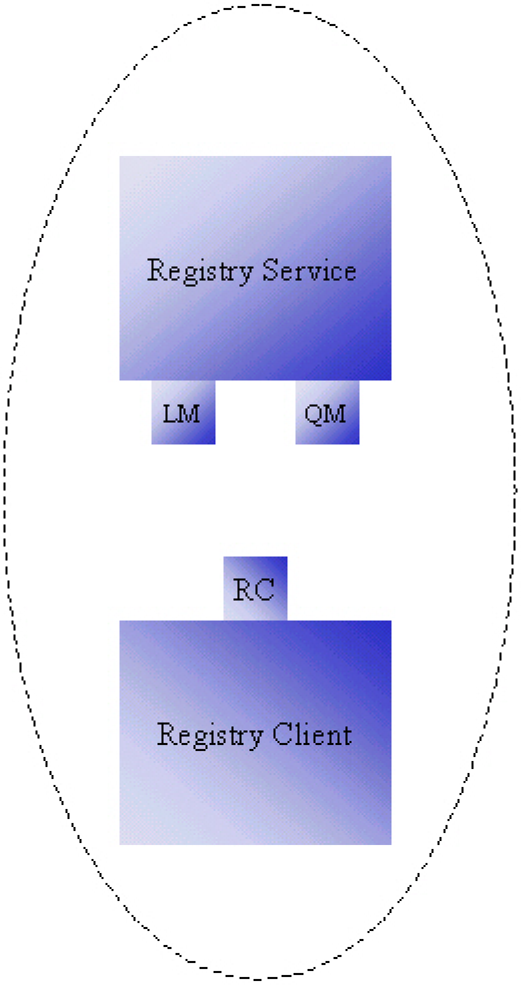


Figure 2: ebXML Registry Service Architecture

6.1 Registry Service Described

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).

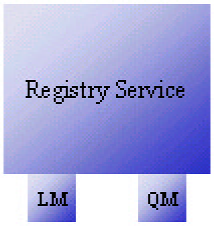


Figure 3: The Abstract ebXML Registry Service

Appendix A provides hyperlinks to 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.

[1] Known as ObjectQueryManager in V1.0

[2] Known as ObjectManager in V1.0

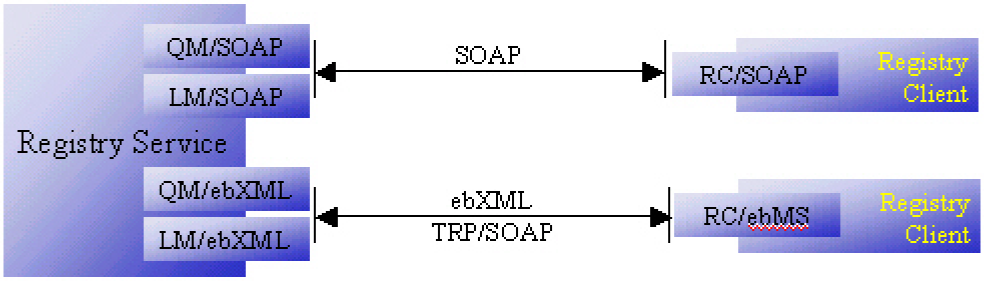


Figure 4: A Concrete ebXML Registry Service

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.

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 may be used to define a concrete service by binding it to a specific protocol. This binding is done 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 can be obtained via a hyperlink in Appendix A.

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 as shown below:

- MessageHeader;
- RegistryResponse element including;
  - Status attribute (Success, Failure);

  - – 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);
  - – 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.

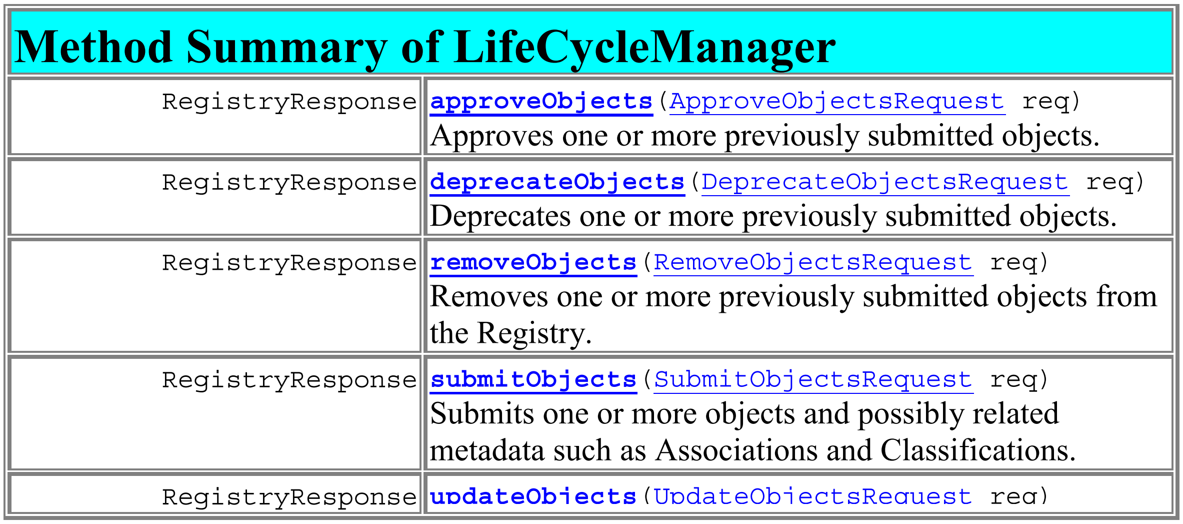
This specification does not mandate the use of a CPA between the Registry and the Registry Client. However if the Registry does not use a CPP, the Registry shall provide an alternate mechanism for the Registry Client to discover the services and other information provided by a CPP. This alternate mechanism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

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.

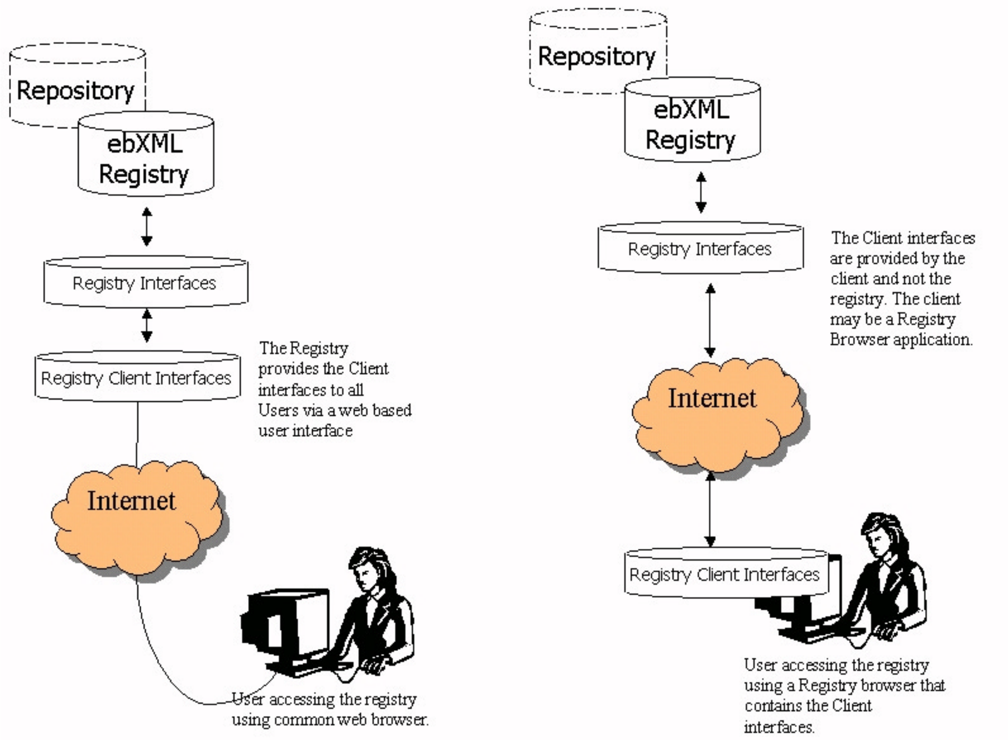

Repository
ebXML Registry
Registry Interfaces
Registry Client Interfaces
The Registry provides the Client interfaces to all Users via a web based user interface
Internet
User accessing the registry using common web browser.
Repository
ebXML Registry
Registry Interfaces
The Client interfaces are provided 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 bootstrapping between 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.1. 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.1. 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 via 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

The RegistryResponse is a common class defined by the Registry interface that is used by the registry to provide responses to client requests.

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 LifeCycleManagement 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.

