ebXML Registry Services

ebXML Registry Project Team

Working Draft 3/19/2001

This version: Version 0.88

1 Status of this Document

This document specifies an ebXML DRAFT STANDARD for the eBusiness community.

Distribution of this document is unlimited.

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

This version:

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.88.pdf

Latest version:


Previous version:

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.87.pdf
2 ebXML participants

The authors wish to acknowledge the support of the members of the Registry Project Team who contributed ideas to this specification by the group’s discussion e-mail list, on conference calls and during face-to-face meetings.

Lisa Carnahan – NIST
Joe Dalman - Tie
Philippe DeSmedt - Viquity
Sally Fuger – AIAG
Len Gallagher - NIST
Steve Hanna - Sun Microsystems
Scott Hinkelman - IBM
Michael Kass, NIST
Jong.L Kim – Innodigital
Sangwon Lim, Korea Institute for Electronic Commerce
Bob Miller - GXS
Kunio Mizoguchi - Electronic Commerce Promotion Council of Japan
Dale Moberg – Sterling Commerce
Ron Monzillo – Sun Microsystems
JP Morgenthal – eThink Systems, Inc.
Joel Munter - Intel
Farrukh Najmi - Sun Microsystems
Scott Nieman - Norstan Consulting
Frank Olken – Lawrence Berkeley National Laboratory
Michael Park - eSum Technologies
Bruce Peat - eProcess Solutions
Mike Rowley – Excelon Corporation
Waqar Sadiq - Vitria
Krishna Sankar – Cisco Systems Inc.
Kim Tae Soo - Government of Korea
Nikola Stojanovic - Encoda Systems, Inc.
David Webber – XML Global
Yutaka Yoshida - Sun Microsystems
Prasad Yendluri - webmethods
Peter Z. Zhoo - Knowledge For the new Millennium
# Table of Contents

1. Status of this Document ...................................................................................... 1
2. ebXML participants ............................................................................................... 2
3. Table of Contents........................................................................................................... 3
4. Table of Tables ............................................................................................................... 7

3. Introduction............................................................................................................. 8
   3.1 Summary of Contents of Document ............................................................. 8
   3.2 General Conventions...................................................................................... 8
   3.3 Audience........................................................................................................... 8
   3.4 Related Documents ........................................................................................ 8

4. Design Objectives .................................................................................................. 9
   4.1 Goals ................................................................................................................. 9
   4.2 Caveats and Assumptions ............................................................................. 9

5. System Overview ................................................................................................... 9
   5.1 What The ebXML Registry Does .................................................................. 9
   5.2 How The ebXML Registry Works .............................................................. 10
       5.2.1 Schema Documents Are Submitted .................................................... 10
       5.2.2 Business Process Documents Are Submitted ....................................... 10
       5.2.3 Seller’s Collaboration Protocol Profile Is Submitted ............................... 10
   5.3 Where the Registry Services May Be Implemented ....................................... 11
   5.4 Implementation Conformance ........................................................................ 11
       5.4.1 Conformance as an ebXML Registry ................................................... 11
       5.4.2 Conformance as an ebXML Registry Client ........................................... 11

6. Registry Architecture.......................................................................................... 12
   6.1 Implicit CPA Between Clients And Registry ............................................ 13
   6.2 Client To Registry Communication Bootstrapping ................................... 14
   6.3 Interfaces Exposed By The Registry .......................................................... 15
       6.3.1 Interface RegistryService ................................................................... 15
       6.3.2 Interface ObjectManager ................................................................... 16
       6.3.3 Interface ObjectQueryManager ........................................................... 16
   6.4 Interfaces Exposed By Registry Clients ..................................................... 18
       6.4.1 Interface RegistryClient ..................................................................... 18
       6.4.2 Interface ObjectManagerClient ......................................................... 18
       6.4.3 Interface ObjectQueryManagerClient ............................................... 20

7. Object Management Service ............................................................................. 20
   7.1 Life Cycle of a Registry Entry ...................................................................... 21
<table>
<thead>
<tr>
<th>Page</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>7.2</td>
</tr>
<tr>
<td>101</td>
<td>7.3</td>
</tr>
<tr>
<td>102</td>
<td>7.3.1</td>
</tr>
<tr>
<td>103</td>
<td>7.3.2</td>
</tr>
<tr>
<td>104</td>
<td>7.3.3</td>
</tr>
<tr>
<td>105</td>
<td>7.4</td>
</tr>
<tr>
<td>106</td>
<td>7.5</td>
</tr>
<tr>
<td>107</td>
<td>7.6</td>
</tr>
<tr>
<td>108</td>
<td>7.7</td>
</tr>
<tr>
<td>109</td>
<td>7.8</td>
</tr>
<tr>
<td>110</td>
<td>7.8.1</td>
</tr>
<tr>
<td>111</td>
<td>7.8.2</td>
</tr>
<tr>
<td>112</td>
<td>8</td>
</tr>
<tr>
<td>113</td>
<td>8.1</td>
</tr>
<tr>
<td>114</td>
<td>8.1.1</td>
</tr>
<tr>
<td>115</td>
<td>8.1.2</td>
</tr>
<tr>
<td>116</td>
<td>8.1.3</td>
</tr>
<tr>
<td>117</td>
<td>8.1.3.1</td>
</tr>
<tr>
<td>118</td>
<td>8.2</td>
</tr>
<tr>
<td>119</td>
<td>8.2.1</td>
</tr>
<tr>
<td>120</td>
<td>8.2.2</td>
</tr>
<tr>
<td>121</td>
<td>8.2.3</td>
</tr>
<tr>
<td>122</td>
<td>8.2.4</td>
</tr>
<tr>
<td>123</td>
<td>8.2.5</td>
</tr>
<tr>
<td>124</td>
<td>8.2.6</td>
</tr>
<tr>
<td>125</td>
<td>8.2.7</td>
</tr>
<tr>
<td>126</td>
<td>8.2.8</td>
</tr>
<tr>
<td>127</td>
<td>8.2.9</td>
</tr>
<tr>
<td>128</td>
<td>8.2.10</td>
</tr>
<tr>
<td>129</td>
<td>8.3</td>
</tr>
<tr>
<td>130</td>
<td>8.3.1</td>
</tr>
<tr>
<td>131</td>
<td>8.3.1.1</td>
</tr>
<tr>
<td>132</td>
<td>8.3.1.2</td>
</tr>
<tr>
<td>133</td>
<td>8.3.1.3</td>
</tr>
<tr>
<td>134</td>
<td>8.3.1.4</td>
</tr>
<tr>
<td>135</td>
<td>8.3.1.5</td>
</tr>
<tr>
<td>136</td>
<td>8.3.1.6</td>
</tr>
<tr>
<td>137</td>
<td>8.3.2</td>
</tr>
<tr>
<td>138</td>
<td>8.3.3</td>
</tr>
<tr>
<td>139</td>
<td>8.3.4</td>
</tr>
<tr>
<td>140</td>
<td>8.3.5</td>
</tr>
<tr>
<td>141</td>
<td>8.3.6</td>
</tr>
<tr>
<td>142</td>
<td>8.3.6.1</td>
</tr>
<tr>
<td>143</td>
<td>8.3.6.2</td>
</tr>
</tbody>
</table>
Table of Tables

Table 1: Terminology Mapping Table .................................................................107
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 [RIM], provides information on the types of metadata that is stored in the Registry as well as the relationships among the various metadata classes.

3.2 General Conventions
- UML diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific implementation or methodology requirements.
- The term "repository item" is used to refer to actual Registry content (e.g. a DTD, as opposed to metadata about the DTD).
- The term "RegistryEntry" is used to refer to an object that provides metadata about a content instance (repository item).

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in RFC 2119 [Bra97].

3.3 Audience
The target audience for this specification is the community of software developers who are:
- Implementers of ebXML Registry Services
- Implementers of ebXML Registry Clients

3.4 Related Documents
The following specifications provide some background and related information to the reader:
- ebXML Registry Business Domain Model [BDM] - defines requirements for ebXML Registry Services
- ebXML Registry Information Model [RIM] - specifies the information model for the ebXML Registry
- ebXML Messaging Service Specification [MS]
4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

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

4.2 Caveats and Assumptions

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

It is assumed that:

1. All interactions between the clients of the ebXML Registry and the ebXML Registry will be conducted using ebXML Messaging Service.
2. All access to the Registry content is exposed via the interfaces defined for the Registry Services.
3. 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 envisioned in [BDM]. 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 their own private Registry).

5.2.1 Schema Documents Are Submitted

A third party such as an industry consortium or standards group can submit the necessary schema documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the Object Manager 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, can submit the necessary business process documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the Registry using the Object Manager 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 [CPA] 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 [CPA] with the seller using the seller’s CPP and their own CPP as input. The buyer proposes a partnership to the seller using the unilateral CPA. The seller accepts the proposed CPA and the partnership is established.

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

5.3 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.

5.4 Implementation Conformance

An implementation may claim conformance as an ebXML Registry, an ebXML Registry Client or both.

5.4.1 Conformance as an ebXML Registry

An implementation claims conformance to this specification if it meets the following conditions:

1. Conforms to the ebXML Registry Information Model [RIM].
2. Supports the syntax and semantics of the Registry Interfaces and Security Model.
3. Supports the defined ebXML Error Message DTD.
4. Supports the defined ebXML Registry DTD.
5. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

5.4.2 Conformance as an ebXML Registry Client

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

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

The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry Clients. The Registry Client interfaces may be local to the registry or local to the user. Figure 1 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 local to the Registry from the user’s perspective.

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 perspective. 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 an 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 current business needs.
Clients communicate with the Registry using the ebXML Messaging Service in the same manner as any two ebXML applications communicating with each other. Future versions of this specification may extend the Registry architecture to support distributed Registries.

This specification defines the interaction between a Registry client and the Registry. Although these interaction protocols are specific to the Registry, they are identical in nature to the interactions between two parties conducting B2B message communication using the ebXML Messaging Service as defined by [MS] and [CPA].

As such, these Registry specific interaction protocols are a special case of interactions between two parties using the ebXML Messaging Service.

6.1 Implicit CPA Between Clients And Registry

ebXML defines that a Collaboration Protocol Agreement [CPA] must exist between two parties in order for them to engage in B2B interactions.
Similarly, this specification defines a CPA between a Registry client and the Registry. Typical B2B interactions in ebXML require an explicit CPA to be negotiated between parties. However, the CPA between clients and the Registry is an implicit CPA that describes the interfaces that the Registry and the client expose to each other for Registry specific interactions. These interfaces are described in Figure 2 and subsequent sections.

6.2 Client To Registry Communication Bootstrapping

Because 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 Registry, although it could be some other type of address such as 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.
6.3 Interfaces Exposed By The Registry

The ebXML Registry is shown to implement the following interfaces as its services (Registry Services).

6.3.1 Interface RegistryService

This is the principal interface implemented by the Registry. It provides the methods that are used by the client to discover service specific interfaces implemented by the Registry.
Method Summary

<table>
<thead>
<tr>
<th><strong>ObjectManager</strong></th>
<th><strong>getObjectName()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the ObjectManager interface implemented by the Registry service.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ObjectQueryManager</strong></th>
<th><strong>getObjectName()</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns the ObjectQueryManager interface implemented by the Registry service.</td>
<td></td>
</tr>
</tbody>
</table>

6.3.2 Interface ObjectManager

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, classify and associate objects and to deprecate and remove objects.

Method Summary

<table>
<thead>
<tr>
<th><strong>Void approveObjects(ApproveObjectsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Approves one or more previously submitted objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Void deprecateObjects(DeprecateObjectsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Deprecates one or more previously submitted objects.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Void removeObject(RemoveObjectsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Removes one or more previously submitted objects from the Registry.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>void submitObjects(SubmitObjectsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Submits one or more objects and possibly metadata related to object such as Associations and Classifications.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>void addSlots(AddSlotsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Add slots to one or more registry entries.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>void removeSlots(RemoveSlotsRequest req)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove specified slots from one or more registry entries.</td>
</tr>
</tbody>
</table>

6.3.3 Interface ObjectQueryManager
This is the interface exposed by the Registry that implements the Object 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 and metadata.

### Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>GetClassificationTreeResponse</code></td>
<td><code>getClassificationTree(GetClassificationTreeRequest req)</code></td>
</tr>
<tr>
<td></td>
<td>Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.</td>
</tr>
<tr>
<td><code>void getClassificationTreeAsync(GetClassificationTreeRequest req)</code></td>
<td>Asynchronous version of <code>getClassificationTree</code>.</td>
</tr>
<tr>
<td><code>GetClassifiedObjectsResponse</code></td>
<td><code>getClassifiedObjects(GetClassifiedObjectsRequest req)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a collection of references to RegistryEntries classified under specified ClassificationItem.</td>
</tr>
<tr>
<td><code>void getClassifiedObjectsAsync(GetClassifiedObjectsRequest req)</code></td>
<td>Asynchronous version of <code>getClassifiedObjects</code>.</td>
</tr>
<tr>
<td><code>GetContentResponse</code></td>
<td><code>getContent()</code></td>
</tr>
<tr>
<td></td>
<td>Returns the specified content. The response includes all the content specified in the request as additional payloads within the response message.</td>
</tr>
<tr>
<td><code>void getContentAsync()</code></td>
<td>Async version of <code>getContent</code>.</td>
</tr>
<tr>
<td><code>GetRootClassificationNodesResponse</code></td>
<td><code>getRootClassificationNodes(GetRootClassificationNodesRequest req)</code></td>
</tr>
<tr>
<td></td>
<td>Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.</td>
</tr>
<tr>
<td><code>void getRootClassificationNodesAsync(GetRootClassificationNodesRequest req)</code></td>
<td>Async version of <code>getRootClassificationNodes</code>.</td>
</tr>
</tbody>
</table>
6.4 Interfaces Exposed By Registry Clients

An ebXML Registry client is shown to implement the following interfaces.

6.4.1 Interface RegistryClient

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 discover service specific interfaces implemented by the client.

