ebXML Registry Services and Protocols Version 3.0

OASIS Standard, 2 May, 2005

Document identifier:

regrep-rs-3.0-os

Location:

http://docs.oasis-open.org/regrep-rs/v3.0/

Editors:

Name	Affiliation
Sally Fuger	Individual
Farrukh Najmi	Sun Microsystems
Nikola Stojanovic	RosettaNet

Contributors:

Name	Affiliation
Diego Ballve	Individual
Ivan Bedini	France Telecom
Kathryn Breininger	The Boeing Company
Joseph Chiusano	Booz Allen Hamilton
Peter Kacandes	Adobe Systems
Paul Macias	LMI Government Consulting
Carl Mattocks	CHECKMi
Matthew MacKenzie	Adobe Systems
Monica Martin	Sun Microsystems
Richard Martell	Galdos Systems Inc
Duane Nickull	Adobe Systems
Goran Zugic	ebXMLsoft Inc.

Abstract:

This document defines the services and protocols for an ebXML Registry

A separate document, ebXML Registry: Information Model [ebRIM], defines the types of metadata and content that can be stored in an ebXML Registry.

Status:

This document is an OASIS ebXML Registry Technical Committee Approved Draft Specification.

Committee members should send comments on this specification to the regrep@lists.oasis-open.org list. Others should subscribe to and send comments to the regrep-comment@lists.oasis-open.org list. To subscribe, send an email message to regrep-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the OASIS ebXML Registry TC web page (http://www.oasis-open.org/committees/regrep/).

Table of Contents

Illustration Index

1 Introduction

An ebXML Registry is an information system that securely manages any content type and the standardized metadata that describes it.

The ebXML Registry provides a set of services that enable sharing of content and metadata between organizational entities in a federated environment. An ebXML Registry may be deployed within an application server, a web server or some other service container. The registry MAY be available to clients as a public, semi-public or private web site.

This document defines the services provided by an ebXML Registry and the protocols used by clients of the registry to interact with these services.

A separate document, ebXML Registry: Information Model [ebRIM], defines the types of metadata and content that can be stored in an ebXML Registry.

1.1 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

1.2 Terminology

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF RFC 2119 [RFC2119].

The term *"repository item*" is used to refer to content (e.g., an XML document or a DTD) that resides in a repository for storage and safekeeping. Each repository item is described by a RegistryObject instance. The RegistryObject catalogs the RepositoryItem with metadata.

1.3 Notational Conventions

Throughout the document the following conventions are employed to define the data structures used. The following text formatting conventions are used to aide readability:

1.3.1 UML Diagrams

Unified Modeling Language [UML] diagrams are used as a way to concisely describe concepts. They are not intended to convey any specific Implementation or methodology requirements.

1.3.2 Identifier Placeholders

Listings may contain values that reference ebXML Registry objects by their id attribute. These id values uniquely identify the objects within the ebXML Registry. For convenience and better readability, these key values are replaced by meaningful textual variables to represent such id values.
For example, the placeholder in the listing below refers to the unique id defined for an example Service object:

```
<rim:Service id="${EXAMPLE_SERVICE_ID}">
```

1.3.3 Constants

Constant values are printed in the `Courier New font` always, regardless of whether they are defined by this document or a referenced document.

1.3.4 Bold Text

Bold text is used in listings to highlight those aspects that are most relevant to the issue being discussed. In the listing below, an example value for the contentLocator slot is shown in italics if that is what the reader should focus on in the listing:

```
<rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rim:RegistryObject:contentLocator">
...
</rim:Slot>
```

1.3.5 Example Values

These values are represented in *italic* font. In the listing below, an example value for the contentLocator slot is shown in italics:

```
<rim:Slot name="urn:oasis:names:tc:ebxml-
regrep:rim:RegistryObject:contentLocator">
       <rim:ValueList>
              <rim:Value>http://example.com/myschema.xsd</rim:Value>
       </rim:ValueList>
</rim:Slot>
```

1.4 XML Schema Conventions

This specification uses schema documents conforming to W3C XML Schema [Schema1] and normative text to describe the syntax and semantics of XML-encoded objects and protocol messages. In cases of disagreement between the ebXML Registry schema documents and schema listings in this specification, the schema documents take precedence. Note that in some cases the normative text of this specification imposes constraints beyond those indicated by the schema documents.