7.1 Life Cycle of a Repository Item

The main purpose of the LifeCycleManagement service is to manage the life cycle of repository items. Figure 6 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

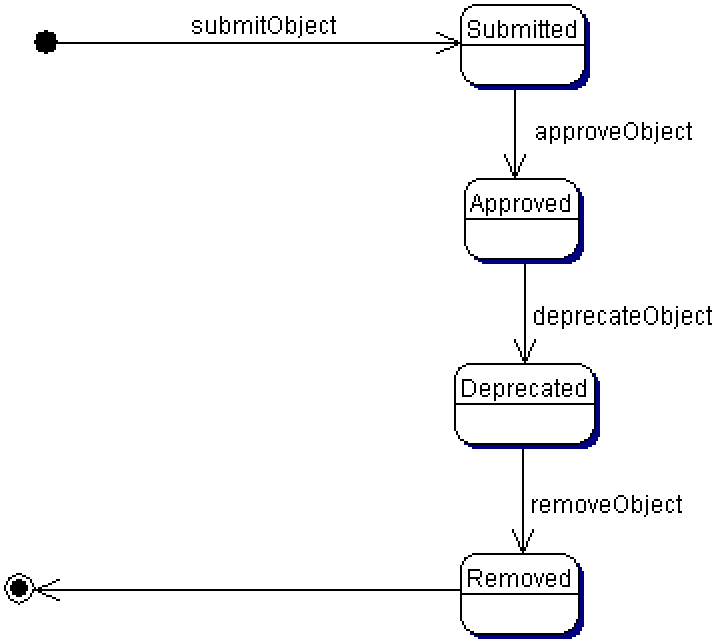


Figure 6: Life Cycle of a Repository Item

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 other elements (See Appendix B.1 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 LifeCycleManager on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

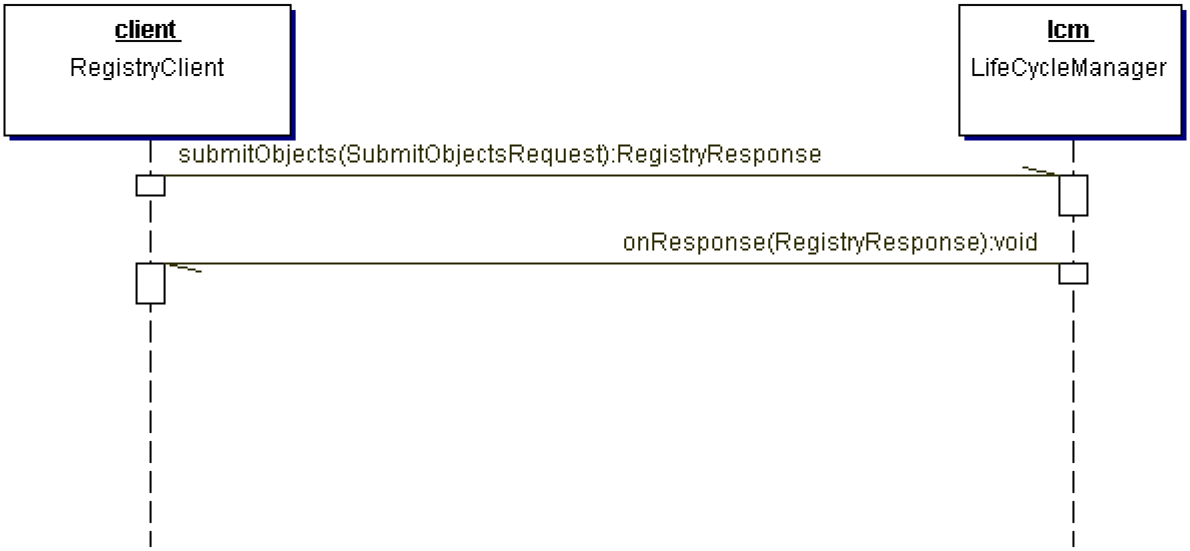

client
RegistryClient
lcm
LifeCycleManager
submitObjects(SubmitObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 7: 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 LeafRegistryObjectList element.

The LeafRegistryObjectList 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 SubmitterOf 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. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of “Uavailable” 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"
  >

  <rim:LeafRegistryObjectList>

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

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

    <rim:ExtrinsicObject id = "acmeCPP1"  >
      <rim:Name>
        <rim:LocalizedString value = "Widget Profile" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's profile for selling widgets" />
      </rim: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" >
      <rim:Name>
        <rim:LocalizedString value = "RegistryPackage #2"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's package #2"/>
      </rim: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" >
      <rim:Name>
        <rim:LocalizedString value = "Link #1"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's Link #1"/>
      </rim:Description>
    </rim:ExternalLink>

    <rim:ExtrinsicObject id = "acmeCPP2"  >
      <rim:Name>
        <rim:LocalizedString value = "Sprockets Profile" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
      </rim: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">
      <rim:Name>
        <rim:LocalizedString value = "Link #2"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "ACME's Link #2"/>
      </rim: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" >
      <rim:Name>
        <rim:LocalizedString value = "DUNS"/>
      </rim:Name>

      <rim:Description>
        <rim:LocalizedString value = "This is the DUNS scheme"/>
      </rim:Description>
    </rim:ClassificationScheme>

    <rim:ExternalIdentifier id = "acmeDUNSId"  identificationScheme="DUNS-id" value =
"13456789012">
      <rim:Name>
        <rim:LocalizedString value = "DUNS" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "DUNS ID for ACME"/>
      </rim: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" >
      <rim:Name>
        <rim:LocalizedString value = "Geography"/>
      </rim:Name>

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

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

      <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
        <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
          <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
        </rim:ClassificationNode>
      </rim: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">
      <rim:Name>
        <rim:LocalizedString value = "Automotive" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
      </rim:Description>
    </rim:ClassificationNode>

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

    <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
      <rim:Name>
        <rim:LocalizedString value = "Engine Supplier" />
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
      </rim: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">
      <rim:Description>
        <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
      </rim:Description>
    </rim:Classification>

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

    <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
  </rim:LeafRegistryObjectList>
</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.

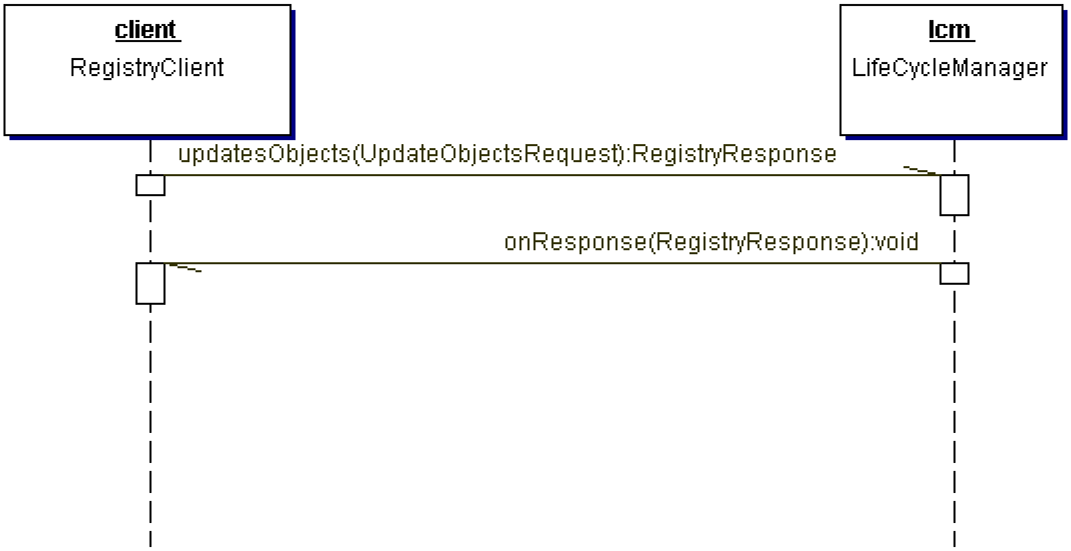

client
RegistryClient
lcm
LifeCycleManager
updatesObjects(UpdateObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 8: 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 LeafRegistryObjectList element. The LeafRegistryObjectList 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 SubmitterOf 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. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of “Uavailable” 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
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 LifeCycleManager. Slots provide a dynamic mechanism for extending registry entries as defined by [ebRIM].

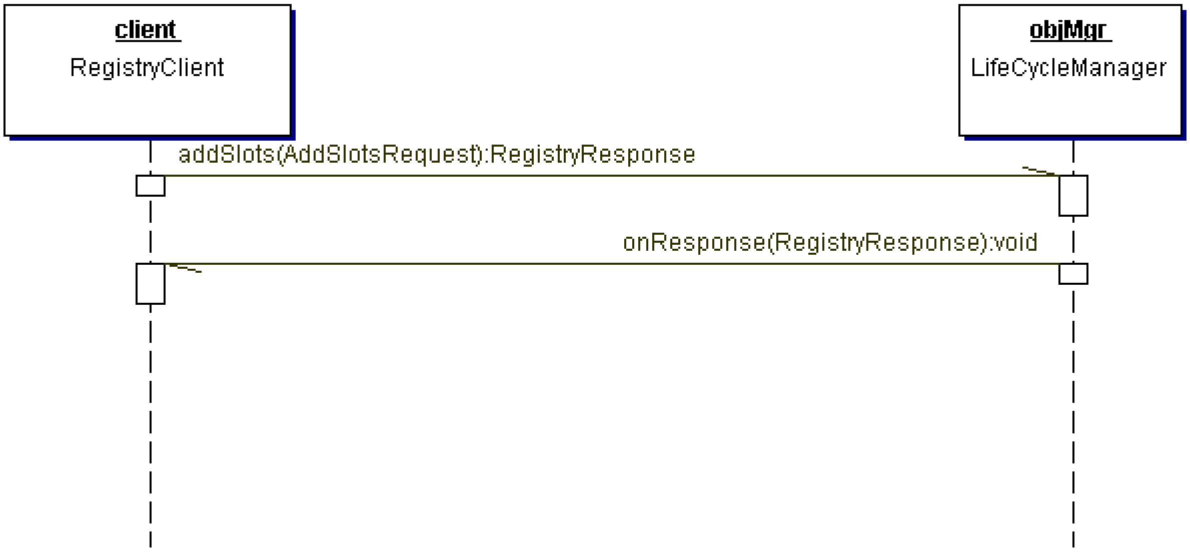

client
RegistryClient
objMgr
LifeCycleManager
addSlots(AddSlotsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 9: 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 LifeCycleManager.