Method Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectManagerClient</td>
<td>getObjectManagerClient()</td>
<td>Returns the ObjectManagerClient interface implemented by the client.</td>
</tr>
<tr>
<td>ObjectQueryManagerClient</td>
<td>getObjectQueryManagerClient()</td>
<td>Returns the ObjectQueryManagerClient interface implemented by the client.</td>
</tr>
</tbody>
</table>

6.4.2 Interface ObjectManagerClient

This is the client callback interface for the ObjectManager service of the Registry. The ObjectManager invokes its methods to notify the client about the results of a previously submitted request from the client to the ObjectManager service.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void addSlotsAccepted()</td>
<td>Notifies client that a previously submitted AddSlotsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addSlotsError(error)</td>
<td>Notifies client that a previously submitted AddSlotsRequest was not accepted by the Registry due to an error.</td>
</tr>
<tr>
<td>approveObjectsAccepted(resp)</td>
<td>Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>approveObjectsError(error)</td>
<td>Notifies client that a previously submitted ApproveObjectsRequest was not accepted by the Registry due to an error.</td>
</tr>
<tr>
<td>deprecateObjectsAccepted(resp)</td>
<td>Notifies client that a previously submitted DeprecateObjectsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>deprecateObjectsError(error)</td>
<td>Notifies client that a previously submitted DeprecateObjectsRequest was not accepted by the Registry due to an error.</td>
</tr>
<tr>
<td>removeObjectsAccepted(resp)</td>
<td>Notifies client that a previously submitted RemoveObjectsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>removeSlotsAccepted(resp)</td>
<td>Notifies client that a previously submitted RemoveSlotsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>removeObjectsError(error)</td>
<td>Notifies client that a previously submitted RemoveObjectsRequest was not accepted by the Registry due to an error.</td>
</tr>
<tr>
<td>removeSlotsError(error)</td>
<td>Notifies client that a previously submitted RemoveSlotsRequest was not accepted by the Registry due to an error.</td>
</tr>
<tr>
<td>submitObjectsAccepted(resp)</td>
<td>Notifies client that a previously submitted SubmitObjectsRequest was accepted by the Registry.</td>
</tr>
<tr>
<td>submitObjectsError(error)</td>
<td>Notifies client that a previously submitted SubmitObjectsRequest was not accepted by the Registry due to an error.</td>
</tr>
</tbody>
</table>
6.4.3 Interface ObjectQueryManagerClient

This is the callback interface for the ObjectQueryManager service of the Registry. The ObjectQueryManager invokes its methods to notify the client about the results of a previously submitted query request from client to the ObjectQueryManager service.

### Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getClassificationTreeAsyncResponse</td>
<td>Async response for getClassificationTreeAsync request.</td>
</tr>
<tr>
<td>getClassifiedObjectsAsyncResponse</td>
<td>Async response for getClassifiedObjectsAsync request.</td>
</tr>
<tr>
<td>getContentAsyncResponse</td>
<td>Async response for getContent request.</td>
</tr>
<tr>
<td>getRootClassificationNodesAsyncResponse</td>
<td>Async response for getRootClassificationNodesAsync request.</td>
</tr>
<tr>
<td>submitAdhocQueryAsyncResponse</td>
<td>Async response for submitAdhocQueryAsync request.</td>
</tr>
</tbody>
</table>

7 Object Management Service

This section defines the Object Management service of the Registry. The Object 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 Object Management Service can be used with all types of repository items as well as the metadata objects specified in [RIM] such as Classification and Association.

In the current version of this specification, any client may submit content as long as the content is digitally signed by a certificate issued by a Certificate Authority recognized by this registry. Submitting Organizations do not have to register prior to submitting content.
7.1 Life Cycle of a Registry Entry

The main purpose of the Object Management service is to manage the life cycle of repository items in the Registry.

Figure 3 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.

7.2 Object Attributes

A repository item is associated with a set of standard metadata defined as attributes of the Object class and its sub-classes as described in [RIM]. These attributes reside outside of the actual repository item and catalog descriptive information about the repository item. XML DTD elements called ExtrinsicObject and IntrinsicObject (See Appendix A.2 for details.) are defined that encapsulates all object metadata attributes defined in [RIM] as attributes of the DTD elements.
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 in the repository using the ObjectManager on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix B.

![Figure 4: Submit Objects Sequence Diagram]

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

The SubmitObjectRequest message includes 1 or more SubmittedObject elements.

Each SubmittedObject element specifies an ExtrinsicObject along with any Classifications, Associations, ExternalLinks, or Packages related to the object being submitted.

An ExtrinsicObject element provides required metadata about the content being submitted to the Registry as defined by [RIM]. Note that these standard ExtrinsicObject attributes are separate from the repository item itself, thus allowing the ebXML Registry to catalog arbitrary objects. In addition each SubmittedObject in the request may optionally specify any number of Classifications, Associations and ExternalLinks for the SubmittedObject.

7.3.1 Universally Unique ID Generation

As specified by [RIM], all objects in the registry have a unique id. This id is usually generated by the registry. The id attribute for various submitted objects may optionally be supplied by the client. 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 honor 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 send an ebXMLError in response with an InvalidIdError message.

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

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 Sample SubmitObjectsRequest

The following example shows several different use cases in a single SubmitObjectsRequest. It does not show the complete ebXML Message with the message header and additional payloads in the message for the repository items. A SubmitObjectsRequest includes a RegistryEntryList 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
<xml version="1.0" encoding="UTF-8">
<!DOCTYPE SubmitObjectsRequest SYSTEM "file:///home/najml/Registry.dtd">

<SubmitObjectsRequest>
  <RegistryEntryList>
    <!-- The following 3 objects package specified ExtrinsicObject in specified Package, where both the Package and the ExtrinsicObject are being submitted -->
    <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
    <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1" objectType = "CPP" name = "Widget Profile" description = "ACME's profile for selling widgets"/>
    <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>

    <!-- The following 3 objects package specified ExtrinsicObject in specified Package, where the Package is being submitted and the ExtrinsicObject is already in registry -->
    <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
    <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
    <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 Package, where the Package and the ExtrinsicObject are already in registry -->
    <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
    <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
    <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 -->
    <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
    <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP" name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
    <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 -->
    <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
    <Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType = "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>

    <!-- The following 2 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 -->
    <ExternalIdentifier id = "acmeDUNSId" name = "DUNS" description = "DUNS ID for ACME"/>
value = "13456789012"/>

<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
-->
<ClassificationNode id = "geographyNode" name = "Geography"
description = "The Geography scheme example from Registry Services Spec" />
<ClassificationNode id = "asiaNode" name = "Asia"
description = "The Asia node under the Geography node" parent="geographyNode" />
<ClassificationNode id = "japanNode" name = "Japan"
description = "The Japan node under the Asia node" parent="asiaNode" />
<ClassificationNode id = "koreaNode" name = "Korea"
description = "The Korea node under the Asia node" parent="koreaNode" />
<ClassificationNode id = "europeNode" name = "Europe"
description = "The Europe node under the Geography node" parent="geographyNode" />
<ClassificationNode id = "germanyNode" name = "Germany"
description = "The Germany node under the Asia node" parent="europeNode" />
<ClassificationNode id = "northAmericaNode" name = "North America"
description = "The North America node under the Geography node" parent="geographyNode" />
<ClassificationNode id = "usNode" name = "US"
description = "The US node under the Asia node" parent="asiaNode" />

<!--
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
-->
<ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
<ClassificationNode id = "automotiveNode" name = "Automotive"
description = "The Automotive sub-tree under Industry scheme"
parent = "urn:uuid:d2345678-1234-1234-123456789012" />
<ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
description = "The Parts Supplier node under the Automotive node",
parent="automotiveNode" />
<ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
description = "The Engine Supplier node under the Automotive node",
parent="automotiveNode" />

<!--
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.
-->
<Classification id = "japanClassification"
description = "Classifies object by /Geography/Asia/Japan node"
classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
classificationNode="japanNode" />
<Classification id = "classificationUsingExistingNode"
description = "Classifies object using a node in the registry"
classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
<ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />

</RegistryEntryList>
</SubmitObjectsRequest>
7.4 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 Object Manager. Slots provide a dynamic mechanism for extending registry entries as defined by [RIM].

![Add Slots Sequence Diagram](image)

Figure 5: Add Slots Sequence Diagram

7.5 The Remove 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 Object Manager.

![Remove Slots Sequence Diagram](image)

Figure 6: Remove Slots Sequence Diagram
7.6 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 Object Manager. 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).

![Approve Objects Sequence Diagram](image)

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

7.7 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 Object Manager. 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.
For details on the schema for the business documents shown in this process refer to Appendix A.2.

7.8 The Remove Objects Protocol

This section describes the protocol of the Registry Service that allows a client to remove one or more Registry Entries and/or repository items using the Object Manager.

The RemoveObjectsRequest message is sent by a client to remove Registry Entries 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.

7.8.1 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.

7.8.2 Deletion Scope DeleteAll

This deletionScope specifies that the request should delete both the RegistryEntry and the repository item for the specified registry entries. Only if all references (e.g. Associations, Classifications, ExternalLinks) to a RegistryEntry have been removed, can that RegistryEntry then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to remove a RegistryEntry while it still has references results in an InvalidRequestError that is returned within an ebXMLError message sent to the ObjectManagerClient by the ObjectManager.
The remove object protocol is expressed in UML notation as described in Appendix B.

**Figure 9: Remove Objects Sequence Diagram**

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

### 8 Object Query Management Service

This section describes the capabilities of the Registry Service that allow a client (ObjectQueryManagerClient) to search for or query RegistryEntries in the ebXML Registry using the ObjectQueryManager interface of the Registry.

The Registry supports multiple query capabilities. These include the following:

1. Browse and Drill Down Query
2. Filtered Query
3. SQL Query

The browse and drill down query [8.1] and the filtered query mechanism [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 they must conform to this document. As such it is 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.
Any errors in the query request messages are indicated in the corresponding query response message. Note that for each query request/response there is both a synchronous and asynchronous version of the interaction.

8.1 Browse and Drill Down Query Support

The browse and drill down query style is completely supported by a set of interaction protocols between the ObjectQueryManagerClient and the ObjectQueryManager as described next.

8.1.1 Get Root Classification Nodes Request

An ObjectQueryManagerClient sends this request to get a list of root ClassificationNodes defined in the repository. Root classification nodes are defined as nodes that have no parent. Note that it is possible to specify a namePattern attribute that can filter on the name attribute of the root ClassificationNodes. The namePattern must be specified using a wildcard pattern defined by SQL-92 LIKE clause as defined by [SQL].

Figure 10: Get Root Classification Nodes Sequence Diagram
For details on the schema for the business documents shown in this process refer to Appendix A.2.

### 8.1.2 Get Classification Tree Request

An ObjectQueryManagerClient sends this request to get the ClassificationNode sub-tree defined in the repository under the ClassificationNodes specified in the request. Note that a GetClassificationTreeRequest can specify an integer attribute called *depth* to get the sub-tree up to the specified depth. If depth is the default value of 1, then only the immediate children of the specified ClassificationNodeList are returned. If depth is 0 or a negative number then the entire sub-tree is retrieved.

For details on the schema for the business documents shown in this process refer to Appendix A.2.
8.1.3 Get Classified Objects Request

An ObjectQueryManagerClient sends this request to get a list of RegistryEntries that are classified by all of the specified ClassificationNodes (or any of their descendants), as specified by the ObjectRefList in the request.

It is possible to get RegistryEntries based on matches with multiple classifications. Note that specifying a ClassificationNode is implicitly specifying a logical OR with all descendants of the specified ClassificationNode.

When a GetClassifiedObjectsRequest is sent to the ObjectQueryManager it should return Objects that are:

1. Either directly classified by the specified ClassificationNode
2. Or are directly classified by a descendant of the specified ClassificationNode

8.1.3.1 Get Classified Objects Request Example

Let us say a classification tree has the structure shown in Figure 14:

![Figure 14: A Sample Geography Classification](image)

- If the Geography node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by Geography or North America or US or Asia or Japan or Korea or Europe or Germany.

- If the Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by Asia or Japan or Korea.

- If the Japan and Korea nodes are specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by both Japan and Korea.

- If the North America and Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all RegistryEntries that are directly classified by (North America or US) and (Asia or Japan or Korea).
8.2 Filter Query Support

The simple XML FilterQuery specified below shall be supported by every Registry implementation.

The FilterQuery syntax is tied to the structures defined in the Registry Information Model [RIM] and is not intended to be extensible. If new structures are added to the RIM, then the FilterQuery syntax and semantics can be extended at the same time. Each query alternative requires a binding to the structures defined by RIM.

The Registry will hold a self-describing profile that identifies all supported Query options. The RegistryProfile DTD is defined in appendix A.2. This profile can be retrieved as defined by section 8.4.1.

An XML FilterQuery element provides alternatives to query selected classes from the RIM. Each choice of a class pre-determines a virtual XML document that can be queried as a tree. The RIM Binding paragraphs in Sections 8.2.2 through 8.2.6 below identify the virtual hierarchy for each query alternative. The Semantic Rules for each query alternative specify the effect of that binding on query semantics.

Each FilterQuery alternative depends upon one or more registry filters, where a registry filter is a restricted predicate clause over the attributes of a single class. The supported registry filters are specified in Section 8.2.9 and the supported predicate clauses are defined in Section 8.2.10.

The GetRegistryEntry and GetRepositoryItem services defined below provide a way to structure an XML document as an expansion of the result of a RegistryEntryQuery. The GetRegistryEntry specified in Section 8.2.7 allows one to specify what metadata one wants returned with each registry entry identified in the result of a RegistryEntryQuery. The GetRepositoryItem specified in section 8.2.8 allows one to specify what repository items one wants returned based on their relationships to the registry entries identified in the result of a RegistryEntryQuery.

A client submits a query to the ObjectQueryManager by sending an Adhoc QueryRequest. The ObjectQueryManager sends an AdhocQueryResponse back to the client. The request and the response for each query alternative, and the sequence diagrams for AdhocQueryRequest and AdhocQueryResponse, are all specified in section 8.3.12 below. A FilterQuery is one of the query options in an AdhocQueryRequest and a FilterQueryResult is the response that is to be returned as part of the AdhocQueryResponse.
8.2.1 FilterQuery

Purpose

To identify a set of registry instances from a specific registry class. Each alternative assumes a specific binding to RIM. The query result for each query alternative is a set of references to instances of the root class specified by the binding. The StatusResult is a success indication or a collection of warnings and/or exceptions.