Conventional XML namespace prefixes are used throughout this specification to stand for their respective namespaces as follows, whether or not a namespace declaration is present in the example. The use of these namespace prefixes in instance documents is non-normative. However, for consistency and understandability instance documents SHOULD use these namespace prefixes.

1.4.1 Schemas Defined by ebXML Registry

Prefix	XML Namespace	Comments
`rim:`	urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0	This is the Registry Information Model namespace [ebRIM]. The prefix is generally elided in mentions of Registry Information Model elements in text.
`rs:`	urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0	This is the ebXML Registry namespace that defines base types for registry service requests and responses [ebRS]. The prefix is generally elided in mentions of ebXML Registry protocol-related elements in text.
`query:`	urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0	This is the ebXML Registry query namespace that is used in the query protocols used between clients and the QueryManager service [ebRS].

Prefix	XML Namespace	Comments
`lcm:`	urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0	This is the ebXML Registry Life Cycle Management namespace that is used in the life cycle management protocols used between clients and the LifeCycleManager service [ebRS].
`cms:`	urn:oasis:names:tc:ebxml-regrep:xsd:cms:3.0	This is the ebXML Registry Content Management Services namespace that is used in the content management protocols used between registry and pluggable content managent services [ebRS].

477

478 1.4.2 Schemas Used By ebXML Registry

479

Prefix	XML Namespace	Comments
`saml:`	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore]. The prefix is generally elided in mentions of SAML assertion-related elements in text.
`samlp:`	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore]. The prefix is generally elided in mentions of XML protocol-related elements in text.
`ecp:`	urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp	This is the SAML V2.0 Enhanced Client Proxy profile namespace, specified in this document and in a schema [SAMLECP-xsd].
`ds:`	http://www.w3.org/2000/09/xmldsig#	This is the XML Signature namespace [XMLSig].
`xenc:`	http://www.w3.org/2001/04/xmlenc#	This is the XML Encryption namespace [XMLEnc].
`SOAP-ENV:`	http://schemas.xmlsoap.org/soap/envelope	This is the SOAP V1.1 namespace [SOAP1.1].
`paos:`	urn:liberty:paos:2003-08	This is the Liberty Alliance PAOS (reverse SOAP) namespace.
`xsi:`	http://www.w3.org/2001/XMLSchema-instance	This namespace is defined in the W3C XML Schema specification [Schema1] for schema-related markup that appears in XML instances.
`wsse:`	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	This namespace is defined by the Web Services Security: SOAP Message Security 1.0 specification [WSS-SMS]. It is used by registry to secure soap message communication.

Prefix	XML Namespace	Comments
`wsu:`	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	This namespace is defined by the Web Services Security: SOAP Message Security 1.0 specification [WSS-SMS]. It is used by registry to secure soap message communication.

1.5 Registry Actors

This section describes the various actors who interact with the registry.

Actor	Description
Registry Operator	An organization that operates an ebXMl Registry and makes it's services available.
Registry Administrator	A privileged user of the registry that is responsible for performing administrative tasks necessary for the ongoing operation of the registry. Such a user is analogous to a "super user" that is authorized to perform *any* action.
Registry Guest	A user of the registry whose identity is not known to the registry. Such a user has limited privileges within the registry.
Registered User	A user of the registry whose identity is known to the registry as an authorized user of the registry.
Submitter	A user that submits content and or metadata to the registry. A Submitter MUST be a Registered User.
Registry Client	A software program that interacts with the registry using registry protocols.

1.6 Registry Use Cases

Once deployed, the ebXML Registry provides generic content and metadata management services and as such supports an open-ended and broad set of use cases. The following are some common use cases that are being addressed by ebXML Registry.

- Web Services Registry: publish, management, discovery and reuse of web service discriptions in WSDL, ebXML CPPA and other forms.
- Controlled Vocabulary Registry: Enables publish, management, discovery and reuse of controlled vocabularies including taxonomies, code lists, ebXML Core Components, XML Schema and UBL schema.
- Business Process Registry: Enables publish, management, discovery and reuse of Business Process specifications such as ebXML BPSS, BPEL and other forms.
- Electronic Medical Records Repository
- Geological Information System (GIS) Repository that stores GIS data from sensors

1.7 Registry Architecture

The following figure provides a simplified view of the architecture of the ebXML Registry.