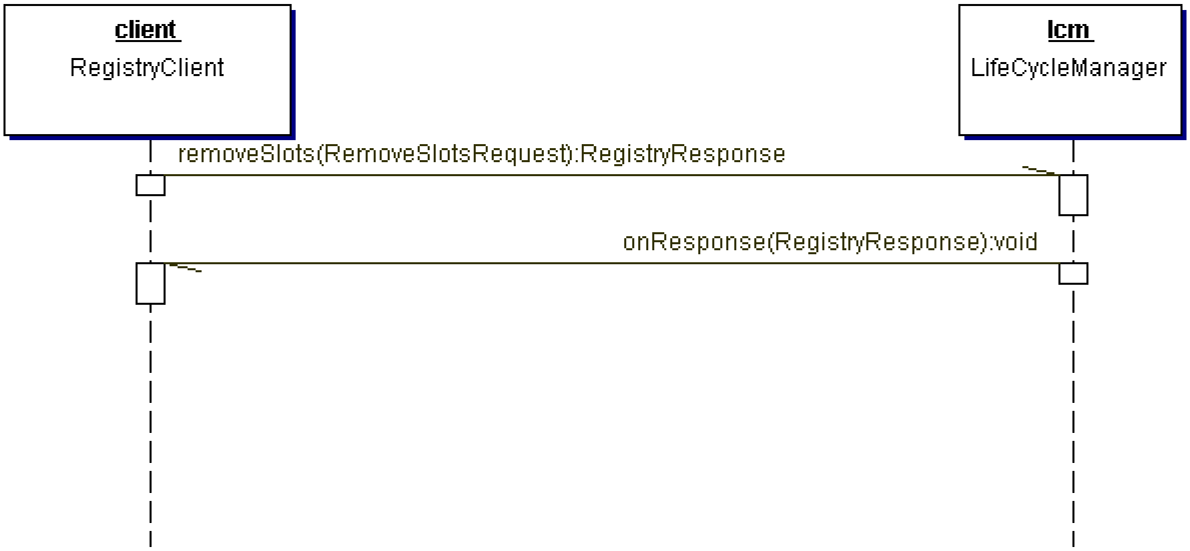

client
RegistryClient
lcm
LifeCycleManager
removeSlots(RemoveSlotsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 10: 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 LifeCycleManager. 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).

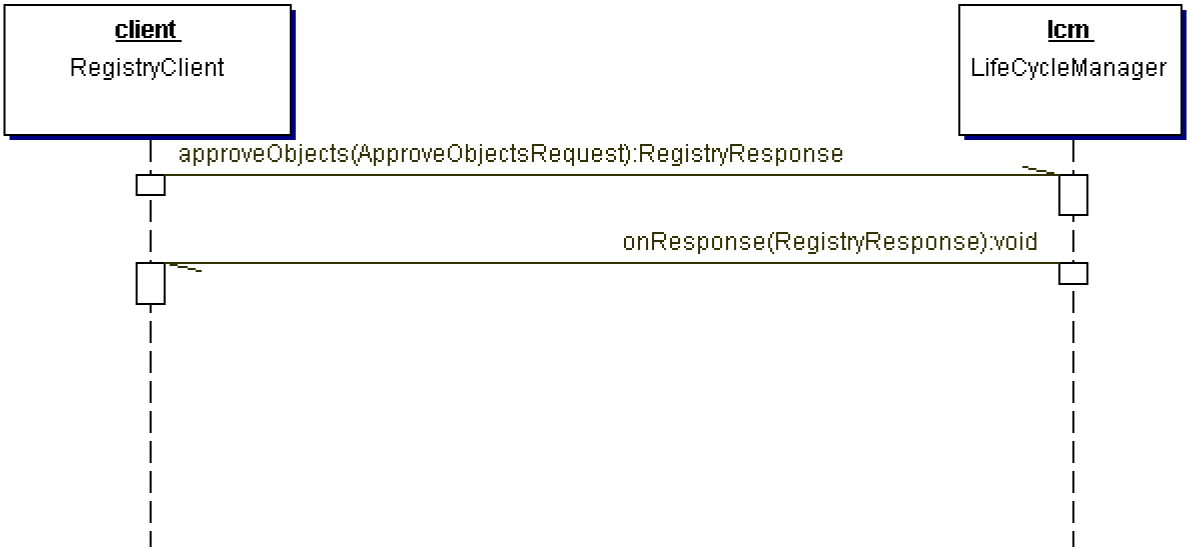

client
RegistryClient
lcm
LifeCycleManager
approveObjects(ApproveObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 11: 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 SubmitterOf 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. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of “Uavailable” 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 LifeCycleManager. 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.

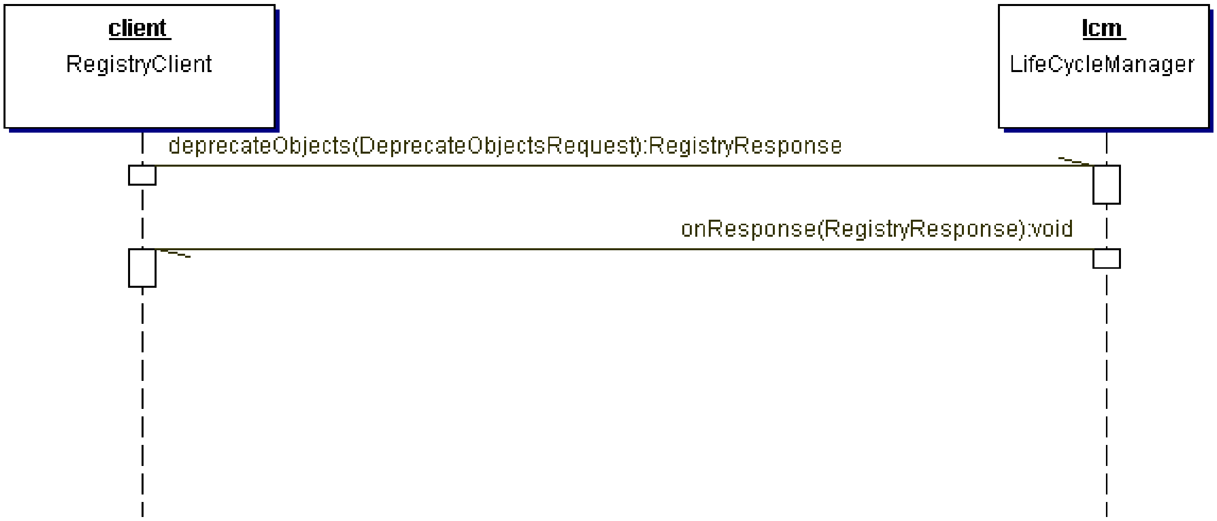

client
RegistryClient
lcm
LifeCycleManager
deprecateObjects(DeprecateObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 12: 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 SubmitterOf 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. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of "Uavailable" 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	Error

	Classes	
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 RegistryObject instances and/or repository items using the LifeCycleManager.

The RemoveObjectsRequest message is sent by a client to remove RegistryObject 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 RegistryObject and the repository item for the specified registry entries. Only if all references (e.g. Associations, Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to remove a RegistryObject while it still has references raises an error condition: InvalidRequestError.

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

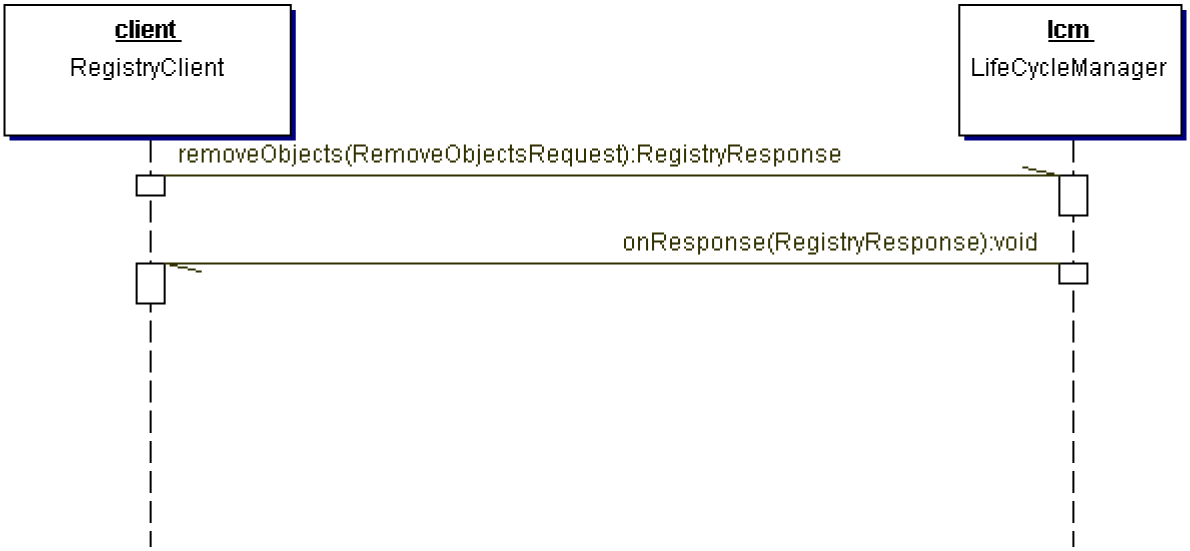

client
RegistryClient
lcm
LifeCycleManager
removeObjects(RemoveObjectsRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 13: 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. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of "Uavailable" 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	RegistryObject 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:

- Filter Query
- 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 Appendix H.