Definition

```xml
<!ELEMENT FilterQuery (    RegistryEntryQuery | AuditableEventQuery | ClassificationNodeQuery | RegistryPackageQuery | OrganizationQuery )>

<!ELEMENT FilterQueryResult ( RegistryEntryQueryResult | AuditableEventQueryResult | ClassificationNodeQueryResult | RegistryPackageQueryResult | OrganizationQueryResult )>

<!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>

<!ELEMENT RegistryEntryView EMPTY >

<!ATTLIST RegistryEntryView
    objectURN CDATA #REQUIRED
    contentURL CDATA #IMPLIED
    objectID CDATA #IMPLIED >

<!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>

<!ELEMENT AuditableEventView EMPTY >

<!ATTLIST AuditableEventView
    objectID CDATA #REQUIRED
    timestamp CDATA #REQUIRED >

<!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>

<!ELEMENT ClassificationNodeView EMPTY >

<!ATTLIST ClassificationNodeView
    objectURN CDATA #REQUIRED
    contentURL CDATA #IMPLIED
    objectID CDATA #IMPLIED >

<!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>

<!ELEMENT RegistryPackageView EMPTY >

<!ATTLIST RegistryPackageView
    objectURN CDATA #REQUIRED
    contentURL CDATA #IMPLIED
    objectID CDATA #IMPLIED >
```
<!ELEMENT RegistryPackageView EMPTY >
<!ATTLIST RegistryPackageView
  objectURN CDATA #REQUIRED
  contentURL CDATA #IMPLIED
  objectID CDATA #IMPLIED >

<!ELEMENT OrganizationQueryResult ( OrganizationView* )>
<!ELEMENT OrganizationView EMPTY >
<!ATTLIST OrganizationView
  orgURN CDATA #REQUIRED
  contactURL CDATA #IMPLIED
  objectID CDATA #IMPLIED >

<!ELEMENT StatusResult ( Success | ( Exception | Warning )+ )>
<!ELEMENT Success EMPTY >
<!ELEMENT Exception ( #PCDATA )>
<!ATTLIST Exception
  code CDATA #REQUIRED >
<!ELEMENT Warning ( #PCDATA )>
<!ATTLIST Warning
  code CDATA #REQUIRED >

Semantic Rules
1. The semantic rules for each FilterQuery alternative are specified in
   subsequent subsections.
2. Each FilterQueryResult is a set of XML reference 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 as follows:
   a) objectID is the value of the ID attribute of the Object class,
   b) objectURN and orgURN are URN values derived from the object ID,
   c) contentURL is a URL value derived from the contentURI attribute of the
      RegistryEntry class,
   d) timestamp is a literal value to represent the value of the timestamp
      attribute of the AuditableEvent class.
3. An Exception indicates that The FilterQuery was not successful, so the
   FilterQueryResult is empty. A warning indicates that the FilterQuery was
   successful, so the FilterQueryResult is accurate, but the warning may give
   additional information back to the user.
4. If any exception or warning results, then it is returned as the appropriate
   alternative of the StatusResult element. NOTE: This StatusResult may need
   to be modified to fit more closely with the ebXML TRP specification.
8.2.2 RegistryEntryQuery

**Purpose**

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

**RIM Binding**

**Definition**

```xml
<!ELEMENT RegistryEntryQuery
  ( RegistryEntryFilter?,
    AsSourceAssociation*,
    AsTargetAssociation*,
    RegistryEntryClassification*,
    SubmittingOrgFilter?,
    ResponsibleOrgFilter?,
    ExternalLinkFilter*,
    RegistryEntryAuditableEvent* )>

<!ELEMENT AsSourceAssociation
  ( AssociationFilter?,
    RegistryEntryFilter? )>

<!ELEMENT AsTargetAssociation
  ( AssociationFilter?,
    RegistryEntryFilter? )>

<!ELEMENT RegistryEntryClassification
  ( Association?,
    Classification Node?,
    Contact?,
    Contact?,
    Organization?,
    Organization?,
    AuditEvent, User, Organization?,
    Registry Entry )>
```

Copyright © ebXML 2000 & 2001. All Rights Reserved.
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 a RegistryEntryFilter is not specified, or if RE is empty, then continue below; otherwise, let x be a registry entry in RE. If x does not satisfy the RegistryEntryFilter as defined in section 8.2.9, then remove x from RE.

   b) If an AsSourceAssociation element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not the source object of some Association instance, then remove x from RE; otherwise, treat each AsSourceAssociation element separately as follows:

      If no AssociationFilter is specified within AsSourceAssociation, 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 RE. If no RegistryEntryFilter is specified within AsSourceAssociation, then let RET be the set of all RegistryEntry instances that are the target object of some element of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the RegistryEntryFilter and are the target object of some element of AF. If RET is empty, then remove x from RE.

   c) If an AsTargetAssociation element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not the target object of some Association instance, then remove x from RE; otherwise, treat each AsTargetAssociation element separately as follows:
If no AssociationFilter is specified within AsTargetAssociation, 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 RE. If no RegistryEntryFilter is specified within AsTargetAssociation, then let RES be the set of all RegistryEntry instances that are the source object of some element of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the RegistryEntryFilter and are the source object of some element of AF. If RES is empty, then remove x from RE.

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

If no ClassificationFilter is specified within the RegistryEntryClassification, then let CL be the set of all Classification instances that have x as a source object; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the source object. If CL is empty, then remove x from RE. If no ClassificationNodeFilter is specified within RegistryEntryClassification, then let CN be the set of all ClassificationNode instances that are the target object of some element of CL; otherwise, let CN be the set of RegistryEntry instances that satisfy the ClassificationNodeFilter and are the target object of some element of CL. If CN is empty, then remove x from RE.

e) If a SubmittingOrgFilter element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x does not have a submitting organization, then remove x from RE. If no OrganizationFilter is specified within SubmittingOrgFilter, then let SO be the set of all Organization instances that are the submitting organization for x; otherwise, let SO be the set of Organization instances that satisfy the OrganizationFilter and are the submitting organization for x. If SO is empty, then remove x from RE. If no ContactFilter is specified within SubmittingOrgFilter, then let CT be the set of all Contact instances that are the contacts for some element of SO; otherwise, let CT be the set of Contact instances that satisfy the ContactFilter and are the contacts for some element of SO. If CT is empty, then remove x from RE.
f) If a ResponsibleOrgFilter element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x does not have a responsible organization, then remove x from RE. If no OrganizationFilter is specified within ResponsibleOrgFilter, then let RO be the set of all Organization instances that are the responsible organization for x; otherwise, let RO be the set of Organization instances that satisfy the OrganizationFilter and are the responsible organization for x. If RO is empty, then remove x from RE. If no ContactFilter is specified within SubmittingOrgFilter, then let CT be the set of all Contact instances that are the contacts for some element of RO; otherwise, let CT be the set of Contact instances that satisfy the ContactFilter and are the contacts for some element of RO. If CT is empty, then remove x from RE.

g) If an ExternalLinkFilter element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some ExternalLink instance, then remove x from RE; otherwise, treat each ExternalLinkFilter element separately as follows:

Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are linked to x. If EL is empty, then remove x from RE.

h) If a RegistryEntryAuditableEvent element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some AuditableEvent instance, then remove x from RE; otherwise, treat each RegistryEntryAuditableEvent element separately as follows:

If an AuditableEventsFilter is not specified within RegistryEntryAuditableEvent, then let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the set of AuditableEvent instances that satisfy the AuditableEventFilter and are auditable events for x. If AE is empty, then remove x from RE. If an UserFilter is not specified within RegistryEntryAuditableEvent, then let AI be the set of all User instances linked to an element of AE; otherwise, let AI be the set of User instances that satisfy the UserFilter and are linked to an element of AE. If AI is empty, then remove x from RE. If an OrganizationFilter is not specified within RegistryEntryAuditableEvent, then let OG be the set of all Organization instances that are linked to an element of AI; otherwise, let OG be the set of Organization instances that satisfy the OrganizationFilter and are linked to an element of AI. If OG is empty, then remove x from RE.

2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return RE as the result of the RegistryEntryQuery.

3. Return any accumulated warnings or exceptions as the StatusResult associated with the RegistryEntryQuery.
**Examples**

A client wants 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 superceded, replaced, deprecated, or withdrawn items.

```
<RegistryEntryQuery>
  <RegistryEntryFilter>
    Status EQ "Registered" -- code by Clause, Section 8.2.10
  </RegistryEntryFilter>
  <SubmittingOrgFilter>
    <OrganizationFilter>
      Name CONTAINS "XYZ" -- code by Clause, Section 8.2.10
    </OrganizationFilter>
  </SubmittingOrgFilter>
</RegistryEntryQuery>
```

A client is using the UNSPSC classification 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 PartyProfile documents in the Registry, and that each profile has been classified by the products the company deals with. The following query returns a set of registry entry identifiers for profiles of companies that deal with integrated circuit components.

```
<RegistryEntryQuery>
  <RegistryEntryFilter>
    ObjectType EQ "PartyProfile" AND Status EQ "Registered" -- code by Clause, Section 8.2.10
  </RegistryEntryFilter>
  <RegistryEntryClassification>
    <ClassificationNodeFilter>
      ID STARTSWITH "urn:un:spsc:321118" -- code by Clause, Section 8.2.10
    </ClassificationNodeFilter>
  </RegistryEntryClassification>
</RegistryEntryQuery>
```

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. The root nodes of each scheme are identified by "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/Automotive" and by "Geography/Asia/Japan".
<RegistryEntryQuery>
  <RegistryEntryClassification>
    <ClassificationNodeFilter>
      ID STARTSWITH "urn:ebxml:cs:industry" AND
      Path EQ "Industry/Automotive" -- code by Clause, Section 8.2.10
    </ClassificationNodeFilter>
    <ClassificationNodeFilter>
      ID STARTSWITH "urn:ebxml:cs:geography" AND
      Path EQ "Geography/Asia/Japan" -- code by Clause, Section 8.2.10
    </ClassificationNodeFilter>
  </RegistryEntryClassification>
</RegistryEntryQuery>

A client application wishes to identify all registry Package instances that have a
given registry entry as a member of the package. The following query identifies
all registry packages that contain the registry entry identified by URN
"urn:path:myitem" as a member:

<RegistryEntryQuery>
  <RegistryEntryFilter>
    objectType EQ "RegistryPackage" -- code by Clause, Section 8.2.10
  </RegistryEntryFilter>
</RegistryEntryQuery>

<AsSourceAssociation>
  <AssociationFilter>
    AssociationType EQ "HasMember" AND
    TargetObject EQ "urn:path:myitem"
  </AssociationFilter>
</AsSourceAssociation>

A client application wishes to identify all ClassificationNode instances that have
some given keyword as part of their name or description. The following query
identifies all registry classification nodes that contain the keyword "transistor" as
part of their name or as part of their description.

<RegistryEntryQuery>
  <RegistryEntryFilter>
    ObjectType="ClassificationNode" AND
    (Name CONTAINS "transistor" OR -- code by Clause, Section 8.2.10
    Description CONTAINS "transistor")
  </RegistryEntryFilter>
</RegistryEntryQuery>
8.2.3 AuditableEventQuery

Purpose

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

RIM Binding

Definition

```
<!ELEMENT AuditableEventQuery
   ( AuditableEventFilter?,
     RegistryEntryQuery*,
     UserFilter?,
     OrganizationQuery? )>
```

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 an AuditableEventFilter is not specified, or if AE is empty, then continue below; otherwise, let x be an auditable event in AE. If x does not satisfy the AuditableEventFilter as defined in section 8.2.9, then remove x from AE.

b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue below; otherwise, let x be a remaining auditable event in AE. Treat each RegistryEntryQuery element separately as follows:

Let RE be the result set of the RegistryEntryQuery as defined in section 8.2.2. If x is not an auditable event for some registry entry in RE, then remove x from AE.

c) If an UserFilter element is not specified, or if AE is empty, then continue below; otherwise, let x be a remaining auditable event in AE. Let AI be the set of all User instances that satisfy the UserFilter and are linked to x as their auditable event. If AI is empty, then remove x from AE.

d) If an OrganizationQuery element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If an UserFilter element is not specified, then let AI be the set of all User instances that are linked to x as their auditable event; otherwise, let AI be the set of all User instances that satisfy the UserFilter and are linked to x as their auditable event. Let OG be the set of Organization instances that are the organization of an element in AI and are in the result set of the OrganizationQuery. If OG is empty, then remove x from AE.

2. If AE is empty, then raise the warning: **auditable event query result is empty**.

3. Return AE as the result of the AuditableEventQuery.

4. Return any accumulated warnings or exceptions as the StatusResult associated with the AuditableEventQuery.

Examples

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

```
<AuditableEventquery>
  <AuditableEventFilter>
    Timestamp GE "2001-01-01" AND -- code by Clause, Section 8.2.10
    RegistryEntry EQ "urn:path:myitem"
  </AuditableEventFilter>
</AuditableEventQuery>
```
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 events that have an impact on an item that you submitted, and you are responsible for, but the event was initiated by some other party.

```xml
<AuditableEventquery>
  <RegistryEntryQuery>
    <SubmittingOrgFilter>
      <OrganizationFilter>
        ID EQ "urn:somepath:myorg" -- code by Clause, Section 8.2.10
      </OrganizationFilter>
    </SubmittingOrgFilter>
    <ResponsibleOrgFilter>
      <OrganizationFilter>
        ID EQ "urn:somepath:myorg" -- code by Clause, Section 8.2.10
      </OrganizationFilter>
    </ResponsibleOrgFilter>
  </RegistryEntryQuery>
  <UserFilter>
    <OrganizationFilter>
      ID NE "urn:somepath:myorg" -- code by Clause, Section 8.2.10
    </OrganizationFilter>
  </UserFilter>
</AuditableEventQuery>
```
8.2.4 ClassificationNodeQuery

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

RIM Binding

ClassifiesRegistryEntry

Classification

RegistryEntry

ClassificationNode

HasParentNode

ClassifiesRegistryEntry

ClassificationNode

HasSubnode

Definition

<!ELEMENT ClassificationNodeQuery (
  ClassificationNodeFilter?,
  ClassifiesRegistryEntry*,
  HasParentNode?,
  HasSubnode* )>

<!ELEMENT ClassifiesRegistryEntry ( ClassificationFilter?,
  RegistryEntryQuery? )>

<!ELEMENT HasParentNode ( ClassificationNodeFilter?,
  HasParentNode? )>

<!ELEMENT HasSubnode ( ClassificationNodeFilter?,
  HasSubnode* )>
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 a ClassificationNodeFilter is not specified, or if CN is empty, then continue below; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter as defined in section 8.2.9, then remove x from AE.

   b) If a ClassifiesRegistryEntry element is not specified, or if CN is empty, then continue below; otherwise, let x be a remaining classification node in CN. If x is not the target object of some Classification instance, then remove x from CN; otherwise, treat each ClassifiesRegistryEntry element separately as follows:

      If no ClassificationFilter is specified within the ClassifiesRegistryEntry element, then let CL be the set of all Classification instances that have x as the target object; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the target object. If CL is empty, then remove x from CN. If no RegistryEntryQuery is specified within the ClassifiesRegistryEntry element, then let RES be the set of all RegistryEntry instances that are the source object of some classification instance in CL; otherwise, let RE be the result set of the RegistryEntryQuery as defined in section 8.2.2 and let RES be the set of all instances in RE that are the source object of some classification in CL. If RES is empty, then remove x from CN.

   c) If a HasParentNode element is not specified, or if CN is empty, then continue below; 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 root node), then remove x from CN. Let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in HasParentNode and if p does not satisfy the ClassificationNodeFilter, then remove x from CN.

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

   d) If a HasSubnode element is not specified, or if CN is empty, then continue below; otherwise, let x be a remaining classification node in CN. If x is not the parent node of some ClassificationNode instance, then remove x from CN; otherwise, treat each HasSubnode element separately and execute the following paragraph with n = x.
Let \( n \) be a classification node instance. If a `ClassificationNodeFilter` is not specified within the `HasSubnode` element then let \( \text{CNC} \) be the set of all classification nodes that have \( n \) as their parent node; otherwise, let \( \text{CNC} \) be the set of all classification nodes that satisfy the `ClassificationNodeFilter` and have \( n \) as their parent node. If \( \text{CNC} \) is empty then remove \( x \) from \( \text{CN} \); otherwise, let \( y \) be an element of \( \text{CNC} \) and continue with the next paragraph.