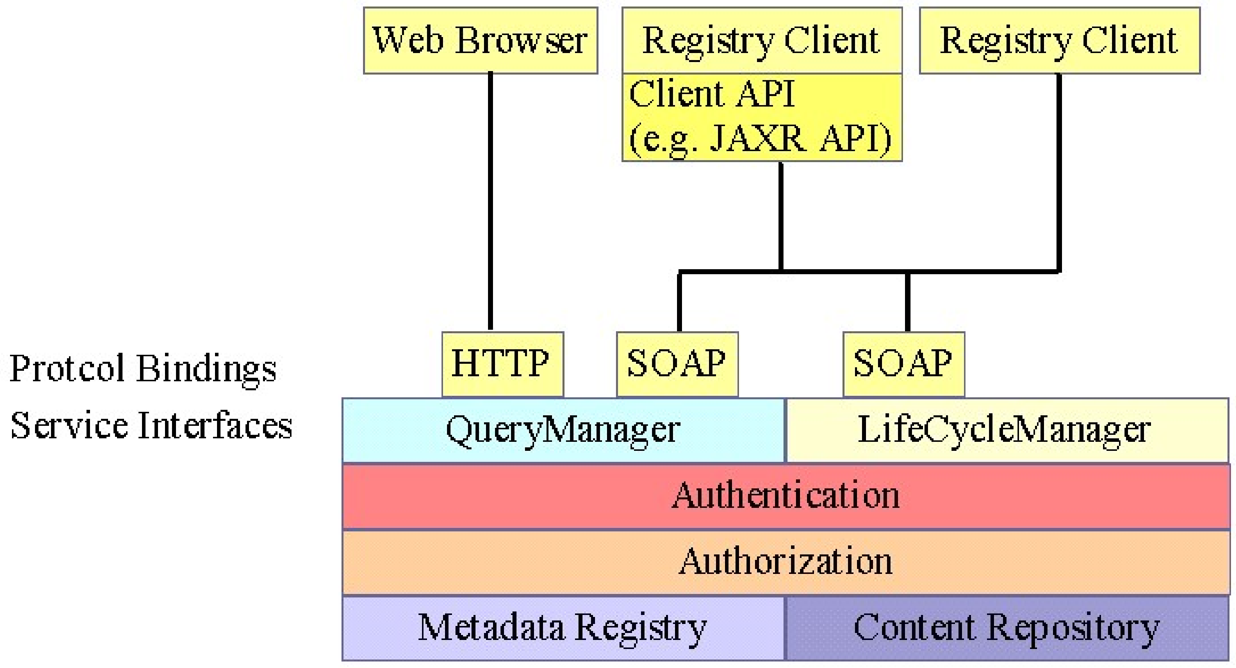


Figure 1: Simplified View of ebXML Registry Architecture

1.7.1 Registry Clients

A Registry Client is a software program that interacts with the registry using registry protocols. The Registry Client MAY be a Graphical User Interface (GUI), software service or agent. The Registry Client typically accesses the registry using SOAP 1.1 with Attachments [SwA] protocol.

A Registry Client may run on a client machine or may be a web tier service running on a server and may accessed by a web browser. In either case the Registry Client interacts with the registry using registry protocols.

1.7.1.1 Client API

A Registry client MAY access a registry interface directly. Alternatively, it MAY use a registry client API such as the Java API for XML Registries [JAXR] to access the registry. Client APIs such as [JAXR] provide programming convenience and are typically specific to a programming language.

1.7.2 Registry Service Interfaces

The ebXML Registry consists of the following service interfaces:

- A LifecycleManager interface that provides a collection of operations for end-to-end lifecycle management of metadata and content within the registry. This includes publishing, update, approval and deletion of metadata and content.
- A QueryManager interface that provides a collection of operations for the discovery and retrieval of metadata and content within the registry.

[RS-Interface-WSDL] provides an abstract (protocol neutral) definition of these Registry Service interfaces in WSDL format.

1.7.3 Service Interface: Protocol Bindings

This specification defines the following concrete protocol binding for the abstract service interfaces of the ebXML Registry:

- SOAP Binding that allows a Registry Client to access the registry using SOAP 1.1 with Attachments [SwA]. [RS-Bindings-WSDL] defines the binding of the abstract Registry Service interfaces to the SOAP protocol in WSDL format.
- HTTP Binding that allows a Web Browser client to access the registry using HTTP 1.1 protocol.