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.

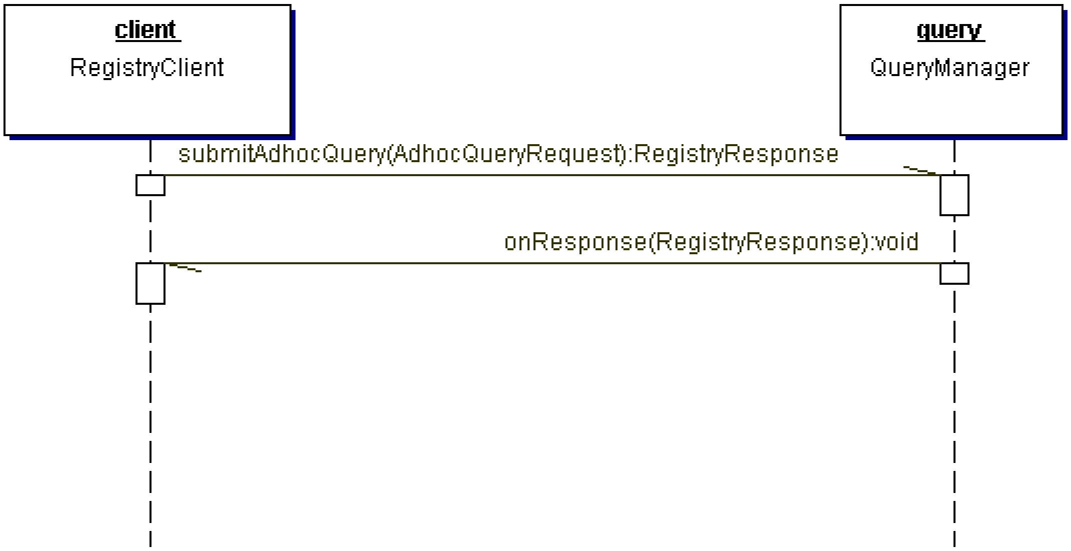

client
RegistryClient
query
QueryManager
submitAdhocQuery(AdhocQueryRequest):RegistryResponse
onResponse(RegistryResponse):void

Figure 14: 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="tns:FilterQuery" />
            <element ref="tns:SQLQuery" />
         </choice>
      </sequence>
   </complexType>
</element>

<element name="AdhocQueryResponse">
   <complexType>
      <choice minOccurs="1" maxOccurs="1”>
         <element ref="tns:FilterQueryResult" />
         <element ref="tns:SQLQueryResult" />
      </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 may contain a collection of ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return just the identifiers of the registry objects.
- RegistryObject - This option specifies that the AdhocQueryResponse may contain a collection of RegistryObject XML elements as defined in [ebRIM 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 may contain a collection of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which correspond to RegistryEntry or RegistryObject attributes.
- LeafClass - This option specifies that the AdhocQueryResponse may contain a collection of XML elements that correspond to leaf classes as defined in [ebRIM Schema].
- LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse may contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema] accompanied with their repository items or RegistryEntry or RegistryObject and their attributes. Linking of ExtrinsicObject and its repository item is done via contentURI as explained in Section 8.4 -Content Retrieval.

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 15

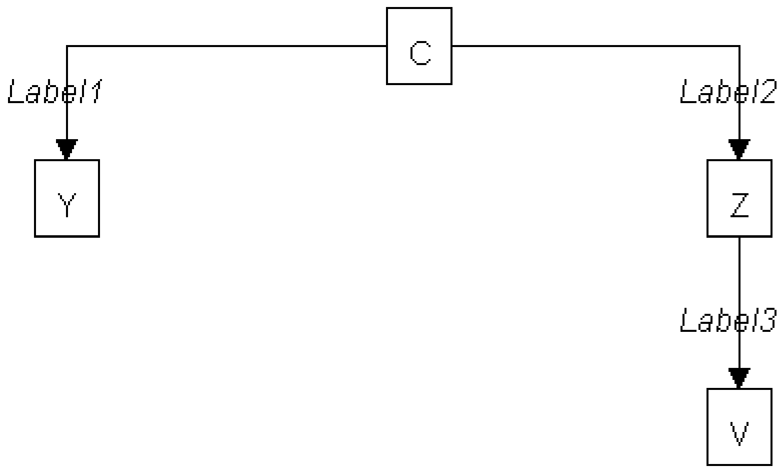


Figure 15: Example ebRIM Binding

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 8.2.13 and the supported predicate clauses are defined in Section 8.2.14. 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 [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in ebRIM whenever possible.

The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 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:AssociationQuery” />
         <element ref="tns:AuditableEventQuery" />
         <element ref=”tns:ClassificationQuery” />
         <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:AssociationQueryResult” />
         <element ref="tns:AuditableEventQueryResult" />
         <element ref=”tns:ClassificationQueryResult” />
         <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. Semantic rules specify the procedure for implementing the evaluation of Filter Queries. Implementations do not necessarily have to follow the same procedure provided that the same effect is achieved.
3. 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 [ebRIM Schema].
4. 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.
5. If a Filter, Branch or Query element for a class has no sub-elements then every persistent instance of that class satisfies the Filter, Branch or Query.
6. 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 AdHocQueryResult 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”.
7. 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 FilterQueryResult 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

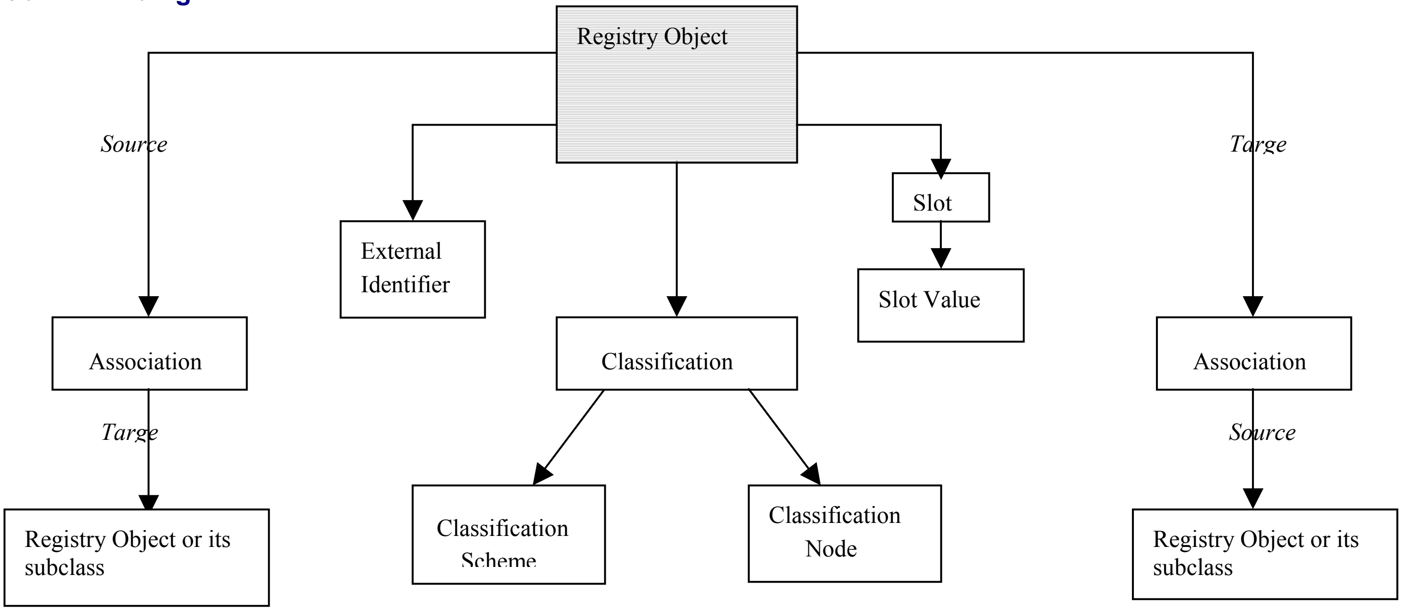


Figure 16: ebRIM Binding for RegistryObjectQuery

Definition

```
<complexType name="RegistryObjectQueryType">
  <sequence>
    <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
    <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
    <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
    <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
    <element ref="tns:TargetAssociationBranch" 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:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
         <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
         <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
         <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
         <element ref=”tns:AssociationQuery” minOccurs=”0” maxOccurs=”1” />
         <element ref=”tns:ClassificationQuery” 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:UserBranch" 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="ClassifiedByBranch">
   <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="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:EmailAddressFilter" minOccurs = "0" maxOccurs="unbounded"/>
              <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
          </sequence>
      </complexType>
  </element>

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

<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>
```