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

2. If \( \text{CN} \) is empty, then raise the warning: `classification node query result is empty`.

3. Return \( \text{CN} \) as the result of the `ClassificationNodeQuery`.

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

**Examples**

A client application wishes to identify all classification nodes defined in the Registry that are root nodes and have a name that contains the phrase “product code” or the phrase “product type”. Note: By convention, if a classification node has no parent (i.e. is a root node), then the parent attribute of that instance is set to null and is represented as a literal by a zero length string.

```xml
<ClassificationNodeQuery>
    <ClassificationNodeFilter>
        (name CONTAINS "product code" OR name CONTAINS "product type") AND parent EQ ""
    </ClassificationNodeFilter>
</ClassificationNodeQuery>
```

A client application wishes to identify all of the classification nodes at the third level of a classification scheme hierarchy. The client knows that the URN identifier for the root node is `urn:ebxml:cs:myroot`. The following query identifies all nodes at the second level under “myroot” (i.e. third level overall).

```xml
<ClassificationNodeQuery>
    <HasParentNode>
        <HasParentNode>
            <ClassificationNodeFilter>
                ID EQ "urn:ebxml:cs:myroot"
            </ClassificationNodeFilter>
        </HasParentNode>
    </HasParentNode>
</ClassificationNodeQuery>
```
8.2.5 RegistryPackageQuery

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

RIM Binding

```
<!ELEMENT RegistryPackageQuery ( PackageFilter?, PackageHasMember* )>
<!ELEMENT PackageHasMember ( RegistryEntryQuery?, )>
```

Semantic Rules

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

   a) If a PackageFilter is not specified, or if RP is empty, then continue below; otherwise, let x be a package instance in RP. If x does not satisfy the PackageFilter as defined in section 8.2.9, then remove x from RP.

   b) If a PackageHasMember element is not directly contained in the RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x be a remaining package instance in RP. If x is an empty package, then remove x from RP; otherwise, treat each PackageHasMember element separately as follows:
If a RegistryEntryQuery element is not directly contained in the
PackageHasMember element, then let PM be the set of all RegistryEntry
instances that are members of the package x; otherwise, let RE be the set
of RegistryEntry instances returned by the RegistryEntryQuery as defined
in section 8.2.2 and let PM be the subset of RE that are members of the
package x. If PM is empty, then remove x from RP.

2. If RP is empty, then raise the warning: \textit{registry package query result is empty}.

3. Return RP as the result of the RegistryPackageQuery.

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

\textbf{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.

\begin{verbatim}
<RegistryPackageQuery>
  <PackageHasMember>
    <RegistryEntryQuery>
      <RegistryEntryFilter>
        objectType EQ "Invoice" -- code by Clause, Section 8.2.10
      </RegistryEntryFilter>
    </RegistryEntryQuery>
  </PackageHasMember>
</RegistryPackageQuery>
\end{verbatim}

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

\begin{verbatim}
<RegistryEntryQuery>
  <PackageHasMember/>
</RegistryEntryQuery>
\end{verbatim}

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.

\textbf{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 “AsSource” and “As Target” associations. However, the equivalent
RegistryEntryQuery is often more complicated to write.
### 8.2.6 OrganizationQuery

**Purpose**

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

**RIM Binding**

**Definition**

```xml
<!ELEMENT OrganizationQuery
  ( OrganizationFilter?,
    SubmitsEntry*,
    HasParentOrganization?,
    InvokesEvent*,
    ContactFilter* )>

<!ELEMENT SubmitsEntry ( RegistryEntryQuery? )>

<!ELEMENT HasParentOrganization
  ( OrganizationFilter?,
    HasParentOrganization? )>

<!ELEMENT InvokesEvent
  ( UserFilter?,
    AuditableEventFilter?,
    AuditableEvent )>

<!ELEMENT RegistryEntry
  ( RegistryEntry? )>
```
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 an OrganizationFilter element is not directly contained in the OrganizationQuery element, or if ORG is empty, then continue below; otherwise, let x be an organization instance in ORG. If x does not satisfy the OrganizationFilter as defined in section 8.2.9, then remove x from ORG.

   b) If a SubmitsEntry element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, consider each SubmitsEntry element separately as follows:

      If no RegistryEntryQuery is specified within the SubmitsEntry element, then let RES be the set of all RegistryEntry instances that have been submitted to the Registry by organization x; otherwise, let RE be the result of the RegistryEntryQuery as defined in section 8.2.2 and let RES be the set of all instances in RE that have been submitted to the Registry by organization x. If RES is empty, then remove x from ORG.

   c) If a HasParentOrganization element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute the following paragraph with o = x:

      Let o be an organization instance. If an OrganizationFilter is not specified within the HasParentOrganization 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 another HasParentOrganization element is directly contained within this HasParentOrganization element, then repeat the previous paragraph with o = p.

   d) If an InvokesEvent element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, consider each InvokesEvent element separately as follows:
If an UserFilter is not specified, and if \( x \) is not the submitting organization of some AuditableEvent instance, then remove \( x \) from ORG. If an AuditableEventFilter is not specified, then let \( AE \) be the set of all AuditableEvent instances that have \( x \) as the submitting organization; otherwise, let \( AE \) be the set of AuditableEvent instances that satisfy the AuditableEventFilter and have \( x \) as the submitting organization. If \( AE \) is empty, then remove \( x \) from ORG. If a RegistryEntryQuery is not specified in the InvokesEvent element, then let \( RES \) be the set of all RegistryEntry instances associated with an event in \( AE \); otherwise, let \( RE \) be the result set of the RegistryEntryQuery, as specified in section 8.2.2, and let \( RES \) be the subset of \( RE \) of entries submitted by \( x \). If \( RES \) is empty, then remove \( x \) from ORG.

e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, consider each ContactFilter separately as follows:

Let \( CT \) be the set of Contact instances that satisfy the ContactFilter and are the contacts for organization \( x \). If \( CT \) is empty, then remove \( x \) from ORG.

2. If ORG is empty, then raise the warning: organization query result is empty.

3. Return ORG as the result of the OrganizationQuery.

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

Examples

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

```xml
<OrganizationQuery>
  <OrganizationFilter>
    country EQ "France"  -- code by Clause, Section 8.2.10
  </OrganizationFilter>
  <SubmitsEntry>
    <RegistryEntryQuery>
      <RegistryEntryFilter>
        objectType EQ "PartyProfile"  -- code by Clause, Section 8.2.10
      </RegistryEntryFilter>
      <RegistryEntryAuditableEvent>
        <AuditableEventFilter>
          timestamp GE "2001-01-01"  -- code by Clause, Section 8.2.10
        </AuditableEventFilter>
      </RegistryEntryAuditableEvent>
    </RegistryEntryQuery>
  </SubmitsEntry>
</OrganizationQuery>
```
A client application wishes to identify all organizations that have XYZ, Corporation as a parent. The client knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is no guarantee that subsidiaries of XYZ have a URN that uses the same format, so a full query is required.

```xml
<OrganizationQuery>
  <HasParentOrganization>
    <OrganizationFilter>
      ID = "urn:ebxml:org:xyz" -- code by Clause, Section 8.2.10
    </OrganizationFilter>
  </HasParentOrganization>
</OrganizationQuery>
```
8.2.7 GetRegistryEntry

Purpose
To construct an XML document that contains selected registry metadata associated with the registry entries identified by a RegistryEntryQuery. NOTE: Initially, the RegistryEntryQuery could be the URN identifier for a single registry entry.

Definition

```xml
<!ELEMENT GetRegistryEntry
  ( RegistryEntryQuery,
    WithClassifications?,
    WithAsSourceAssociations?,
    WithAsTargetAssociations?,
    WithAuditableEvents?,
    WithExternalLinks? )>

<!ELEMENT WithClassifications  ( ClassificationFilter? )>
<!ELEMENT WithAsSourceAssociations  ( AssociationFilter? )>
<!ELEMENT WithAsTargetAssociations  ( AssociationFilter? )>
<!ELEMENT WithAuditableEvents  ( AuditableEventFilter? )>
<!ELEMENT WithExternalLinks  ( ExternalLinkFilter? )>

<!ELEMENT GetRegistryEntryResult
  ( RegistryEntryMetadata*,  StatusResult )>

<!ELEMENT RegistryEntryMetadata
  ( RegistryEntry,
    Classification*,
    AsSourceAssociations?,
    AsTargetAssociations?,
    AuditableEvent*,
    ExternalLink* )>

<!ELEMENT AsSourceAssociations ( Association* )>
<!ELEMENT AsTargetAssociations ( Association* )>
```

Semantic Rules

1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink elements contained in the GetRegistryEntryResult are defined by the ebXML Registry DTD specified in Appendix A.2.

2. Execute the RegistryEntryQuery according to the Semantic Rules specified in section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let S be the set of status elements returned in the StatusResult. If any status element in S is an exception condition, then stop execution and return the same StatusResult element in the GetRegistryEntryResult.
3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the GetRegistryEntryResult. Instead, raise the warning: no resulting registry entry. Add this warning to the StatusResult returned by the RegistryEntryQuery and return this enhanced StatusResult with the GetRegistryEntryResult.

4. For each registry entry E referenced by an element of R, use the attributes of E to create a new RegistryEntry element as defined in Appendix A.2. Then create a new RegistryEntryMetadata element as defined above to be the parent element of that RegistryEntry element.

5. If no With option is specified, then the resulting RegistryEntryMetadata element has no Classification, AsSourceAssociations, AsTargetAssociations, AuditableEvent, or ExternalData subelements. The set of RegistryEntryMetadata elements, with the StatusResult from the RegistryEntryQuery, is returned as the GetRegistryEntryResult.

6. If WithClassifications is specified, then for each E in R do the following: If a ClassificationFilter is not present, then let C be any classification instance linked to E; otherwise, let C be a classification instance linked to E that satisfies the ClassificationFilter (Section 8.2.9). For each such C, create a new Classification element as defined in Appendix A.2. Add these Classification elements to their parent RegistryEntryMetadata element.

7. If WithAsSourceAssociations is specified, then for each E in R do the following: If an AssociationFilter is not present, then let A be any association instance whose source object is E; otherwise, let A be an association instance that satisfies the AssociationFilter (Section 8.2.9) and whose source object is E. For each such A, create a new Association element as defined in Appendix A.2. Add these Association elements as subelements of the WithAsSourceAssociations and add that element to its parent RegistryEntryMetadata element.

8. If WithAsTargetAssociations is specified, then for each E in R do the following: If an AssociationFilter is not present, then let A be any association instance whose target object is E; otherwise, let A be an association instance that satisfies the AssociationFilter (Section 8.2.9) and whose target object is E. For each such A, create a new Association element as defined in Appendix A.2. Add these Association elements as subelements of the WithAsTargetAssociations and add that element to its parent RegistryEntryMetadata element.

9. If WithAuditableEvents is specified, then for each E in R do the following: If an AuditableEventFilter is not present, then let A be any auditable event instance linked to E; otherwise, let A be any auditable event instance linked to E that satisfies the AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent element as defined in Appendix A.2. Add these AuditableEvent elements to their parent RegistryEntryMetadata element.
10. If WithExternalLinks is specified, then for each E in R do the following: If an ExternalLinkFilter is not present, then let L be any external link instance linked to E; otherwise, let L be any external link instance linked to E that satisfies the ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink element as defined in Appendix A.2. Add these ExternalLink elements to their parent RegistryEntryMetadata element.

11. If any warning or exception condition results, then add the code and the message to the StatusResult that came from the RegistryEntryQuery result.

12. Return the set of RegistryEntryMetadata elements and the revised StatusResult as the content of the GetRegistryEntryResult.

Examples

A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer wishes to check on the current status of that DTD, especially if it has been superceded or replaced, and get all of its current classifications. The following query request will return an XML document with the registry entry for the existing DTD as the root, with all of its classifications, and with associations to registry entries for any items that have superceded or replaced it.

```
<GetRegistryEntry>
  <RegistryEntryQuery>
    <RegistryEntryFilter>
      ID EQ "urn:com:xyz:po:325"  -- code by Clause, Section 8.2.10
    </RegistryEntryFilter>
  </RegistryEntryQuery>
  <WithClassifications/>
  <WithAsSourceAssociations>
    <AssociationFilter>
      -- code by Clause, Section 8.2.10
      AssociationType EQUALS "SupercededBy" OR AssociationType EQUALS "ReplacedBy"
    </AssociationFilter>
  </WithAsSourceAssociations>
</GetRegistryEntry>
```

A client of the Registry registered an XML DTD several years ago and is now thinking of replacing it with a revised version. The identifier for the existing DTD is "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the existing DTD. The client desires a list of all registered items that use the existing DTD so they can assess the impact of an incompatible change. The following query returns an XML document that is a list of all RegistryEntry elements that represent registered items that use, contain, or extend the given DTD. The document also links each RegistryEntry element in the list to an element for the identified association.
A user has been browsing the registry and has found a registry entry that describes a package of core-components that should solve the user’s problem. The packageURN identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to know what's in the package. The following query returns an XML document with a registry entry for each member of the package along with that member's Uses and PackageHasMember associations.

```
<GetRegistryEntry>
  <RegistryEntryQuery>
    <AsSourceAssociation>
      <AssociationFilter> -- code by Clause, Section 8.2.10
        AssociationType EQ "Contains" OR
        AssociationType EQ "Uses" OR
        AssociationType EQ "Extends"
      </AssociationFilter>
      <RegistryEntryFilter> -- code by Clause, Section 8.2.10
        ID = "urn:xyz:.dtd:po97"
      </RegistryEntryFilter>
    </AsSourceAssociation>
  </RegistryEntryQuery>
  <WithAsSourceAssociations>
    <AssociationFilter> -- code by Clause, Section 8.2.10
      AssociationType EQ "Contains" OR
      AssociationType EQ "Uses" OR
      AssociationType EQ "Extends"
    </AssociationFilter>
  </WithAsSourceAssociations>
</GetRegistryEntry>

<GetRegistryEntry>
  <RegistryEntryQuery>
    <AsTargetAssociation>
      <AssociationFilter> -- code by Clause, Section 8.2.10
        AssociationType EQ "HasMember"
      </AssociationFilter>
      <RegistryEntryFilter> -- code by Clause, Section 8.2.10
        ID = "urn:com:cc:pkg:ccstuff"
      </RegistryEntryFilter>
    </AsTargetAssociation>
  </RegistryEntryQuery>
  <WithAsSourceAssociations>
    <AssociationFilter> -- code by Clause, Section 8.2.10
      AssociationType EQ "HasMember" OR
      AssociationType EQ "Uses"
    </AssociationFilter>
  </WithAsSourceAssociations>
</GetRegistryEntry>
```
8.2.8 GetRepositoryItem

Purpose
To construct an XML document that contains one or more repository items, and
some associated metadata, by submitting a RegistryEntryQuery to the
registry/repository that holds the desired objects. NOTE: Initially, the
RegistryEntryQuery could be the URN identifier for a single registry entry.