Additional bindings may be defined in the future as needed by the community.

1.7.4 Authentication and Authorization

A Registry Client SHOULD be authenticated by the registry to determine the identity associated with them. Typically, this is the identity of the user associated with the Registry Client. Once the registry determines the identity it MUST perform authorization and access control checks before permitting the Registry Client's request to be processed.

1.7.5 Metadata Registry and Content Repository

An ebXML Registry is both a registry of metadata and a repository of content. A typical ebXML Registry implementation uses some form of persistent store such as a database to store its metadata and content. Architecturally, registry is distinct from the repository. However, all access to the registry as well as repository is through the operations defined by the Registry Service interfaces.

2 Registry Protocols

This chapter introduces the registry protocols supported by the registry service interfaces. Specifically it introduces the generic message exchange patterns that are common to all registry protocols.

2.1 Requests and Responses

Specific registry request and response messages derive from common types defined in XML Schema in [RR-RS-XSD]. The Registry Client sends an element derived from **RegistryRequestType** to a registry, and the registry generates an element adhering to or deriving from **RegistryResponseType**, as shown next.

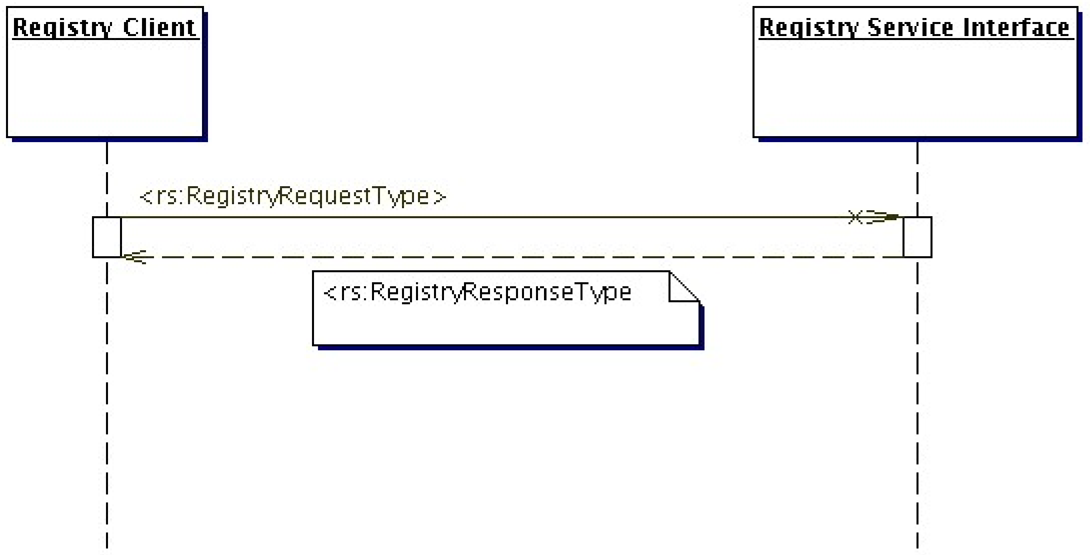


Figure 2: Registry Protocol Request-Response Pattern

Throughout this section, text mentions of elements and types are indicated with a namespace prefix. The namespace prefix conventions are defined in the “Introduction” chapter.

Each registry request is atomic and either succeeds or fails in entirety. In the event of success, the registry sends a RegistryResponse with a status of “Success” back to the client. In the event of failure, the registry sends a RegistryResponse with a status of “Failure” back to the client. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of “Unavailable” back to the client. Failure occurs when one or more Error conditions are raised in the processing of the submitted objects. Warning messages do not result in failure of the request.

2.1.1 RegistryRequestType

The RegistryRequestType type is used as a common base type for all registry request messages.