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 go to number 2 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 to the next numbered rule.

c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one 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 to the next numbered rule.

d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x doesn’t have an auditable event that satisfy AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is empty then continue to the next numbered rule.

e) 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 to the next numbered rule; otherwise treat NameBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by at least one of the LocalizedStrings that constitute the name of the registry object then remove x from RO. If RO is empty then continue to the next numbered rule.

f) 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 to the next numbered rule; 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 to the next numbered rule.

g) If a ClassifiedByBranch 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 classifiedObject of at least one Classification instance, then remove x from RO; otherwise, treat each ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified within the ClassifiedByBranch, 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule.

h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one Slot instance, then remove x from RO. If RO is empty then continue to the next numbered rule; 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 to the next numbered rule. 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. If RO is empty then continue to the next numbered rule.

i) 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 at least one Association instance, then remove x from RO. If RO is empty then continue to the next numbered rule; 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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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. If RO is empty then continue to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by at least one 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 to the next numbered rule. 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 8.2.11 then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Classification instances that satisfy the ClassificationQuery 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 to the next numbered rule (Rule 2).

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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 at least one 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 to the next numbered rule. 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 at least one 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 to the next numbered rule. If a ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and are the target service binding of some element of ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 at least one registry entry in RE, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be the set of Association instances that satisfy the AssociationQuery 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 to the next numbered rule (Rule 2).

j) 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 to the next numbered rule; 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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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. If RO is empty then continue to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 8.2.11 then remove u from ROS. If ROS is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Classification instances that satisfy the ClassificationQuery 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 to the next numbered rule (Rule 2).

If a ServiceBindingBranch is specified within the TargetAssociationBranch, 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule.

If a SpecificationLinkBranch is specified within the TargetAssociationBranch, 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule. If a ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and are the target service binding of some element of ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be the set of Association instances that satisfy the AssociationQuery 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 to the next numbered rule (Rule 2).

2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set RO to be the result of the RegistryObjectQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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>
```

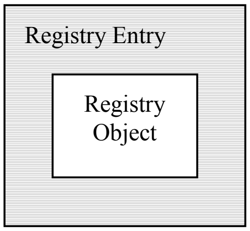
A client application wishes to identify all RegistryObject instances that are classified by some internal classification scheme and have some given keyword as part of the description of one of the classification nodes of that classification scheme. The following query identifies all such RegistryObject 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 17: 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 to the next numbered rule.
   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 to the next numbered rule.
   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, set RE to be the result of the RegistryEntryQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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>
      <TargetAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument = "associationType">
              <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
             <Clause>
              <SimpleClause leftArgument = "value">
               <StringClause stringPredicate = "Contains">XYZ</StringClause>
              </SimpleClause>
             </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </OrganizationQuery>
      </TargetAssociationBranch>
      <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 AssociationQuery

Purpose

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

ebRIM Binding

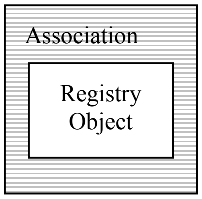


Figure 18: ebRIM Binding for AssociationQuery

Definition

```

<complexType name = "AssociationQueryType">
   <complexContent>
      <extension base = "tns:RegistryObjectQueryType">
         <sequence>
            <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name = "AssociationQuery" type = "tns:AssociationQueryType"/>

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

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

```

Semantic Rules

1. Let A denote the set of all persistent Association instances in the Registry. The following steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
    a) If A is empty then continue to the next numbered rule.
    b) If an AssociationFilter element is not directly contained in the AssociationQuery element, then go to the next step; otherwise let x be an association instance in A. If x does not satisfy the AssociationFilter then remove x from A. If A is empty then continue to the next numbered rule.
    c) Let A be the set of remaining Association instances. Evaluate inherited RegistryObjectQuery over A as explained in Section 8.2.2.
2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to be the result of the AssociationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify a set of associations that are ‘equivalentTo’ a set of other associations.

```

<AdhocQueryRequest">
  <ResponseOption returnType="LeafClass" />
  <FilterQuery>
    <AssociationQuery>
      <SourceAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument="associationType">
              <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <AssociationQuery>
          <AssociationFilter>
            <Clause>
              <SimpleClause leftArgument="associationType">
                <StringClause stringPredicate="StartsWith">Sin</StringClause>
              </SimpleClause>
            </Clause>
          </AssociationFilter>
        </AssociationQuery>
      </SourceAssociationBranch>
      <AssociationFilter>
```

```
        <Clause>
          <SimpleClause leftArgument="associationType">
            <StringClause stringPredicate="StartsWith">Son</StringClause>
          </SimpleClause>
        </Clause>
      </AssociationFilter>
    </AssociationQuery>
  </FilterQuery>
</AdhocQueryRequest>

```

8.2.5 AuditableEventQuery

Purpose

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

ebRIM Binding

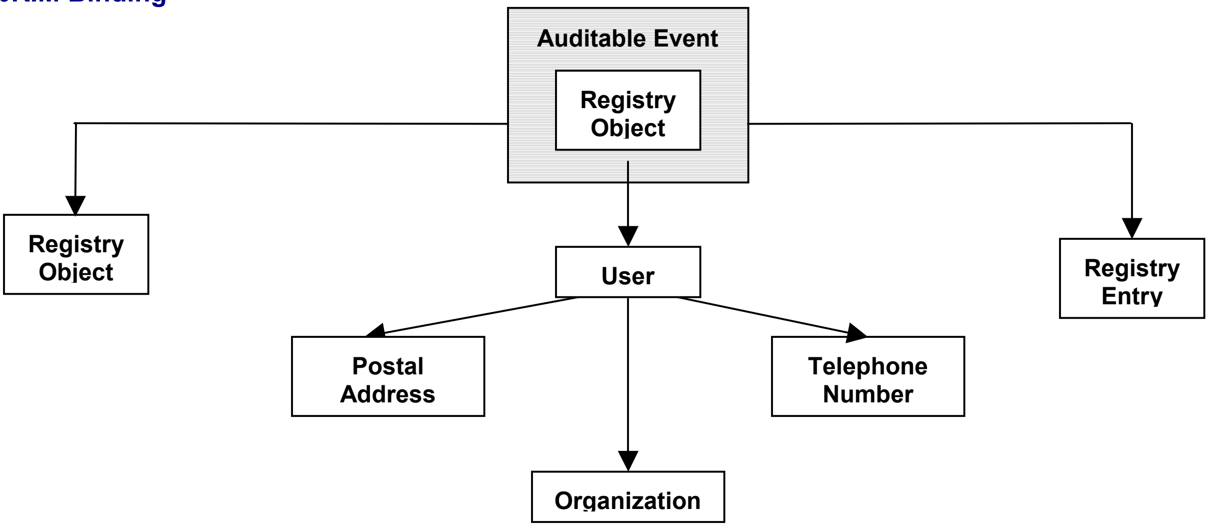


Figure 19: ebRIM Binding for AuditableEventQuery

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 to the next numbered rule.

   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 to the next numbered rule.

   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 to the next numbered rule.

   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 to the next numbered rule.

   e) If a 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 EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilters isn't satisfied by some of the email addresses 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 8.2.11 then remove x from AE. If AE is empty then continue to the next numbered rule.

   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 set AE to be the result of the AuditableEventQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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>
         <AuditableEventFilter>
            <Clause>
               <SimpleClause leftArgument = "timestamp">
                  <RationalClause logicalPredicate = "GE">
                     DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
                  </RationalClause>
               </SimpleClause>
            </Clause>
         </AuditableEventFilter>
         <RegistryEntryQuery>
            <NameBranch>
               <LocalizedStringFilter>
                  <Clause>
                     <SimpleClause leftArgument = "value">
                        <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
                     </SimpleClause>
                  </Clause>
               </LocalizedStringFilter>
            </NameBranch>
         </RegistryEntryQuery>
      </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>
         <RegistryEntryQuery>
            <TargetAssociationBranch>
               <AssociationFilter>
                  <Clause>
                    <SimpleClause leftArgument = "associationType">
                     <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
                    </SimpleClause>
                  </Clause>
               </AssociationFilter>
               <OrganizationQuery>
                  <NameBranch>
                     <LocalizedStringFilter>
               <Clause>
                <SimpleClause leftArgument = "value">
                <StringClause stringPredicate = "Equal">myorg</StringClause>
                </SimpleClause>
               </Clause>
              </LocalizedStringFilter>
            </NameBranch>
          </OrganizationQuery>
        </TargetAssociationBranch>
      </RegistryEntryQuery>
      <UserBranch>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "-Equal">myorg</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </OrganizationQuery>
      </UserBranch>
    </AuditableEventQuery>
  </FilterQuery>
</AdhocQueryRequest>