Definition

```xml
<!ELEMENT GetRepositoryItem
( RegistryEntryQuery,
   RecursiveAssociationOption?,
   WithShortDescription? )>

<!ELEMENT RecursiveAssociationOption ( AssociationRole+ )>
<!ATTLIST RecursiveAssociationOption
depthLimit CDATA #IMPLIED >

<!ELEMENT AssociationRole EMPTY >
<!ATTLIST AssociationRole
role CDATA #REQUIRED >

<!ELEMENT WithShortDescription EMPTY >

<!ELEMENT GetRepositoryItemResult
( RepositoryItem*, StatusResult )>

<!ELEMENT RepositoryItem
(   ClassificationScheme
| RegistryPackage
| ExtrinsicObject
| WithdrawnObject
| ExternalItem )>
<!ATTLIST RepositoryItem
identifier CDATA #REQUIRED
name CDATA #REQUIRED
repositoryURL CDATA #REQUIRED
objectType CDATA #REQUIRED
status CDATA #REQUIRED
stability CDATA #REQUIRED
description CDATA #IMPLIED >

<!ELEMENT ExtrinsicObject (#PCDATA) >
<!ATTLIST ExtrinsicObject
byteEncoding CDATA "Base64" >

<!ELEMENT WithdrawnObject EMPTY >

<!ELEMENT ExternalItem EMPTY >
```

Semantic Rules

1. If the RecursiveOption element is not present, then set Limit=0. If the RecursiveOption element is present, interpret its depthLimit attribute as an integer literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0 means that no recursion occurs. A Limit of -1 means that recursion occurs indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive integer, then stop execution and raise the exception: invalid depth limit; otherwise, set Limit=N, where N is that positive integer. A Limit of N means that exactly N recursive steps will be executed unless the process terminates prior to that limit.

2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned as part of the GetRepositoryItemResult. Initially Result is empty. Semantic rules 4 through 10 determine the content of Result.

3. If the WithShortDescription element is present, then set WSD="yes"; otherwise, set WSD="no".

4. Execute the RegistryEntryQuery according to the Semantic Rules specified in section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let S be the set of status elements returned in the StatusResult. If any status element in S is an exception condition, then stop execution and return the same StatusResult element in the GetRepositoryItemResult.

5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from R. After execution of these rules, if Depth is now equal to Limit, then return the content of Result as the set of RepositoryItem elements in the GetRepositoryItemResult element; otherwise, continue with Semantic Rule 8.

6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the following:
   a) If E.repositoryURL references a repository item in this registry/repository, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7.
      1) If E.objectType="ClassificationScheme", then put the referenced ClassificationScheme DTD as the subelement of this RepositoryItem. [NOTE: Requires DTD specification!]
      2) If E.objectType="RegistryPackage", then put the referenced RegistryPackage DTD as the subelement of this RepositoryItem. [NOTE: Requires DTD specification!]
      3) Otherwise, i.e., if the object referenced by E has an unknown internal structure, then put the content of the repository item as the #PCDATA of a new ExtrinsicObject subelement of this RepositoryItem.
b) If E.objectURL references a registered object in some other registry/repository, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7, and create a new ExternalItem element as the subelement of this RepositoryItem.

c) If E.objectURL is void, i.e. the object it would have referenced has been withdrawn, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7, and create a new WithdrawnObject element as the subelement of this RepositoryItem.

7. Let E be a registry entry and let RO be the RepositoryItem element created in Semantic Rule 6. Set the attributes of RO to the values derived from the corresponding attributes of E. If WSD="yes", include the value of the description attribute; otherwise, do not include it. Insert this new RepositoryItem element into the Result set.

8. Let R be defined as in Semantic Rule 4. Execute Semantic Rule 9 with Y as the set of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.

9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty set of RegistryEntry instances. For each registry entry E in Y, and for each AssociationRole A of the RecursiveAssociationOption, do the following:

   a) Let Z be the set of target items E' linked to E under association instances having E as the source object, E' as the target object, and A as the AssociationType.

   b) Add the elements of Z to NextLevel.

10. Let X be the set of new registry entries that are in NextLevel but are not yet represented in the Result set.

    Case:

       a) If X is empty, then return the content of Result as the set of RepositoryItem elements in the GetRepositoryItemResult element.

       b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set. When finished, add the elements of X to Y and set Depth=Depth+1. If Depth is now equal to Limit, then return the content of Result as the set of RepositoryItem elements in the GetRepositoryItemResult element; otherwise, repeat Semantic Rules 9 and 10 with the new set Y of registry entries.

11. If any exception, warning, or other status condition results during the execution of the above, then return appropriate status elements as the StatusResult of the GetRepositoryItemResult element created in Semantic Rule 5 or Semantic Rule 10.
Examples

A registry client has found a registry entry for a core-component item. The item's URN identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many other registered items. The client desires the collection of all items needed for a complete implementation of "goodthing". The following query returns an XML document that is a collection of all needed items.

```xml
<GetRepositoryItem>
  <RegistryEntryQuery>
    <RegistryEntryFilter> -- code by Clause, Section 8.2.10
      ID EQ "urn:ebxml:cc:goodthing"
    </RegistryEntryFilter>
  </RegistryEntryQuery>
  <RecursiveAssociationOption>
    <AssociationRole role="Uses" />
    <AssociationRole role="ValidatesTo" />
  </RecursiveAssociationOption>
</GetRepositoryItem>
```

A registry client has found a reference to a core-component routine ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client knows that all routines have a required association to its defining UML specification. The following query returns both the routine and its UML specification as a collection of two items in a single XML document.

```xml
<GetRepositoryItem>
  <RegistryEntryQuery>
    <RegistryEntryFilter> -- code by Clause, Section 8.2.10
      ID EQ "urn:ebxml:cc:rtn:nice87"
    </RegistryEntryFilter>
  </RegistryEntryQuery>
  <RecursiveAssociationOption depthLimit="1">
    <AssociationRole role="ValidatesTo" />
  </RecursiveAssociationOption>
</GetRepositoryItem>
```

A user has been told that the 1997 version of the North American Industry Classification System (NAICS) is stored in the NIST registry with URN identifier "urn:nist:cs:naics-1997". The following query would retrieve the complete classification scheme, with all 1810 nodes, as an XML document that validates to a classification scheme DTD.

```xml
<GetRepositoryItem>
  <RegistryEntryQuery>
    <RegistryEntryFilter> -- code by Clause, Section 8.2.10
      ID EQ "urn:nist:cs:naics-1997"
    </RegistryEntryFilter>
  </RegistryEntryQuery>
</GetRepositoryItem>
```
Note: The GetRepositoryItemResult would include a single RepositoryItem that consists of the ClassificationScheme document with content:
ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt
8.2.9 Registry Filters

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

Definition

<!ELEMENT ObjectFilter ( Clause )>
<!ELEMENT RegistryEntryFilter ( Clause )>
<!ELEMENT IntrinsicObjectFilter ( Clause )>
<!ELEMENT ExtrinsicObjectFilter ( Clause )>
<!ELEMENT PackageFilter ( Clause )>
<!ELEMENT OrganizationFilter ( Clause )>
<!ELEMENT ContactFilter ( Clause )>
<!ELEMENT ClassificationNodeFilter ( Clause )>
<!ELEMENT AssociationFilter ( Clause )>
<!ELEMENT ClassificationFilter ( Clause )>
<!ELEMENT ExternalLinkFilter ( Clause )>
<!ELEMENT AuditableEventFilter ( Clause )>
<!ELEMENT UserFilter ( Clause )>

Semantic Rules
1. The Clause element is defined in section 8.2.10, Clause.
2. For every ObjectFilter XML element, the leftargument attribute of any containing SimpleClause shall identify a public attribute of the Object UML class defined in [RIM]. If not, raise exception: object attribute error. The ObjectFilter returns a set of identifiers for Object 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 [RIM].
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 IntrinsicObjectFilter XML element, the leftargument attribute of any
containing SimpleClause shall identify a public attribute of the IntrinsicObject
UML class defined in [RIM]. If not, raise exception: *intrinsic object attribute
error*. The IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject
instances whose attribute values evaluate to *True* for the Clause predicate.

5. For every ExtrinsicObjectFilter XML element, the leftargument attribute of any
containing SimpleClause shall identify a public attribute of the ExtrinsicObject
UML class defined in [RIM]. 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.

6. For every PackageFilter XML element, the leftargument attribute of any
containing SimpleClause shall identify a public attribute of the Package UML
class defined in [RIM]. If not, raise exception: *package attribute error*. The
PackageFilter returns a set of identifiers for Package instances whose
attribute values evaluate to *True* for the Clause predicate.

7. 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 [RIM]. 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.

8. For every ContactFilter XML element, the leftargument attribute of any
containing SimpleClause shall identify a public attribute of the Contact or
PostalAddress UML class defined in [RIM]. If not, raise exception: *contact attribute error*. The ContactFilter returns a set of identifiers for Contact
instances whose attribute values evaluate to *True* for the Clause predicate.

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

10. For every AssociationFilter XML element, the leftargument attribute of any
containing SimpleClause shall identify a public attribute of the Association
UML class defined in [RIM]. 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.
11. For every ClassificationFilter XML element, the leftargument attribute of any containing SimpleClause shall identify a public attribute of the Classification UML class defined in [RIM]. 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.

12. For every ExternalLinkFilter XML element, the leftargument attribute of any containing SimpleClause shall identify a public attribute of the ExternalLink UML class defined in [RIM]. 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.

13. For every AuditableEventFilter XML element, the leftargument attribute of any containing SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [RIM]. 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.

14. For every UserFilter XML element, the leftargument attribute of any containing SimpleClause shall identify a public attribute of the User UML class defined in [RIM]. If not, raise exception: auditable identity attribute error. The UserFilter returns a set of identifiers for User instances whose attribute values evaluate to True for the Clause predicate.
8.2.10 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 UML Diagram
The following is a conceptual diagram outlining the Clause base structure. It is expressed in UML for visual depiction.
**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 left argument 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**

```xml
<!ELEMENT Clause ( SimpleClause | CompoundClause )>  
<!ELEMENT SimpleClause ( BooleanClause | RationalClause | StringClause )>  
<!ATTLIST SimpleClause leftargument CDATA #REQUIRED >  
<!ELEMENT CompoundClause ( Clause, Clause+ )>  
<!ATTLIST CompoundClause connectivepredicate ( And | Or ) #REQUIRED>  
<!ELEMENT BooleanClause EMPTY >  
<!ATTLIST BooleanClause booleanpredicate ( True | False ) #REQUIRED>  
<!ELEMENT RationalClause ( IntClause | FloatClause )>  
<!ATTLIST RationalClause logicalpredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >  
<!ELEMENT IntClause ( #PCDATA )>  
<!ATTLIST IntClause e-dtype NMTOKEN #FIXED 'int' >  
<!ELEMENT FloatClause ( #PCDATA )>  
<!ATTLIST FloatClause e-dtype NMTOKEN #FIXED 'float' >  
<!ELEMENT StringClause ( #PCDATA )>  ```
<!ATTLIST StringClause stringpredicate (contains | -contains | startswith | -startswith | endswith | -endswith ) #REQUIRED >

Examples

Simple BooleanClause: "Smoker" = True

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

Simple StringClause: "Smoker" contains "mo"

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

Simple IntClause: "Age" >= 7

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

Simple FloatClause: "Size" = 4.3

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <SimpleClause leftargument="Size">
    <RationalClause logicalpredicate="E">
      <FloatClause e-dtype="float">4.3</FloatClause>
    </RationalClause>
  </SimpleClause>
</Clause>
Compound with two Simples ("Smoker" = False) AND ("Age" <= 45)

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <CompoundClause connectivepredicate="And">
    <Clause>
      <SimpleClause leftargument="Smoker">
        <BooleanClause booleanpredicate="False"/>
      </SimpleClause>
    </Clause>
    <Clause>
      <SimpleClause leftargument="Age">
        <RationalClause logicalpredicate="EL">
          <IntClause e-dtype="int">45</IntClause>
        </RationalClause>
      </SimpleClause>
    </Clause>
  </CompoundClause>
</Clause>

Compound with one Simple and one Compound
( ("Smoker" = False) AND ("Age" <= 45) OR ("American" = True) )

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Clause SYSTEM "Clause.dtd" >
<Clause>
  <CompoundClause connectivepredicate="And">
    <Clause>
      <SimpleClause leftargument="Smoker">
        <BooleanClause booleanpredicate="False"/>
      </SimpleClause>
    </Clause>
    <Clause>
      <CompoundClause connectivepredicate="Or">
        <Clause>
          <SimpleClause leftargument="Age"/>
          <RationalClause logicalpredicate="EL">
            <IntClause e-dtype="int">45</IntClause>
          </RationalClause>
        </SimpleClause>
      </CompoundClause>
    </Clause>
    <Clause>
      <SimpleClause leftargument="American"/>
    </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 complex 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 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 C.3. The exact syntax of the Registry query language is defined by the BNF grammar in C.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 [RIM]

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

8.3.1.1 Interface and Class Binding

A subset of the Interface and class names defined in [RIM] map to table names that may be queried by an SQL query. Appendix C.3 defines the names of the RIM interfaces and classes that may be queries by an SQL query.

The algorithm used to define the binding of [RIM] classes to table definitions in appendix C.3 is as follows:

Only those classes and interfaces that have concrete instances are mapped to relational tables. This results in intermediate interfaces in the inheritance hierarchy, such as Object and IntrinsicObject to not map to SQL tables. An exception to this rule is RegistryEntry as defined next.
A special view called RegistryEntry is defined to allow SQL queries to be made against RegistryEntry instances. This is the only interface defined in [RIM] that does not have concrete instances but is queryable by SQL queries.

The names of relational tables are the same as the corresponding [RIM] class or interface name. However, the name binding is case insensitive.

Each [RIM] class or interface that maps to a table in appendix C.3 includes column definitions in appendix C.3 where the column definitions are based on a subset of attributes defined for that class or interface in [RIM]. The attributes that map to columns include the inherited attributes for the [RIM] class or interface. Comments in appendix C.3 indicate which ancestor class or interface contributed which column definitions.

An SQLQuery against a table not defined in appendix C.3 may result in an ebXMLError message with an InvalidQueryException.

The algorithm for mapping of attributes to column definitions will be described in following sections.

**8.3.1.2 Accessor Method To Attribute Binding**

Most of the [RIM] interfaces methods are simple get methods that map directly to attributes. For example the getName method on Object maps to a name attribute of type String. Each get method in [RIM] defines the exact attribute name that it maps to in the interface definitions in [RIM].

**8.3.1.3 Primitive Attributes Binding**

Attributes defined by [RIM] 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 interface definitions in [RIM]. 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 [RIM].

**8.3.1.4 Reference Attribute Binding**

A few of the [RIM] interface methods return references to instances of interfaces or classes defined by [RIM]. For example, the getAccessControlPolicy method of the Object 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 [RIM] as defined by 8.3.1.3. The data type for the column is UUID as defined in appendix C.3.
When a reference attribute value holds a null reference it maps to a null value in the SQL binding which may be tested with the <null specification> as defined by [SQL].

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

8.3.1.5 Complex Attribute Binding

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

The SQL query schema algorithmically maps such complex attributes as multiple primitive attributes within the parent table. The mapping simply flattens out the entity class attributes within the parent table. The attribute name for the flattened attributes are composed of a concatenation of attribute names in the reference chain. For example Organization has a contact attribute of type Contact. Contact has an address attribute of type PostalAddress. PostalAddress has a String attribute named city. This city attribute will be named contact_address_city.

8.3.1.6 Collection Attribute Binding

A few of the [RIM] interface methods return collection of references to instances of interfaces or classes defined by [RIM]. 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 attributes in [RIM] classes have been mapped to stored procedures in appendix C.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 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].
The SQL select column specified must always be \texttt{id} for this version of the specification, where \texttt{t} is a table reference in the FROM clause.

### 8.3.3 SQL Query Results

The results of an SQL query is always an ObjectRefList as defined by the AdHocQueryResponse in 8.3.12. This means the result of an SQL query is always a collection of references to instances of a sub-class of the Object interface in [RIM]. This is reflected in a semantic constraint that requires that the SQL select column specified must always be an \texttt{id} column in a table in appendix C.3 for this version of the specification.

### 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 [RIM]. 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 predicates must be supported:

\begin{verbatim}
SELECT id FROM ExtrinsicObject WHERE name LIKE '%Acme%' AND majorVersion >= 1 AND (majorVersion >= 2 OR minorVersion > 3);
\end{verbatim}

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 RegistryEntry Queries

Given the central role played by the RegistryEntry interface in RIM, 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 8.3.1.2 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.

\begin{verbatim}
SELECT id FROM RegistryEntry WHERE name LIKE '%Acme%' AND majorVersion >= 1 AND (majorVersion >= 2 OR re.minorVersion > 3);
\end{verbatim}
8.3.6 Classification Queries

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

8.3.6.1 Identifying ClassificationNodes

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

8.3.6.2 Getting Root Classification Nodes

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

```
SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

The above query returns all ClassificationNodes that have their parent attribute set to null. Note that the above query may also specify a predicate on the name if a specific root ClassificationNode is desired.