2.1.1.1 Syntax:

```
  <complexType name="RegistryRequestType">
    <sequence>
      <!-- every request may be extended using Slots. -->
      <element maxOccurs="1" minOccurs="0" name="RequestSlotList"
type="rim:SlotListType"/>
    </sequence>
    <attribute name="id" type="anyURI" use="required"/>
    <!--Comment may be used by requestor to describe the request. Used
in VersionInfo.comment-->
    <attribute name="comment" type="string" use="optional"/>
  </complexType>
  <element name="RegistryRequest" type="tns:RegistryRequestType"/>
```

2.1.1.2 Parameters:

comment: This parameter allows the requestor to specify a string value that describes the action being performed by the request. This parameter is used by the "Registry Managed Version Control" feature of the registry.

id: This parameter specifies a request identifier that is used by the corresponding response to correlate the response with its request. It MAY also be used to correlate a request with another related request. The value of the id parameter MUST abide by the same constraints as the value of the id attribute for the <rim:IdentifiableType> type.

RequestSlotList: This parameter specifies a collection of Slot instances. A RegistryReuqestType MAY include Slots as an extensibility mechanism that provides a means of adding additional attributes to the request in form of Slots. The use of registry implementation specific slots MUST be ignored silently by a registry that does not support such Slots and MAY not be interoperable across registry implementations.

2.1.1.3 Returns:

All RegistryRequests return a response derived from the common RegistryResponseType base type.

2.1.1.4 Exceptions:

The following exceptions are common to all registry protocol requests:

AuthorizationException: Indicates that the requestor attempted to perform an operation for which he or she was not authorized.

InvalidRequestException: Indicates that the requestor attempted to perform an operation that was semantically invalid.

SignatureValidationException: Indicates that a Signature specified for the request failed to validate.

TimeoutException: Indicates that the processing time for the request exceeded a registry specific limit.

UnsupportedCapabilityException: Indicates that this registry did not support the capability required to service the request.

In addition to above exceptions there are additional exceptions defined by [WSS-SMS] that a registry protocol request MUST return when certain errors occur during the processing of the <wsse:Security> SOAP Header element.

2.1.2 RegistryRequest

RegistryRequest is an element whose base type is RegistryRequestType. It adds no additional elements or attributes beyond those described in RegistryRequestType. The RegistryRequest element MAY be used by a registry to support implementation specific registry requests.

2.1.3 RegistryResponseType

The RegistryResponseType type is used as a common base type for all registry responses.

2.1.3.1 Syntax:

```
  <complexType name="RegistryResponseType">
    <sequence>
      <!-- every response may be extended using Slots. -->
      <element maxOccurs="1" minOccurs="0" name="ResponseSlotList"
type="rim:SlotListType"/>
      <element minOccurs="0" ref="tns:RegistryErrorList"/>
    </sequence>
    <attribute name="status" type="rim:referenceURI" use="required"/>
    <!-- id is the request if for the request for which this is a
response -->
    <attribute name="requestId" type="anyURI" use="optional"/>
  </complexType>
  <element name="RegistryResponse" type="tns:RegistryResponseType"/>
```

2.1.3.2 Parameters:

status: The status attribute is used to indicate the status of the request. The value of the status attribute MUST be a reference to a ClassificationNode within the canonical ResponseStatusType ClassificationScheme as described in [ebRIM]. A Registry MUST support the status types as defined by the canonical ResponseStatusType ClassificationScheme. The canonical ResponseStatusType ClassificationScheme may be extended by adding additional ClassificationNodes to it.

The following canonical values are defined for the ResponseStatusType ClassificationScheme:

- ***Success*** - This status specifies that the request was successful.
- ***Failure*** - This status specifies that the request encountered a failure. One or more errors MUST be included in the RegistryErrorList in this case or returned as a SOAP Fault.
- ***Unavailable*** – This status specifies that the response is not yet available. This may be the case if this RegistryResponseType represents an immediate response to an asynchronous request where the actual response is not yet available.

requestId*:* This parameter specifies the id of the request for which this is a response. It matches value of the id attribute of the corresponding RegistryRequestType.

ResponseSlotList*:* This parameter specifies a collection of Slot instances. A RegistryResponseType MAY include Slots as an extensibility mechanism that provides a means of adding dynamic attributes in form of Slots. The use of registry implementation specific slots MUST be ignored silently by a Registry Client that does not support such Slots and MAY not be interoperable across registry implementations.

RegistryErrorList**:** This parameter specifies an optional collection of RegistryError elements in the event that there are one or more errors that were encountered while the registry processed the request for this response. This is described in more detail in 6.9.4.