```

8.2.6 ClassificationQuery

Purpose

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

ebRIM Binding

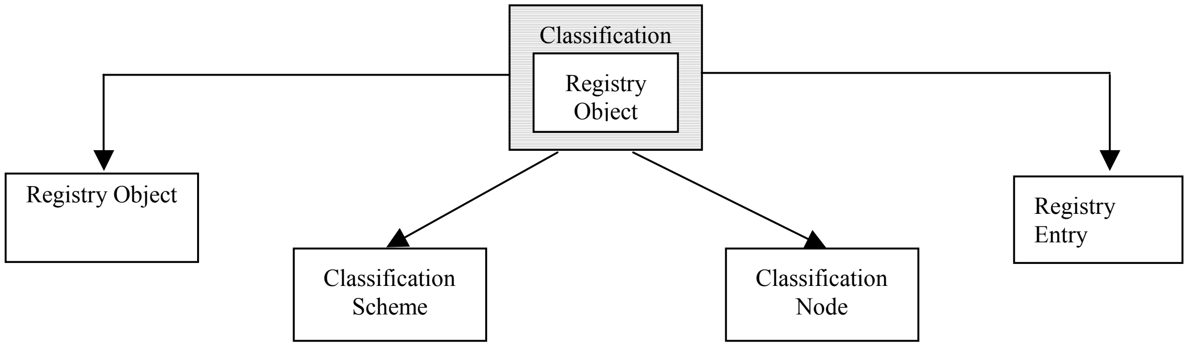


Figure 20: ebRIM Binding for ClassificationQuery

Definition

```

<complexType name = "ClassificationQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
      <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>
      </extension>
   </complexContent>
</complexType>
<element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>

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

Semantic Rules

1. Let C denote the set of all persistent Classification instances in the Registry. The following steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
   a) If C is empty then continue to the next numbered rule.
   b) If a ClassificationFilter element is not directly contained in the ClassificationQuery element, then go to the next step; otherwise let x be an classification instance in C. If x does not satisfy the ClassificationFilter then remove x from C. If C is empty then continue to the next numbered rule.
   c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C. If C is empty then continue to the next numbered rule.
   d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the classification node of x does not satisfy the ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is empty then continue to the next numbered rule.
   e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. 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 a classification of at least one registry object in RO, then remove x from C. If C is empty then continue to the next numbered rule.
   f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. 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 a classification of at least one registry entry in RE, then remove x from C. If C is empty then continue to the next numbered rule.

2. If C is empty, then raise the warning: *classification query result is empty*; otherwise otherwise, set C to be the result of the ClassificationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.7 ClassificationNodeQuery

Purpose

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

ebRIM Binding

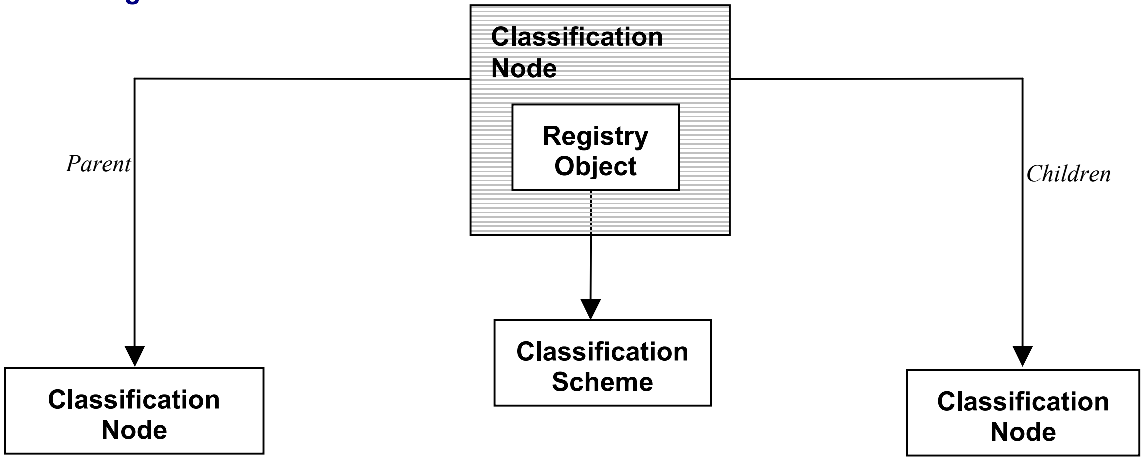


Figure 21: ebRIM Binding for ClassificationNodeQuery

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 to the next numbered rule.

   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 to the next numbered rule.

   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 8.2.8, then remove x from CN. If CN is empty then continue to the next numbered rule.

   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 to the next numbered rule. 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 to the next numbered rule.

      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 to the next numbered rule; 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 to the next numbered rule; 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 to the next numbered rule; 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 to the next numbered rule; 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 to the next numbered rule; 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 set CN to be the result of the ClassificationNodeQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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

8.2.8 ClassificationSchemeQuery

Purpose

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

ebRIM Binding

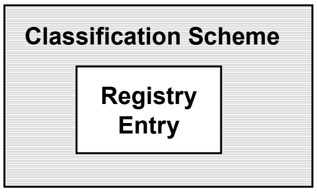


Figure 22: ebRIM Binding for ClassificationSchemeQuery

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 to the next numbered rule.

   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 to the next numbered rule.

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, set CS to be the result of the ClassificationSchemeQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify all classification scheme instances in the Registry.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ClassificationSchemeQuery/>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.9 RegistryPackageQuery

Purpose

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

ebRIM Binding

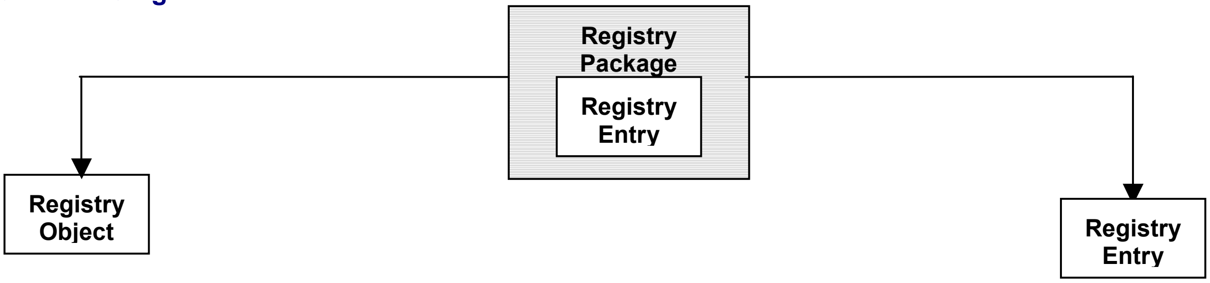


Figure 23: ebRIM Binding for RegistryPackageQuery

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 to the next numbered rule.

   b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule; 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 to the next numbered rule.

   c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryObjectQuery as follows: 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 to the next numbered rule. If a RegistryEntryQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryEntryQuery as follows: 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. If RP is empty then continue to the next numbered rule.

   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 set RP to be the result of the RegistryPackageQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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>
      <RegistryObjectQuery/>
    </RegistryPackageQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

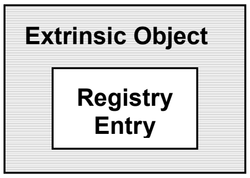
A client application wishes to identify all package instances in the Registry that are empty. Since the RegistryPackageQuery is not set up to do negations, clients will have to do two separate RegistryPackageQuery requests, one to find all packages and another to find all non-empty packages, and then 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.

8.2.10 ExtrinsicObjectQuery

Purpose

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



ebRIM Binding

Figure 24: 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 to the next numbered rule.
    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 to the next numbered rule.
    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, set EO to be the result of the ExtrinsicObjectQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.11 OrganizationQuery

Purpose

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

ebRIM Binding

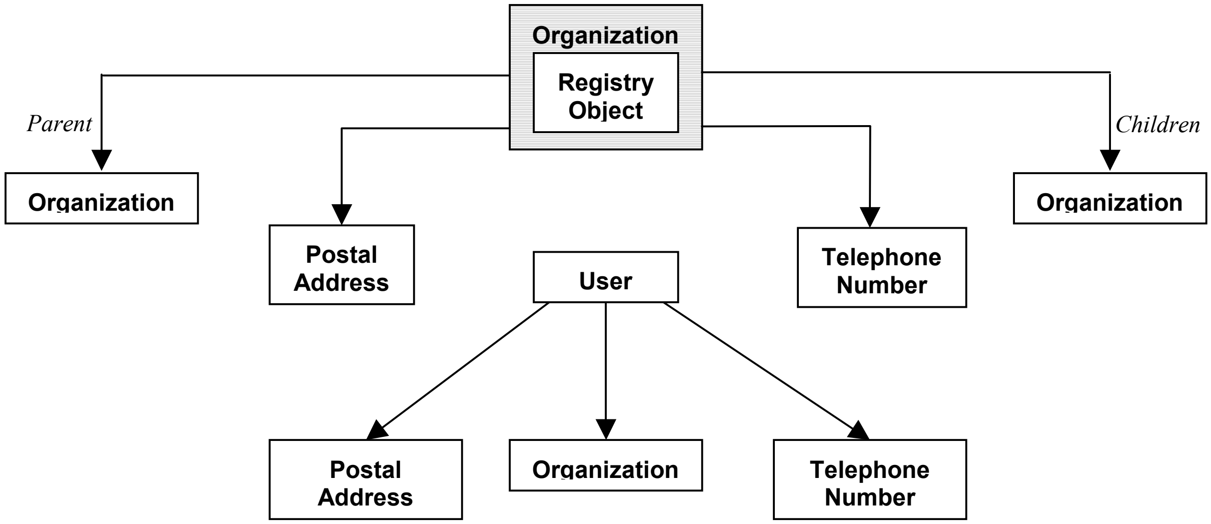


Figure 25: ebRIM Binding for OrganizationQuery

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule.

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 EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilters isn't satisfied by some of the email addresses 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 8.2.11 then remove x from ORG. If ORG is empty then continue to the next numbered rule.

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 to the next numbered rule.

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 to the next numbered rule; 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 to the next numbered rule; 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 to the next numbered rule; 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 to the next numbered rule. 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 to the next numbered rule; 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 to the next numbered rule; 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 set ORG to be the result of the OrganizationQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

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>
            <SourceAssociationBranch>
                <AssociationFilter>
                    <Clause>
                        <SimpleClause leftArgument = "associationType">
                            <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
                        </SimpleClause>
                    </Clause>
                </AssociationFilter>
                <RegistryObjectQuery>
                    <RegistryObjectFilter>
                        <Clause>
                            <SimpleClause leftArgument = "objectType">
                                <StringClause stringPredicate = "Equal">CPP</StringClause>
                            </SimpleClause>
                        </Clause>
                    </RegistryObjectFilter>
                    <AuditableEventQuery>
                        <AuditableEventFilter>
                            <Clause>
                                <SimpleClause leftArgument = "timestamp">
                                  <RationalClause logicalPredicate = "GE">
                                    <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
                                  </RationalClause>
                                </SimpleClause>
                            </Clause>
                        </AuditableEventFilter>
                    </AuditableEventQuery>
                 </RegistryObjectQuery>
            </SourceAssociationBranch>
            <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>
```