8.3.6.3 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.

8.3.6.4 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.
8.3.6.5 Getting ClassificationNodes That Classify an Object

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

```
SELECT id FROM ClassificationNode
WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

8.3.7 Association Queries

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

8.3.7.1 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>
```

8.3.7.2 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>
```

8.3.7.3 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 source role name.
```
SELECT id FROM Association WHERE sourceRole = <roleName>
```

Select Associations that have the specified target role name.
```
SELECT id FROM Association WHERE targetRole = <roleName>
```

Select Associations that have the specified association type, where association type is a string containing the corresponding field name described in [RIM].
```
SELECT id FROM Association WHERE
associationType = <associationType>
```

8.3.7.4 Complex Association Queries

The various forms of Association queries may be combined into complex predicates. The following query selects Associations from an object with a specified id, that have the sourceRole “buysFrom” and targetRole “sellsTo”:

```
SELECT id FROM Association WHERE
sourceObject = <id> AND
sourceRole = 'buysFrom' AND
sourceRole = 'sellsTo'
```
8.3.8 Package Queries

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

```
SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

8.3.8.1 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 WHERE
id IN (RegistryEntry_packages(<id>)) AND
name LIKE '%RosettaNet%' AND
status <> 'DEPRECATED'
```

8.3.9 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 (RegistryEntry_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 (RegistryEntry_linkedObjects(<id>))
```

8.3.9.1 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 (RegistryEntry_externalLinks(<id>)) AND
description LIKE '%legal%' AND
externalURI LIKE '%http://%'
```

8.3.10 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 registryEntry = <id>
```

8.3.11 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.
8.3.11.1 Automatic Classification of XML Content

Content-based queries are indirectly supported through the existing classification mechanism supported by the Registry.

A submitting organization may define logical indexes on any XML schema or DTD when it is submitted. An instance of such a logical index defines a link between a specific attribute or element node in an XML document tree and a ClassificationNode in a classification scheme within the registry.

The registry utilizes this index to automatically classify documents that are instances of the schema at the time the document instance is submitted. Such documents are classified according to the data contained within the document itself.

Such automatically classified content may subsequently be discovered by clients using the existing classification-based discovery mechanism of the Registry and the query facilities of the ObjectQueryManager.

[Note] This approach is conceptually similar to the way databases support indexed retrieval. DBAs define indexes on tables in the schema. When data is added to the table, the data gets automatically indexed.

8.3.11.2 Index Definition

This section describes how the logical indexes are defined in the SubmittedObject element defined in the Registry DTD. The complete Registry DTD is specified in Appendix A.2.

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 Object in [RIM]. 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 path expression as defined by [XPT].

8.3.11.3 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:
8.3.11.4 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. Note that if either of the two ClassificationNodes named “buyer” and “seller” did not previously exist, the ObjectManager would automatically create these ClassificationNodes.

8.3.12 Ad Hoc Query Request/Response

A client submits an ad hoc query to the ObjectQueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a query in one of the supported Registry query mechanisms. The ObjectQueryManager sends an AdhocQueryResponse either synchronously or asynchronously back to the client. The AdhocQueryResponse return a collection of objects whose element type is in the set of element types represented by the leaf nodes of the RegistryEntry hierarchy in [RIM].

Figure 15: Submit Ad Hoc Query Sequence Diagram
For details on the schema for the business documents shown in this process refer to Appendix A.2.

8.4 Content Retrieval

A client retrieves content via the Registry by sending the GetContentRequest to the ObjectQueryManager. The GetContentRequest specifies a list of Object references for Objects that need to be retrieved. The ObjectQueryManager returns the specified content by sending a GetContentResponse message to the ObjectQueryManagerClient 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 GetContentResponse payload includes an ebXMLError and there are no additional content specific payloads.

8.4.1 Retrieval of Registry Profile

A special case of content retrieval is the retrieval of the RegistryProfile XML document. The RegistryProfile XML document is retrieved by specifying a special id named “RegistryProfileID” as the value of the id attribute for the ObjectRef element in GetContentRequest.
8.4.2 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 for each RegistryEntry instance that describes the repository item as the DocumentLabel element in the DocumentReference for that object in the Manifest element of the ebXMLHeader.