2.1.4 RegistryResponse

RegistryResponse is an element whose base type is RegistryResponseType. It adds no additional elements or attributes beyond those described in RegistryResponseType. RegistryResponse is used by many registry protocols as their response.

2.1.5 RegistryErrorList

A RegistryErrorList specifies an optional collection of RegistryError elements in the event that there are one or more errors that were encountered while the registry processed a request.

2.1.5.1 Syntax:

```
<element name="RegistryErrorList">
   <complexType>
     <complexContent>
       <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
         <sequence>
           <element ref="rs:RegistryError" maxOccurs="unbounded"/>
         </sequence>
         <attribute name="highestSeverity" type="rim:referenceURI" />
       </restriction>
     </complexContent>
   </complexType>
 </element>
```

2.1.5.2 Parameters:

highestSeverity: This parameter specifies the ErrorType for the highest severity RegistryError in the RegistryErrorList. Values for highestSeverity are defined by ErrorType in .

RegistryError: A RegistryErrorList has one or more RegistryErrors. A RegistryError specifies an error or warning message that is encountered while the registry processes a request. RegistryError is defined in 2.1.6.

2.1.6 RegistryError

A RegistryError specifies an error or warning message that is encountered while the registry processes a request.

2.1.6.1 Syntax:

```
  <element name="RegistryError">
    <complexType>
      <simpleContent>
        <extension base="string">
          <attribute name="codeContext" type="string" use="required"/>
          <attribute name="errorCode" type="string" use="required"/>
          <attribute default="urn:oasis:names:tc:ebxml-
regrep:ErrorSeverityType:Error" name="severity" type="rim:referenceURI"
/>
          <attribute name="location" type="string" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
```

2.1.6.2 Parameters:

codeContext: This attribute specifies a string that indicates contextual text that provides additional detail to the errorCode. For example, if the errorCode is InvalidRequestException the codeContext MAY provide the reason why the request was invalid.

errorCode: This attribute specifies a string that indicates the error that was encountered. Implementations MUST set this attribute to the Exception or Error as defined by this specification (e.g. InvalidRequestException).

severity: This attribute indicates the severity of error that was encountered. The value of the severity attribute MUST be a reference to a ClassificationNode within the canonical ErrorSeverityType ClassificationScheme as described in [ebRIM]. A Registry MUST support the error severity types as defined by the canonical ErrorSeverityType ClassificationScheme. The canonical ErrorSeverityType ClassificationScheme may be extended by adding additional ClassificationNodes to it.

The following canonical values are defined for the ErrorSeverityType ClassificationScheme:

- ***Error*** – An Error is a fatal error encountered by the registry while processing a request. A registry MUST return a status of Failure in the RegistryResponse for a request that encountered Errors during its processing.
- ***Warning*** – A Warning is a non-fatal error encountered by the registry while processing a request. A registry MUST return a status of Success in the RegistryResponse for a request that only encountered Warnings during its processing and encountered no Errors.

location: This attribute specifies a string that indicated where in the code the error occured. Implementations SHOULD show the stack trace and/or, code module and line number information where the error was encountered in code.

3 SOAP Binding

This chapter defines the SOAP protocol binding for the ebXML Registry service interfaces. The SOAP binding enables access to the registry over the SOAP 1.1 with Attachments [SwA] protocol. The complete SOAP Binding is described by the following WSDL description files:

- ebXML Registry Service Interfaces: Abstract Definition [RR-INT-WSDL]
- ebXML Registry Service Interfaces: SOAP Binding [RR-SOAPB-WSDL]
- ebXML Registry Service Interfaces: SOAP Service [RR-SOAPS-WSDL]

3.1 ebXML Registry Service Interfaces: Abstract Definition

In [RR-INT-WSDL], each registry Service Interface is mapped to an abstract WSDL portType as follows:

- A portType is defined for each Service Interface:

```
<portType name="QueryManagerPortType">
...
</portType>
<portType name="LifeCycleManagerPortType">
...
</portType>
```

- Within each portType an operation is defined for each protcol supported by the service interafce:

```
<portType name="QueryManagerPortType">
  <operation name="submitAdhocQuery">
  ...
  </operation>
</portType>
```