8.2.12 ServiceQuery

Purpose

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

ebRIM Binding

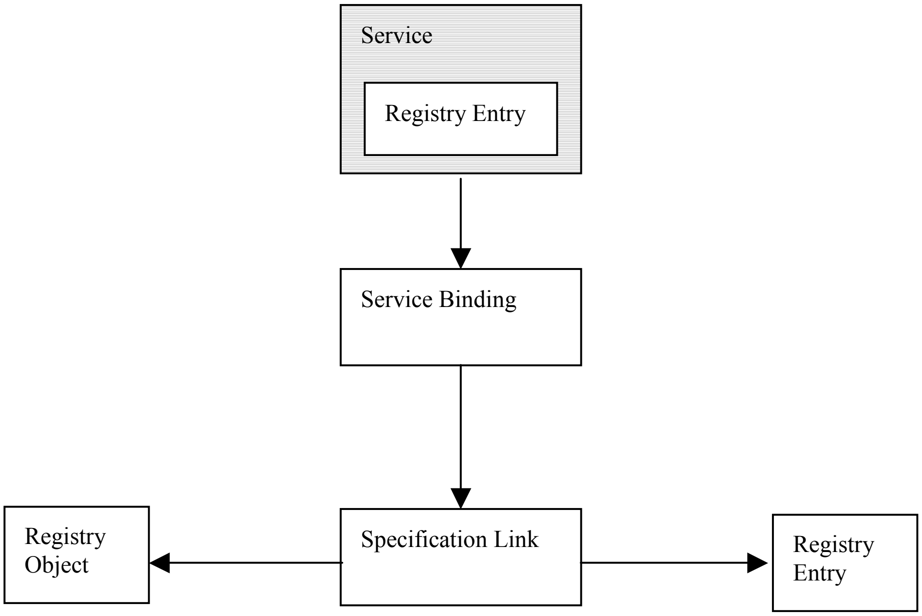


Figure 26: ebRIM Binding for ServiceQuery

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 to the next numbered rule.

   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 to the next numbered rule.

   c) If a ServiceBindingBranch is not specified then continue to the next numbered rule; 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 to the next numbered rule. If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then continue to the next numbered rule; 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 to the next numbered rule. 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 to the next numbered rule. 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 to the next numbered rule.

   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 set S to be the result of the ServiceQuery.

3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

8.2.13 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 8.2.14.

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*. If the leftAttribute is the visible attribute "path" then if stringPredicate of the StringClause is not "Equal" then raise exception: *classification node path 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.

8.2.14 XML Clause Constraint Representation

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 Diagram

The following is a conceptual diagram outlining the Clause structure.

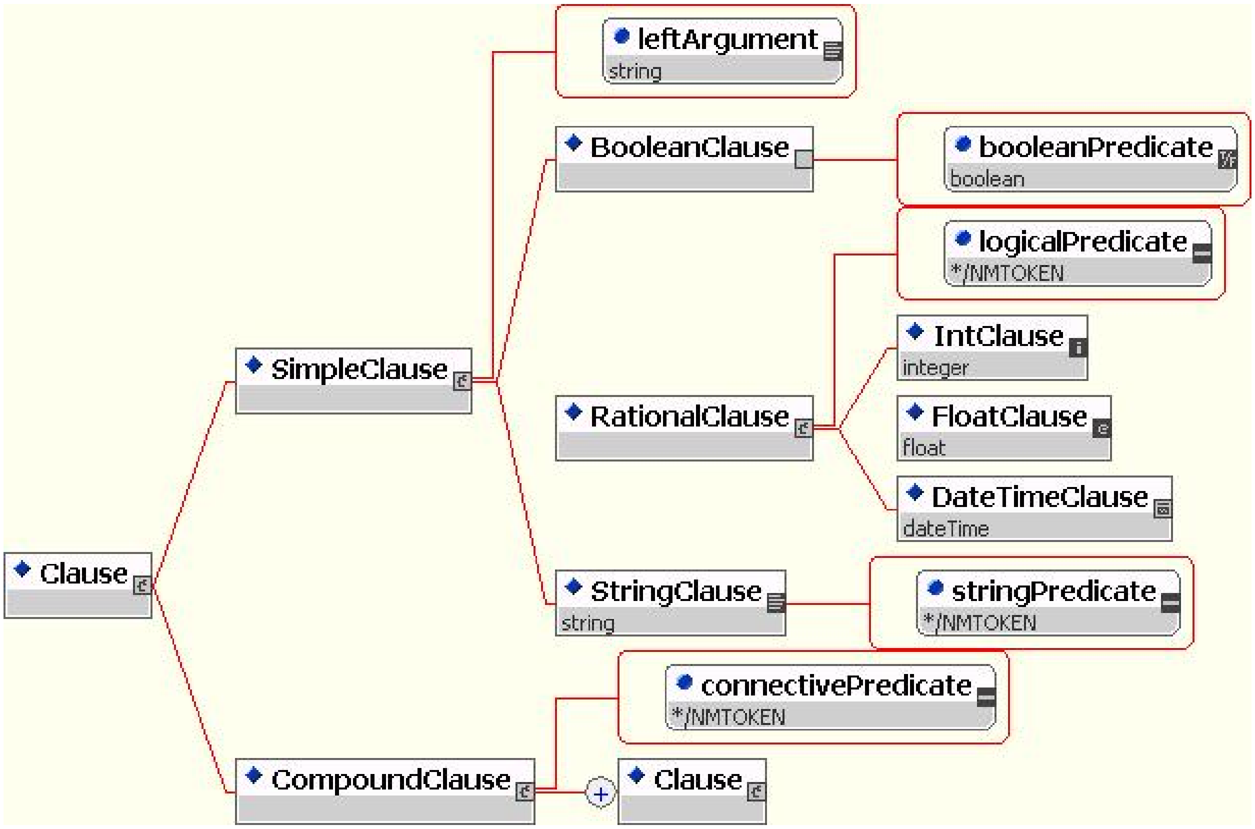


Figure 27: The Clause Structure

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.

SimpleClauses

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.