8.4.3 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. Note that the ID for each object retrieved in the message as additional payloads is used as its DocumentLabel in the Manifest of the ebXMLHeader.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<GetContentsResponse />
```
8.5 Query And Retrieval: Typical Sequence

The following diagram illustrates the use of both browse/drilldown and ad hoc queries followed by a retrieval of content that was selected by the queries.

![Diagram of Typical Query and Retrieval Sequence]

Figure 17: Typical Query and Retrieval Sequence

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 [RIM].
In the current version of this specification, a minimalist approach has been specified for Registry security. The philosophy is that “Any known entity 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.

9.1 Integrity of Registry Content

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

9.1.1 Message Payload Signature

Integrity of Registry content requires that all submitted content must be signed by the Registry client as defined by [SEC]. The signature on the submitted content ensures that:

?? The content has not been tampered with en-route or within the Registry.

?? The content’s veracity can be ascertained by its association with a specific submitting organization

9.2 Authentication

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

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

The Registry must implement a credential-based authentication mechanism based on digital certificates and signatures. The Registry uses the certificate DN from the signature to authenticate the user.
9.2.1 Message Header Signature

Message headers may be signed by the sending ebXML Messaging Service as defined by [SEC]. Since this specification is not yet finalized, this version does not require that the message header be signed. In the absence of a message header signature, the payload signature is used to authenticate the identity of the requesting client.

9.3 Confidentiality

9.3.1 On-the-wire Message Confidentiality

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

9.3.2 Confidentiality of Registry Content

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

9.4 Authorization

The Registry must provide an authorization mechanism based on the information model defined in [RIM]. 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.

9.4.1 Pre-defined Roles For Registry Users

The following roles must be pre-defined in the Registry:

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentOwner</td>
<td>The submitter or owner of a Registry content. Submitting Organization (SO) in ISO 11179</td>
</tr>
<tr>
<td>RegistryAdministrator</td>
<td>A “super” user that is an administrator of the Registry. Registration Authority (RA) in ISO 11179</td>
</tr>
<tr>
<td>RegistryGuest</td>
<td>Any unauthenticated user of the Registry. Clients that</td>
</tr>
</tbody>
</table>
9.4.2 Default Access Control Policies

The Registry must create a default AccessControlPolicy object that grants the default permissions to Registry users based upon their assigned role.

The following table defines the Permissions granted by the Registry to the various pre-defined roles for Registry users based upon the default AccessControlPolicy.

<table>
<thead>
<tr>
<th>Role</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentOwner</td>
<td>Access to all methods on Registry Objects that are owned by the ContentOwner.</td>
</tr>
<tr>
<td>RegistryAdministrator</td>
<td>Access to all methods on all Registry Objects</td>
</tr>
<tr>
<td>RegistryGuest</td>
<td>Access to all read-only (getXXX) methods on all Registry Objects</td>
</tr>
</tbody>
</table>

The following list summarizes the default role-based AccessControlPolicy:

- The Registry must implement the default AccessControlPolicy and associate it with all Objects in the Registry.
- Anyone can publish content, but needs to be authenticated.
- Anyone can access the content without requiring authentication.
- The ContentOwner has access to all methods for Registry Objects owned by them.
- The RegistryAdministrator has access to all methods on all Registry Objects.
- Unauthenticated clients can access all read-only (getXXX) methods.
- At the time of content submission, the Registry must assign the default ContentOwner role to the Submitting Organization (SO) as authenticated by the credentials in the submission message. In the current version of this specification, it 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  Schemas and DTD Definitions

The following are definitions for the various ebXML Message payloads described in this document.

A.1 ebXMLError Message DTD

See [ERR] for ebXMLError Message DTD.

A.2 ebXML Registry DTD

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE errorSchema SYSTEM "ebXMLError.dtd">

<!-- Begin information model mapping. -->
%errorSchema;

<!-- ObjectAttributes are attributes from the Object interface in RIM. 
id may be empty. If specified it may be in urn:uuid format or be in some arbitrary format. 
If id is empty registry must generate globally unique id. 
If id is provided and in proper UUID syntax (starts with urn:uuid:) registry will honour it 
If id is provided and is not in proper UUID syntax then it is used for linkage within document 
and is ignored by the registry. In this case the registry generates a UUID for id attribute. 
id must not be null when object is being retrieved from the registry. -->
<!ENTITY % ObjectAttributes "
  id          ID  #IMPLIED
  name        CDATA  #IMPLIED
  description CDATA  #IMPLIED">

<!ELEMENT ObjectRef EMPTY>
<!ATTLIST ObjectRef
  id ID #IMPLIED>
<!ELEMENT ObjectRefList (ObjectRef)*>

<!-- RegistryEntryAttributes are attributes from the RegistryEntry interface in RIM. 
It inherits ObjectAttributes -->
<!ENTITY % RegistryEntryAttributes " %ObjectAttributes;
  majorVersion    CDATA  '1'
  minorVersion    CDATA  '0'
  status          CDATA  #IMPLIED
  userVersion     CDATA  #IMPLIED
  stability       CDATA  'Dynamic'
  expirationDate  CDATA  #IMPLIED">
<!ELEMENT RegistryEntry (SlotList?)>
<!ATTLIST RegistryEntry %RegistryEntryAttributes;>
<!ELEMENT Value (#PCDATA)>  
<!ELEMENT ValueList (Value*)>  
<!ELEMENT Slot (ValueList?)>  
<!ATTLIST Slot name CDATA #REQUIRED  
slotType CDATA #IMPLIED >  
<!ELEMENT SlotList (Slot*)>  
<!ELEMENT ExtrinsicObject (ClassificationIndexList?)>  
<!ATTLIST ExtrinsicObject %RegistryEntryAttributes;  
contentURI CDATA #REQUIRED  
mimeType CDATA #IMPLIED  
objectType CDATA #REQUIRED  
opaque (true | false) "false" >  
<!ELEMENT ClassificationIndex EMPTY>  
<!ATTLIST ClassificationIndex %ObjectAttributes;  
classificationNode IDREF #REQUIRED  
contentIdentifier CDATA #REQUIRED >  
<!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">  
<!-- Leaf classes that reflect the concrete classes in RIM -->  
<!ELEMENT RegistryEntryList (Association | Classification | ClassificationNode | Package  
| ExternalLink | ExternalIdentifier | Organization | ExtrinsicObject | ObjectRef)+>  
<!ELEMENT ExternalLink EMPTY>  
<!ATTLIST ExternalLink %IntrinsicObjectAttributes;  
externalURI CDATA #IMPLIED >  
<!ELEMENT ExternalIdentifier EMPTY>  
<!ATTLIST ExternalIdentifier %IntrinsicObjectAttributes;>

EnhancementsObject are attributes from the EnhancedObject interface in RIM. It inherits RegistryEntryAttributes -->  
<!ELEMENT ExtrinsicObject (ClassificationIndexList?)>  
<!ATTLIST ExtrinsicObject %RegistryEntryAttributes;  
contentURI CDATA #IMPLIED  
mimeType CDATA #IMPLIED  
objectType CDATA #IMPLIED  
opaque (true | false) "false" >  
<!ELEMENT ClassificationIndexList (ClassificationIndex)*>  
<!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">  
<!-- Leaf classes that reflect the concrete classes in RIM -->  
<!ELEMENT RegistryEntryList (Association | Classification | ClassificationNode | Package  
| ExternalLink | ExternalIdentifier | Organization | ExtrinsicObject | ObjectRef)+>  
<!ELEMENT ExternalLink EMPTY>  
<!ATTLIST ExternalLink %IntrinsicObjectAttributes;  
externalURI CDATA #IMPLIED >  
<!ELEMENT ExternalIdentifier EMPTY>  
<!ATTLIST ExternalIdentifier %IntrinsicObjectAttributes;>
An Association specifies references to two previously submitted registry entries.

The sourceObject is id of the sourceObject in association

The targetObject is id of the targetObject in association

<!ELEMENT Association EMPTY>
<!ATTLIST Association
   %IntrinsicObjectAttributes;
   fromLabel CDATA #IMPLIED
   toLabel CDATA #IMPLIED
   associationType CDATA #REQUIRED
   bidirection (true | false) "false"
   sourceObject IDREF #REQUIRED
   targetObject IDREF #REQUIRED>

A Classification specifies references to two registry entries.

The classifiedObject is id of the Object being classified.

The classificationNode is id of the ClassificationNode classifying the object

<!ELEMENT Classification EMPTY>
<!ATTLIST Classification
   %IntrinsicObjectAttributes;
   classifiedObject IDREF #REQUIRED
   classificationNode IDREF #REQUIRED>

A Package is a named collection of objects.

<!ELEMENT Package EMPTY>
<!ATTLIST Package
   %IntrinsicObjectAttributes;>

<!-- Attributes inherited by various types of telephone number elements -->
<ENTITY % TelephoneNumberAttributes " areaCode   CDATA  #REQUIRED
   countryCode CDATA  #REQUIRED
   extension  CDATA  #IMPLIED
   number     CDATA  #REQUIRED
   url        CDATA  #IMPLIED">
<!ELEMENT TelephoneNumber EMPTY>
<!ATTLIST TelephoneNumber
   %TelephoneNumberAttributes;>

<!ELEMENT FaxNumber EMPTY>
<!ATTLIST FaxNumber
   %TelephoneNumberAttributes;>

<!ELEMENT PagerNumber EMPTY>
<!ATTLIST PagerNumber
   %TelephoneNumberAttributes;>

<!ELEMENT MobileTelephoneNumber EMPTY>
<!ATTLIST MobileTelephoneNumber
   %TelephoneNumberAttributes;>

<!-- PostalAddress -->
<!ELEMENT PostalAddress EMPTY>
<!ATTLIST PostalAddress
  city CDATA #REQUIRED
  country CDATA #REQUIRED
  postalCode CDATA #REQUIRED
  state CDATA #REQUIRED
  street CDATA #REQUIRED
>
<!-- PersonName -->
<!ELEMENT PersonName EMPTY>
<!ATTLIST PersonName
  firstName CDATA #REQUIRED
  middleName CDATA #REQUIRED
  lastName CDATA #REQUIRED
>
<!-- Organization -->
<!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
<!ATTLIST Organization
  %IntrinsicObjectAttributes;
  parent IDREF #IMPLIED
  primaryContact IDREF #REQUIRED
>
<!ELEMENT User (PersonName, PostalAddress, TelephoneNumber, MobileTelephoneNumber?, FaxNumber?, PagerNumber?)>
<!ATTLIST User
  %ObjectAttributes;
  organization IDREF #IMPLIED
  email CDATA #IMPLIED
  url CDATA #IMPLIED
>
<!ELEMENT AuditableEvent EMPTY>
<!ATTLIST AuditableEvent
  %ObjectAttributes;
  eventType CDATA #REQUIRED
  registryEntry IDREF #REQUIRED
  timestamp CDATA #REQUIRED
  user IDREF #REQUIRED
>
<!-- ClassificationNode is used to submit a Classification tree to the Registry. parent is the id to the parent node. code is an optional code value for a ClassificationNode often defined by an external taxonomy (e.g. NAICS) -->
<!ELEMENT ClassificationNode EMPTY>
<!ATTLIST ClassificationNode
  %IntrinsicObjectAttributes;
  parent IDREF #IMPLIED
  code CDATA #IMPLIED
>
<!-- End information model mapping. Begin Registry Services Interface -->
<!ELEMENT RequestAcceptedResponse EMPTY>
<!ATTLIST RequestAcceptedResponse
  xml:lang NMTOKEN #REQUIRED
  interfaceId CDATA #REQUIRED
  requestMessage CDATA #REQUIRED
  actionId CDATA #REQUIRED
>
The SubmittedObject provides meta data for submitted object
Note object being submitted is in a separate document that is not
in this DTD.

<!--
<!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
<!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
<!-- Only need name in Slot within SlotList -->
<!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
<!-- The ObjectRefList is the list of refs to the registry entries being approved. -->
<!ELEMENT ApproveObjectsRequest (ObjectRefList)>
<!-- The ObjectRefList is the list of refs to the registry entries being deprecated. -->
<!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
<!-- The ObjectRefList is the list of refs to the registry entries being removed -->
<!ELEMENT RemoveObjectsRequest (ObjectRefList)>
<!ATTLIST RemoveObjectsRequest
   deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
>
<!ELEMENT GetRootClassificationNodesRequest EMPTY>
<!-- The namePattern follows SQL-92 syntax for the pattern specified in LIKE clause. It allows for selecting only those root nodes that match the namePattern. The default value of '*' matches all root nodes. -->
<!ELEMENT GetRootClassificationNodesResponse ((ClassificationNode+) | ebXMLError )>
<!ELEMENT GetClassificationTreeRequest EMPTY>
<!ATTLIST GetClassificationTreeRequest
   parent CDATA #REQUIRED
   depth CDATA "1"
>
<!ELEMENT GetClassificationTreeResponse ((ClassificationNode+) | ebXMLError)>
The default value of '*' implies that client is interested in all types of registry entries that are classified by the specified ClassificationNode.

The response includes a RegistryEntryList which has zero or more RegistryEntries that are classified by the ClassificationNodes specified in the ObjectRefList in GetClassifiedObjectsRequest.

An Ad hoc query request specifies a query string as defined by [RS] in the queryString attribute.

The response includes a RegistryEntryList which has zero or more RegistryEntries that match the query specified in AdhocQueryRequest.

The GetObjectsResponse will have no sub-elements if there were no errors. The actual contents will be in the other payloads of the message. If any errors were encountered the message will contain the ebXMLError and the content payloads will be empty.

Describes the capability profile for the registry and what optional features are supported.

<!-- Begin FilterQuery DTD -->

<!ELEMENT RegistryEntryQuery (AuditableEventQuery | ClassificationNodeQuery | RegistryPackageQuery | OrganizationQuery)>  
<!ELEMENT AuditableEventQuery (AuditEventQuery*)>  
<!ELEMENT AuditEventQuery EMPTY>  
<!ATTLIST AuditEventQuery  
objectURN CDATA #REQUIRED  
contentURL CDATA #IMPLIED  
objectID CDATA #IMPLIED  
timestamp CDATA #REQUIRED  
>  
<!ELEMENT ClassificationNodeQuery (ClassificationNodeView*)>  
<!ELEMENT ClassificationNodeView EMPTY>  
<!ATTLIST ClassificationNodeView  
objectURN CDATA #REQUIRED  
contentURL CDATA #IMPLIED  
objectID CDATA #REQUIRED  
timestamp CDATA #REQUIRED  
>
<!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
<!ELEMENT RegistryPackageView EMPTY>
<!ATTLIST RegistryPackageView
  objectURN CDATA #REQUIRED
  contentURL CDATA #IMPLIED
  objectID CDATA #IMPLIED>

<!ELEMENT OrganizationQueryResult (OrganizationView*)>
<!ELEMENT OrganizationView EMPTY>
<!ATTLIST OrganizationView
  orgURN CDATA #REQUIRED
  contactURL CDATA #IMPLIED
  objectID CDATA #IMPLIED>

<!ELEMENT StatusResult (Success | (Exception | Warning)+)>
<!ELEMENT Success EMPTY>
<!ELEMENT Exception (#PCDATA)>
<!ATTLIST Exception
  code CDATA #REQUIRED>
<!ELEMENT Warning (#PCDATA)>
<!ATTLIST Warning
  code CDATA #REQUIRED>

<!ELEMENT RegistryEntryQuery (RegistryEntryFilter?, AsSourceAssociation*,
  AsTargetAssociation*, RegistryEntryClassification*, SubmittingOrgFilter?,
  ResponsibleOrgFilter?, ExternalLinkFilter*, RegistryEntryAuditableEvent*)>
<!ELEMENT AsSourceAssociation (AssociationFilter?, RegistryEntryFilter?)>
<!ELEMENT AsTargetAssociation (AssociationFilter?, RegistryEntryFilter?)>
<!ELEMENT RegistryEntryClassification (ClassificationFilter?, ClassificationNodeFilter?)>
<!ELEMENT SubmittingOrgFilter (OrganizationFilter?, ContactFilter?)>
<!ELEMENT ResponsibleOrgFilter (OrganizationFilter?, ContactFilter?)>
<!ELEMENT RegistryEntryAuditableEvent (AuditableEventFilter?, UserFilter?,
  OrganizationFilter?)>

<!ELEMENT AuditableEventQuery (AuditableEventFilter?, RegistryEntryQuery*, UserFilter?,
  OrganizationQuery?)>
<!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?, ClassifiesRegistryEntry*,
  HasParentNode?, HasSubnode*)>
<!ELEMENT ClassifiesRegistryEntry (ClassificationNodeFilter?, RegistryEntryQuery?)>
<!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
<!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
<!ELEMENT RegistryPackageQuery (PackageFilter?, PackageHasMember*)>
<!ELEMENT PackageHasMember (RegistryEntryQuery?)>
<!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsEntry*, HasParentOrganization?,
  InvokesEvent*, ContactFilter*)>
<!ELEMENT SubmitsEntry (RegistryEntryQuery?)>
<!ELEMENT HasParentOrganization (OrganizationFilter?, HasParentOrganization?)>
<!ELEMENT InvokesEvent (UserFilter?, AuditableEventFilter?, RegistryEntryQuery?)>
<!ELEMENT GetRegistryEntry (RegistryEntryQuery, WithClassifications?,
  WithAsSourceAssociations?, WithAsTargetAssociations?, WithAuditableEvents?,
  WithExternalLinks?)>
<!ELEMENT GetRepositoryItem (RegistryEntryQuery, RecursiveAssociationOption?,
  WithShortDescription?)>
<!ELEMENT RecursiveAssociationOption (AssociationRole+)>
<!ELEMENT GetRegistryEntryResult (RegistryEntryMetadata*, StatusResult)>
<!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*, AsSourceAssociations?,
  AsTargetAssociations?, AuditableEvent*, ExternalLink*)>
<!ATTLIST RecursiveAssociationOption
  depthLimit CDATA #IMPLIED
>
<!ELEMENT AssociationRole EMPTY>
<!ATTLIST AssociationRole
  role CDATA #REQUIRED
>
<!ELEMENT WithShortDescription EMPTY>
<!ELEMENT GetRepositoryItemResult (RepositoryItem*, StatusResult)>
<!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject | ExternalItem)>
<!ATTLIST RepositoryItem
  identifier CDATA #REQUIRED
  name CDATA #REQUIRED
  repositoryURL CDATA #REQUIRED
  objectType CDATA #REQUIRED
  status CDATA #REQUIRED
  stability CDATA #REQUIRED
  description CDATA #IMPLIED
>
<!ELEMENT RegistryPackage EMPTY>
<!ELEMENT WithdrawnObject EMPTY>
<!ELEMENT ExternalItem EMPTY>
<!ELEMENT ObjectFilter (Clause)>
<!ELEMENT RegistryEntryFilter (Clause)>
<!ELEMENT IntrinsicObjectFilter (Clause)>
<!ELEMENT ExtrinsicObjectFilter (Clause)>
<!ELEMENT PackageFilter (Clause)>
<!ELEMENT OrganizationFilter (Clause)>
<!ELEMENT ClassificationNodeFilter (Clause)>
<!ELEMENT AssociationFilter (Clause)>
<!ELEMENT ClassificationFilter (Clause)>
<!ELEMENT ExternalLinkFilter (Clause)>
<!ELEMENT AuditableEventFilter (Clause)>
<!ELEMENT UserFilter (Clause)>
<!ELEMENT Clause (SimpleClause | CompoundClause)>
<!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
<!ATTLIST SimpleClause
  leftArgument CDATA #REQUIRED
>
<!ELEMENT CompoundClause (Clause, Clause+)>
<!ATTLIST CompoundClause
  connectivePredicate (And | Or) #REQUIRED
>
<!ELEMENT BooleanClause EMPTY>
<!ATTLIST BooleanClause
  booleanPredicate (true | false) #REQUIRED
>
<!ELEMENT RationalClause (IntClause | FloatClause)>
<!ATTLIST RationalClause
  logicalPredicate (LE | LT | GE | GT | EQ | NE) # REQUIRED
>
<!ELEMENT IntClause (#PCDATA)>
<!ATTLIST IntClause
e-dtype NMTOKEN #FIXED "int"
>
<!ELEMENT FloatClause (#PCDATA)>
<!ATTLIST FloatClause
e-dtype NMTOKEN #FIXED "float"
>
<!ELEMENT StringClause (#PCDATA)>
<!ATTLIST StringClause
  stringPredicate (contains | -contains | startswith | -startswith | endswith | -endswith) #REQUIRED
>
<!-- End FilterQuery DTD -->
Appendix B Interpretation of UML Diagrams

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

B.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces (as defined by [CPA]) required to implement an ebXML Registry Services and clients. See Figure 2 on page 15 for an example. The UML class diagram contains:

1. A collection of UML interfaces where each interface represents a Service Interface for a Registry service.

2. Tabular description of methods on each interface where each method represents an Action (as defined by [CPA]) 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.

B.2 UML Sequence Diagram

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

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 4 on page 22 for an
example.

Appendix C   SQL Query

C.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].

1. The SQL query syntax conforms to the <query specification>, modulo the
restrictions identified below
2. A <select list> may contain at most one <select sublist>
3. In a <select list> must be is a single column whose data type is UUID,
from the table in the <from clause>,
4. A <derived column> may not have an <as clause>
5. <table expression> does not contain the optional <group by clause> and
 <having clause> clauses.
6. A <table reference> can only consist of <table name> and <correlation
name>
7. A <table reference> does not have the optional AS between <table name>
and <correlation name>
8. There can only be one <table reference> in the <from clause>
9. 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.
10. A <search condition> within the <where clause> may not include a <query
expression>.
11. The SQL query syntax allows for the use of <sql invoked routines>
 invocation from [SQL/PSM] as the RHS of the <in predicate>.
C.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 C.1.

```plaintext
/****************************
* The Registry Query (Subset of SQL-92) grammar starts here
* ****************************/
RegistryQuery = SQLSelect [";"]
SQLSelect = "SELECT" 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 = SQILikeClause
    | SQLInClause
    | SQLCompareOp SQLSumExpr
SQLCompareOp = "+="
    | ">="
    | ">>
    | "><="
    | "<=
    | "<"
SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
SQLLValueElement = "NULL" | SQLSelect
SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
SQILikeClause = [ "NOT" ] "LIKE" SQLPattern
SQLPattern = STRING_LITERAL
SQLLiteral = STRING_LITERAL
    | INTEGER_LITERAL
    | FLOATING_POINT_LITERAL
```
C.3 Relational Schema For SQL Queries

-- SQL Load file for creating the ebXML Registry tables

-- Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92

CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
CREATE TYPE UUID UNDER ShortName FINAL;
CREATE TYPE URI UNDER LongName FINAL;
CREATE TABLE ExtrinsicObject (
    id     UUID PRIMARY KEY NOT NULL,
    name     LongName,
    description    FreeFormText,
    accessControlPolicy  UUID NOT NULL,

    -- Versionable attributes
    majorVersion    INT DEFAULT 0 NOT NULL,
    minorVersion    INT DEFAULT 1 NOT NULL,

    -- RegistryEntry attributes
    status    INT DEFAULT 0 NOT NULL,
    userVersion ShortName,
    stability    INT DEFAULT 0 NOT NULL,

    SQLColRef = SQLvalue
    SQLvalue = SQLvalueTerm
    SQLvalueTerm = 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)?
    EXPONENT = ["e","E"] (["+","-"])? (["0"-"9"])+
    STRING_LITERAL: "'" (~\['\])* ( "''" (~\['\])* )* "'
    ID = ( <LETTER> )+ ( "_" | "\$" | "#" | <DIGIT> | <LETTER> )*
    LETTER = ["A"-"Z", "a"-"z"]
    DIGIT = ("0"-"9")

Copyright © ebXML 2000 & 2001. All Rights Reserved.
CREATE TABLE Package {
  id     UUID PRIMARY KEY NOT NULL,
  name     LongName,
  description    FreeFormText,
  accessControlPolicy  UUID NOT NULL,
  majorVersion    INT DEFAULT 0 NOT NULL,
  minorVersion    INT DEFAULT 1 NOT NULL,
  status    INT DEFAULT 0 NOT NULL,
  userVersion    ShortName,
  stability     INT DEFAULT 0 NOT NULL,
  expirationDate   TIMESTAMP,
  --Package attributes
};

CREATE PROCEDURE Package_memberb jects(packageId) {
  --Must return a collection of UUIDs for RegistryEntrys that are members of this Package.
}

CREATE TABLE ExternalLink {
  id     UUID PRIMARY KEY NOT NULL,
  name     LongName,
  description    FreeFormText,
  accessControlPolicy  UUID NOT NULL,
  contentURI    URI,
  mimeType     ShortName,
  objectType    INT DEFAULT 0 NOT NULL,
  opaque    BOOLEAN DEFAULT false NOT NULL
 );

CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
  --Must return a collection of UUIDs for related RegistryEntry instances
}

CREATE PROCEDURE RegistryEntry审计Trail(registryEntryId) {
  --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
  --Collection must be in ascending order by timestamp
}

CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
  --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
  --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
  --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
  --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
}

CREATE TABLE ExternalLink {
  id     UUID PRIMARY KEY NOT NULL,
  name     LongName,
  description    FreeFormText,
  accessControlPolicy  UUID NOT NULL,
  contentURI    URI,
  mimeType     ShortName,
  objectType    INT DEFAULT 0 NOT NULL,
  opaque    BOOLEAN DEFAULT false NOT NULL
 );

CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
  --Must return a collection of UUIDs for related RegistryEntry instances
}

CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
  --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
  --Collection must be in ascending order by timestamp
}

CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
  --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
  --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
  --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
}

CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
  --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
}

CREATE TABLE Package {
  id     UUID PRIMARY KEY NOT NULL,
  name     LongName,
  description    FreeFormText,
  accessControlPolicy  UUID NOT NULL,
  majorVersion    INT DEFAULT 0 NOT NULL,
  minorVersion    INT DEFAULT 1 NOT NULL,
  status    INT DEFAULT 0 NOT NULL,
  userVersion    ShortName,
  stability     INT DEFAULT 0 NOT NULL,
  expirationDate   TIMESTAMP,
  --Package attributes
};

CREATE PROCEDURE Package_memberb jects(packageId) {
  --Must return a collection of UUIDs for RegistryEntrys that are members of this Package.
}

CREATE TABLE ExternalLink {
  id     UUID PRIMARY KEY NOT NULL,
  name     LongName,
  description    FreeFormText,
  accessControlPolicy  UUID NOT NULL,
--Versionable attributes
majorVersion   INT DEFAULT 0 NOT NULL,
minorVersion   INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
status         INT DEFAULT 0 NOT NULL,
userVersion    ShortName,
stability      INT DEFAULT 0 NOT NULL,
expirationDate TIMESTAMP,

--ExternalLink attributes
externalURI    URI NOT NULL

CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
    --Must return a collection of UUIDs for objects in this relationship
}

CREATE TABLE ExternalIdentifier (
    id     UUID PRIMARY KEY NOT NULL,
    name     LongName,
    description    FreeFormText,
    accessControlPolicy  UUID NOT NULL,
)

--Versionable attributes
majorVersion    INT DEFAULT 0 NOT NULL,
minorVersion    INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
status    INT DEFAULT 0 NOT NULL,
userVersion    ShortName,
stability     INT DEFAULT 0 NOT NULL,
expirationDate   TIMESTAMP,

--ExternalIdentifier attributes
value    ShortName NOT NULL

--A SlotValue row represents one value of one slot in some
--RegistryEntry
CREATE TABLE SlotValue (}
    registryEntry   UUID PRIMARY KEY NOT NULL,
    name    LongName NOT NULL PRIMARY KEY NOT NULL,
    value   ShortName NOT NULL
)

CREATE TABLE Association (}
    id     UUID PRIMARY KEY NOT NULL,
    name     LongName,
    description    FreeFormText,
    accessControlPolicy  UUID NOT NULL,
)

--Versionable attributes
majorVersion    INT DEFAULT 0 NOT NULL,
minorVersion    INT DEFAULT 1 NOT NULL,

--RegistryEntry attributes
status    INT DEFAULT 0 NOT NULL,
CREATE TABLE Classification (
    id     UUID PRIMARY KEY NOT NULL,
    name   LongName,
    description    FreeFormText,
    accessControlPolicy  UUID NOT NULL,

    majorVersion    INT DEFAULT 0 NOT NULL,
    minorVersion    INT DEFAULT 1 NOT NULL,

    status    INT DEFAULT 0 NOT NULL,
    userVersion    ShortName,
    stability     INT DEFAULT 0 NOT NULL,
    expirationDate   TIMESTAMP,

    sourceObject    UUID NOT NULL,
    targetObject    UUID NOT NULL,

    userVersion    ShortName,
    stability     INT DEFAULT 0 NOT NULL,
    expirationDate   TIMESTAMP,

    associationType   INT NOT NULL,
    bidirectional    BOOLEAN DEFAULT false NOT NULL,
    sourceObject    UUID NOT NULL,
    sourceRole    ShortName,
    targetObject    UUID NOT NULL,
    targetRole    ShortName
);
CREATE TABLE AuditableEvent (  
id     UUID PRIMARY KEY NOT NULL,  
name     LongName,  
description    FreeFormText,  
accessControlPolicy  UUID NOT NULL,  
user  UUID,  
eventType     INT DEFAULT 0 NOT NULL,  
registryEntry    UUID NOT NULL,  
timestamp     TIMESTAMP NOT NULL,  
);  

CREATE TABLE User (  
id     UUID PRIMARY KEY NOT NULL,  
name     LongName,  
description    FreeFormText,  
accessControlPolicy  UUID NOT NULL,  
organization   UUID NOT NULL,  
email   ShortName,  
fax_areaCode   VARCHAR(4) NOT NULL,  
fax_countryCode  VARCHAR(4),  
fax_extension   VARCHAR(8),  
fax_umber    VARCHAR(8) NOT NULL,  
fax_url    URI  
);
CREATE TABLE Organization (  
--Object Attributes  
id UUID PRIMARY KEY NOT NULL,  
name LongName,  
description FreeFormText,  
accessControlPolicy UUID NOT NULL,  

--Versionable attributes  
majorVersion INT DEFAULT 0 NOT NULL,  
minorVersion INT DEFAULT 1 NOT NULL,  

--RegistryEntry attributes  
status INT DEFAULT 0 NOT NULL,  
userVersion ShortName,  
stability INT DEFAULT 0 NOT NULL,  
expirationDate TIMESTAMP,  

--Organization attributes  

--Organization.address attribute flattened  
address_city ShortName,  
address_country ShortName,  
address_postalCode ShortName,  
address_state ShortName,  
address_street ShortName,  

--primary contact for Organization, points to a User.  
contact UUID NOT NULL,  

--Organization.fax attribute flattened  
fax_areaCode VARCHAR(4) NOT NULL,  
fax_countryCode VARCHAR(4),  
fax_extension VARCHAR(8),  
fax_umber VARCHAR(8) NOT NULL,  
fax_url URI,  

--Organization.parent attribute  
parent UUID,  

--Organization.telephone attribute flattened  
television_areaCode VARCHAR(4) NOT NULL,  
television_countryCode VARCHAR(4),  
television_extension VARCHAR(8),  
television_umber VARCHAR(8) NOT NULL,  
television_url URI  
);  

--Note that the RIM security view is not visible through the public query mechanism  
in the current release  

--The RegistryEntry View allows polymorphic queries over all RIM classes derived  
from RegistryEntry  

CREATE VIEW RegistryEntry (  
--Object Attributes  
id,  
name,  
description,  
accessControlPolicy,  

--Versionable attributes
-- RegistryEntry attributes
status, userVersion, stability, expirationDate

) AS
SELECT -- Object Attributes
  id, name, description, accessControlPolicy,

-- Versionable attributes
  majorVersion, minorVersion,

-- RegistryEntry attributes
status, userVersion, stability, expirationDate

FROM ExtrinsicObject
UNION

SELECT -- Object Attributes
  id, name, description, accessControlPolicy,

-- Versionable attributes
  majorVersion, minorVersion,

-- RegistryEntry attributes
status, userVersion, stability, expirationDate

FROM (Registry)Package
UNION

SELECT -- Object Attributes
  id, name, description, accessControlPolicy,

-- Versionable attributes
  majorVersion, minorVersion,

-- RegistryEntry attributes
status, userVersion, stability, expirationDate

FROM ClassificationNode;
Appendix D  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.

D.1 Authentication

1. As soon as a message is received, the first work is the authentication. A principal object is created.
2. If the message is signed, it is verified (including the validity of the certificate) and the DN of the certificate becomes the identity of the principal. Then the Registry is searched for the principal and if found, the roles and groups are filled in.
3. If the message is not signed, an empty principal is created with the role RegistryGuest. This step is for symmetry and to decouple the rest of the processing.
4. Then the message is processed for the command and the objects it will act on.

D.2 Authorization

For every object, the access controller will iterate through all the AccessControlPolicy objects with the object and see if there is a chain through the permission objects to verify that the requested method is permitted for the Principal. If any of the permission objects which the object is associated with has a common role, or identity, or group with the principal, the action is permitted.

D.3 Registry Bootstrap

When a Registry is newly created, a default Principal object should be created with the identity of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message signed by the Registry Admin will get all the privileges.

When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the default AccessControlPolicy. This includes the creation of the necessary Permission instances as well as the Privileges and Privilege attributes.

D.4 Content Submission – Client Responsibility

The Registry client has to sign the contents before submission – otherwise the content will be rejected.
D.5 Content Submission – Registry Responsibility

1. Like any other request, the client will be first 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.

D.6 Content Delete/Deprecate – Client Responsibility

The Registry client has to sign the payload (not entire message) before submission, for authentication purposes; otherwise, the request will be rejected.

D.7 Content Delete/Deprecate – Registry Responsibility

1. Like any other request, the client will be first 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 Object will be accessed.
3. The access controller performs the authorization by iterating through the Permission objects associated with this object via the singleton default AccessControlPolicy.
4. If authorization succeeds then the action will be permitted. Otherwise an error response is sent back with a suitable AuthorizationException error message.

Appendix E Native Language Support (NLS)

E.1 Definitions

Although this section discusses only character set and language, the following terms have to be defined clearly.
E.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.

E.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.

E.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].

E.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".

Ex. 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].

E.3 NLS And Storing of RegistryEntry
This section provides NLS guidelines on how a registry should store RegistryEntry instances.
E.3.1 Character Set of RegistryEntry

This is basically an implementation issue because the actual character set that the RegistryEntry is stored with, does not affect the interface. However, it is highly recommended to use UTF-16 or UTF-8 for covering various languages.

E.3.2 Language Information of RegistryEntry

The language may be specified in xml:lang attribute (section 2.12 [REC-XML]). If the xml:lang attribute is specified, then the registry may use that language code as the value of a special Slot with name language and sloType of nls in the RegistryEntry. The value must be compliant to [RFC 1766]. Slots are defined in [RIM].

E.4 NLS And Storing of Repository Items

This section provides NLS guidelines on how a registry should store repository items.

E.4.1 Character Set of Repository Items

Unlike the character set of RegistryEntry, the charset of a repository item must be preserved as it is originally specified in the transaction. The registry may use a special Slot with name repositoryItemCharset, and sloType of nls for the RegistryEntry for storing the charset of the corresponding repository item. Value must be the one defined in [RFC 2277], [RFC 2278]. The repositoryItemCharset is optional because not all repository items require it.

E.4.2 Language information of repository item

Specifying only character set is not enough to tell which language is used in the repository item. A registry may use a special Slot with name repositoryItemLang, and sloType of nls to store that information. This attribute is optional because not all repository items require it. Value must be compliant 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 F  Terminology Mapping

While every attempt has been made to use the same terminology used in other works there are some terminology differences.

The following table shows the terminology mapping between this specification and that used in other specifications and working groups.

<table>
<thead>
<tr>
<th>This Document</th>
<th>OASIS</th>
<th>ISO 11179</th>
</tr>
</thead>
<tbody>
<tr>
<td>“repository item”</td>
<td>Registered Object</td>
<td></td>
</tr>
<tr>
<td>RegistryEntry</td>
<td>Registry Item</td>
<td>Administered</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Component</td>
</tr>
<tr>
<td>ExternalObject</td>
<td>Related Data</td>
<td>N/A</td>
</tr>
<tr>
<td>Object.ID</td>
<td>Raltermld</td>
<td></td>
</tr>
<tr>
<td>ExtrinsicObject.uri</td>
<td>ObjectLocation</td>
<td></td>
</tr>
<tr>
<td>ExtrinsicObject.objectType</td>
<td>DefnSource,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PrimaryClass,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SubClass</td>
<td></td>
</tr>
<tr>
<td>RegistryEntry.name</td>
<td>CommonName</td>
<td></td>
</tr>
<tr>
<td>Object.description</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>ExtrinsicObject.mimeType</td>
<td>MimeType</td>
<td></td>
</tr>
<tr>
<td>Versionable.majorVersion</td>
<td>partially to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>Versionable.minorVersion</td>
<td>partially to</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>RegistryEntry.status</td>
<td>RegStatus</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Terminology Mapping Table

10 References


The Report of the IAB Character Set Workshop held 29 February - 1
March, 1996,
C. Weider, C. Preston, K. Simonsen, H. Alvestrand, R. Atkinson, M.
http://www.cis.ohio-state.edu/htbin/rfc/rfc2130.html

[RFC 3023]  IETF (Internet Engineering Task Force). RFC 3023:
ftp://ftp.isi.edu/in-notes/rfc3023.txt

language(XML)1.0(Second Edition)
http://www.w3.org/TR/REC-xml

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: Scott Nieman
Company: Norstan Consulting
Street: 5101 Shady Oak Road
City, State, Postal Code: Minnetonka, MN 55343
Country: USA
Phone: 952.352.5889
Email: Scott.Nieman@Norstan

Vice Team Lead
Name: Yutaka Yoshida
Company: Sun Microsystems
Street: 901 San Antonio Road, MS UMPK17-102
City, State, Postal Code: Palo Alto, CA 94303
Country: USA
Phone: 650.786.5488
Email: Yutaka.Yoshida@eng.sun.com

Editor
Name: Farrukh S. Najmi
Company: Sun Microsystems
Street: 1 Network Dr., MS BUR02-302
City, State, Postal Code: Burlington, MA, 01803-0902
Country: USA
Phone: 781.442.0703
Email: najmi@east.sun.com
Copyright Statement

Copyright © ebXML 2000. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and
derivative works that comment on or otherwise explain it or assist in its
implementation may be prepared, copied, published and distributed, in whole or
in part, without restriction of any kind, provided that the above copyright notice
and this paragraph are included on all such copies and derivative works.
However, this document itself may not be modified in any way, such as by
removing the copyright notice or references to the Internet Society or other
Internet organizations, except as needed for the purpose of developing Internet
standards in which case the procedures for copyrights defined in the Internet
Standards process must be followed, or as required to translate it into languages
other than English.

The limited permissions granted above are perpetual and will not be revoked by
ebXML or its successors or assigns.

This document and the information contained herein is provided on an
"AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE
USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A
PARTICULAR PURPOSE.