CompoundClauses

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 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" 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"/>
            <element ref = "tns:DateTimeClause"/>
         </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 = "DateTimeClause" type = "dateTime"/>

   <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>

```

Examples

Simple BooleanClause: "Smoker" = True

```
<Clause>
   <SimpleClause leftArgument="Smoker">
      <BooleanClause booleanPredicate="True"/>
   </SimpleClause>
</Clause>
```

Simple StringClause: "Smoker" contains "mo"

```
<Clause>
   <SimpleClause leftArgument = "Smoker">
      <StringClause stringPredicate = "Contains">mo</StringClause>
   </SimpleClause>
<Clause>
```

Simple IntClause: "Age" >= 7

```
<Clause>
   <SimpleClause leftArgument="Age">
      <RationalClause logicalPredicate="GE">
         <IntClause>7</IntClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

Simple FloatClause: "Size" = 4.3

```
<Clause>
   <SimpleClause leftArgument="Size">
      <RationalClause logicalPredicate="Equal">
         <FloatClause>4.3</FloatClause>
      </RationalClause>
   </SimpleClause>
</Clause>
```

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

```
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
            <BooleanClause booleanPredicate="False"/>
         </SimpleClause>
      </Clause>
      <Clause>
         <SimpleClause leftArgument="Age">
            <RationalClause logicalPredicate="LE">
               <IntClause>45</IntClause>
            </RationalClause>
         </SimpleClause>
      </Clause>
   </CompoundClause>
</Clause>
```

Coumpound with one Simple and one Compound

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

```
<Clause>
   <CompoundClause connectivePredicate="And">
      <Clause>
         <SimpleClause leftArgument="Smoker">
            <BooleanClause booleanPredicate="False"/>
         </SimpleClause>
      </Clause>
      <Clause>
         <CompoundClause connectivePredicate="Or">
            <Clause>
               <SimpleClause leftArgument="Age">
                  <RationalClause logicalPredicate="LE">
                     <IntClause>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 8.1.

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 QueryManager. The GetContentRequest specifies a list of Object references for Objects that need to be retrieved. The QueryManager returns the specified content by sending a GetContentResponse message to the RegistryClient interface 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

In the current version of this specification, we address data integrity and source integrity (item 1 in Appendix F.1). We have used a minimalist approach to address the access control concern as in item 2 of Appendix F.1. 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 mechanisms described in this section can be used to ensure that any tampering with the content submitted by a Submitting Organization can be detected. Furthermore, these mechanisms support unambiguous identification of the Responsible Organization for any registry content. The Registry Client has to sign the contents before submission – otherwise the content will be rejected. Note that in the discussions in this section we assume a Submitting Organization to be also the Responsible Organization. Future version of this specification may provide more examples and scenarios where a Submitting Organization and Responsible Organization are different.

9.2.1 Message Payload Signature

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

- Any tampering of the content can be detected.
- 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 Registration Authority communicate over vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML Messaging Service does not specify the generation, validation and packaging of payload signatures. The specification of payload signatures is left upto the application (such as Registry). So the requirements on the payload signatures augment the [ebMS] specification.

Use Case

This Use Case illustrates the use of header and payload signatures (we discuss header signatures later).

- 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 Registration Authority 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.
  - The public key for validating RC1's payload signature in ds:Signature element (using the KeyInfo element as specified in [XMLDSIG]) so RC2 can obtain the public 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 Registration Authority (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 Section 9.2.2.2, "Payload Signature Generation Requirements".
- The second through nth body part must be the content.

The packaging of the payload signature with one payload 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. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a client may use any of these attributes. However, signing using the following Algorithm attribute: http://www.w3.org/2000/09/xmldsig/#dsa-sha1 will allow interoperability with all XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this algorithm.

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 each of the payloads 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. However, when present, the ds:KeyInfo field is subject to the requirements stated in Section 9.4, “KeyDistrbution and KeyInfo element”.

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="http://www.w3.org/TR/2000/09/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. The identity of the Principal can be identified by verifying the message header signature with the certificate of the Principal. The certificate may be in the message itself or provided to the registry through means unspecified in this specification. If not provided in the message, this specification does not specify how the Registry correlates a specific message with a certificate. Authentication of each payload must also be possible by using the signature associated with each payload. Authentication is also required to identify the "privileges" a Principal is authorized ("authorization") to have with respect to specific objects in the Registry.

The Registry must perform authentication on a per message 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 message header signature can only guarantee data integrity and it may be used for Authentication knowing that it is vulnerable to replay types of attacks. True support for authentication requires timestamps or nonce (nonrecurring series of numbers to identify each message) 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 Registration Authority communicate using vanilla SOAP with Attachments. When ebXML MS is used for communication, then the message handler (i.e. [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. When present, it is subject to the requirements stated in Section 9.4, "KeyDistrbution and KeyInfo element".

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 public key corresponding to the signer's public key. The participants may use the KeyInfo field of ds:Signature, or distribute the public keys out-of-band. In this section we consider the case when the public key is sent in the KeyInfo field. The following use cases need to be handled:

- Registration Authority needs the public key of the Registry Client to validate the signature
- Registry Client needs the public key of the Registration Authority to validate the Registry's signature.
- Registry Client RC1 needs the public 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 public 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 public key to a recipient. ds:Keyinfo can be used to pass information such as keys, certificates, names etc. The intended usage of KeyInfo field is to send the X509 Certificate, and subsequently extract the public key from the certificate. Therefore, the KeyInfo field must contain a X509 Certificate as specified in [XMLDSIG], if the KeyInfo field is present.

The following assumptions are also made:

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

Couple of scenarios on the use of KeyInfo field is in Appendix F.8.

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. This specification does not specify how payload encryption is to be done.

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. This implies that the Registry and the client need to 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. Note that we have “ContentOwner” as a role. This role maps to the Submitting Organization in the current version of the specification.

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 Registry must implement the default AccessControlPolicy and associate it with all Objects in the Registry. The following list summarizes the default role-based AccessControlPolicy:

- Anyone can publish content, but needs to be a Registered User
- Anyone can access the content without requiring authentication
- The ContentOwner has access to all methods for Registry Objects created by it.
- 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

The normative definition of the Abstract Registry Service in WSDL is defined at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/services/Registry.wsdl

A.2 Registry Service SOAP Binding

The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/services/SOAPBinding.wsdl

Appendix B ebXML Registry Schema Definitions

B.1 RIM Schema

The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd

B.2 Query Schema

The normative XML Schema definition for the XML query syntax for the registry service interface can be found at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd

B.3 Registry Services Interface Schema

The normative XML Schema definition that defines the XML requests and responses supported by the registry service interfaces in this document can be found at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd

B.4 Examples of Instance Documents

A growing number of non-normative XML instance documents that conform to the normative Schema definitions described earlier may be found at the following location on the web:

http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/

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 required to implement an ebXML Registry Services and clients. 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.

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 7 on page 27 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

The normative Relational Schema definition for SQL queries can be found at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/sql/database.sql

The stored procedures that must be supported by the SQL query feature are defined at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/2.0/sql/storedProcedures.sql

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 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 QueryManager.

[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.2 Index Definition

This section describes how the logical indexes are defined in the SubmittedObject element defined in the Registry Schema. The complete Registry Schema is available via hyperlinks in Appendix B.

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.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.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.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 LifeCycleManager 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 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.

   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?

F.2 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.3 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.4 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.5 Content Submission – Client Responsibility

The Registry client must sign the contents before submission – otherwise the content will be rejected.

F.6 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.7 Content Delete/Deprecate – Client Responsibility

The Registry client must sign the header before submission, for authentication purposes; otherwise, the request will be rejected

F.8 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.

F.9 Using ds:KeyInfo Field

Two typical usage scenarios for ds:KeyInfo are described below.

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. Registration Authority retrieves the certificate from the KeyInfo field in the header signature
5. Registration Authority validates the header signature using the public key from the certificate.
6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the certificate from the KeyInfo field of the payload signature. Note that this step is not an essential one if the onus of validation is that of the eventual user, another Registry Client, of the content.

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 RC1 is passed to the Registry in the KeyInfo field of the payload signature. This step is required in addition to step 2 because when RC2 retrieves content, it should see RC1's signature with the payload.
4. RC2 retrieves content from the Registry.
5. Registration Authority signs the SOAP envelope using its private key. Registration Authority sends RC1's content and the RC1's signature (signed by RC1).
6. Registration Authority need not send its certificate in the KeyInfo field sinceRC2 is assumed to have obtained the Registration Authority's certificate out of band and installed it in its local key store.
7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the Registration Authority's signature.
8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates the signature on the payload.

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*

The character set used by a locale specific String (LocalizedString) is defined by the charset attribute. It is highly recommended to use UTF-8 or UTF-16 for maximuminter-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 Registry Profile

Every registry must support exactly one Registry Profile. The Registry Profile is an XML document that describes the capabilities of the registry. The profile document must conform to the RegistryProfile element as described in the Registry Services Interface schema defined in Appendix B. The registry must make the RegistryProfile accessible over HTTP protocol via a URL. The URL must conform to the pattern:

http://<base url>/registryProfile

10 References

[Bra97] Keywords for use in RFCs to Indicate Requirement Levels.

[ebRIM] ebXML Registry Information Model version 2.0
http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf

[ebRIM Schema] ebXML Registry Information Model Schema
http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd

[ebBPSS] ebXML Business Process Specification Schema
http://www.ebxml.org/specs

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

[ebMS] ebXML Messaging Service Specification, Version 1.0
http://www.ebxml.org/specs/

[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 2119] IETF (Internet Engineering Task Force). RFC 2119

[RFC 2130] IETF (Internet Engineering Task Force). RFC 2130

[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 2828] IETF (Internet Engineering Task Force). RFC 2828:
Internet Security Glossary, ed. R. Shirey. May 2000.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.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/

11 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.

12 Contact Information

Team Leader

Name:	Lisa J. Carnahan
Company:	National Institute of Standards and Technology
Street:	100 Bureau Drive, Stop 8970
City, State, Postal Code:	Gaithersburg, Md. 20899
Country:	USA
Phone:	301-975-3362
Email:	lisa.carnahan@nist.gov

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

Technical Editor

Name:	Farrukh S. Najmi
Company:	Sun Microsystems
Street:	1 Network Dr., MS BUR02-302
City, State, Postal Code:	Burlington, MA 01803-0902
Country:	USA
Phone:	781-442-0703
Email:	najmi@east.sun.com

13 Copyright Statement

Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification, can be obtained from the OASIS Executive Director.

OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to implement this specification. Please address the information to the OASIS Executive Director.