- Within each operation the the request and response message for the corresponding protocol are defined as input and output for the operation:

```
<portType name="QueryManagerPortType">
  <operation name="submitAdhocQuery">
    <input message="tns:msgAdhocQueryRequest"/>
    <output message="tns:msgAdhocQueryResponse"/>
  </operation>
</portType>
```

- For each message used in an operation a message element is defined that references the element corresponding to the registry protocol request or response message from the XML Schema for the registry service interface [RR-LCM-XSD], [RR-QM-XSD]:

```
<message name="msgAdhocQueryRequest">
  <part element="query:AdhocQueryRequest"
    name="partAdhocQueryRequest"/>
</message>
<message name="msgAdhocQueryRespone">
  <part element="query:AdhocQueryResponse"
    name="partAdhocQueryResponse"/>
</message>
```

3.2 ebXML Registry Service Interfaces SOAP Binding

In [RR-SOAPB-WSDL], a SOAP Binding is defined for the registry service interfaces as follows:

- For each portType corresponding to a registry service interface and defined in [RR-INT-WSDL] a <binding> element is defined which has name <ServiceInterfaceName>Binding
- The <binding> element references the portType defined in [RR-INT-WSDL] via its type attribute
- The <soap:binding> extension element uses the “document” style
- An operation element is defined for each protocol defined for the service interface. The operation name relates to the protocol request message.
- The <soap:operation> extension element has <input> and <output> elements that have <soap:body> elements with use="literal".

```
  <binding name="QueryManagerBinding"
type="interfaces:QueryManagerPortType">
    <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="submitAdhocQuery">
      <soap:operation soapAction="urn:oasis:names:tc:ebxml-
regrep:wsdl:registry:bindings:3.0:QueryManagerPortType#submitAdhocQuery
"/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
```

3.3 ebXML Registry Service Interfaces SOAP Service Template

In [RR-SOAPS-WSDL], a non-normative template is provided for a WSDL Service that uses the SOAP Binding from the registry service interfaces as follows:

- A single service element defines the concrete ebXML Registry SOAP Service. The template uses the name “ebXMLRegistrySOAPService”.
- The service element includes a port definitions, where each port corresponds with one of the service interfaces defined for the registry. Each port includes an HTTP URL for accessing that port specified by the location attribute of the <soap:address> element. The HTTP URL to the SOAP Service MUST conform to the pattern *<base URL>/soap* where *<base URL>* MUST be the same as the value of the *home* attribute of the instance of the Registry class defined by [ebRIM] that represents this registry.
- Each port definition also references a SOAP binding element described in the previous section.

```
  <service name="ebXMLRegistrySOAPService">
    <port binding="bindings:QueryManagerBinding"
name="QueryManagerPort">
      <soap:address location="http://your.server.com/soap"/>
    </port>
    <port binding="bindings:LifeCycleManagerBinding"
name="LifeCycleManagerPort">
      <soap:address location="http://your.server.com/soap"/>
    </port>
  </service>
```

3.4 Mapping of Exception to SOAP Fault

The registry protocols defined in this specification include the specification of Exceptions that a registry MUST return when certain exceptional conditions are encountered during the processing of the protocol request message. A registry MUST return Exceptions specified in registry protocol messages as SOAP Faults as described in this section. In addition a registry MUST conform to [WSI-BP] when generating the SOAP Fault. A registry MUST NOT sign a SOAP Fault message it returns.

The following table provides details on how a registry MUST map exceptions to SOAP Faults.

SOAP Fault Element	Description	Example
faultcode	The faultCode MUST be present and MUST be the name of the Exception qualified by the URN prefix: **`urn:oasis:names:tc:ebxml-regrep:rs:exception:`**	*urn:oasis:names:tc:ebxml-regrep:rs:exception:ObjectNotFoundException*
faultstring	The faultstring MUST be present and SHOULD provide some information explaining the nature of the exception.	*Object with id urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription not found in registry.*
detail	At least one detail element MUST be present. The detail element SHOULD include the stack trace and/or, code module and line number information where the Exception was encountered in code. If the Exception has nested Exceptions within it then the registry SHOULD include the nested exceptions as nested detail elements within the top level detail element.	
faultactor	At least one faultactor MUST be present. The first faultactor MUST be the base URL of the registry.	*http://example.server.com:8080/omar/registry*

Table 1: Mapping a Registry Exception to SOAP Fault

4 HTTP Binding

This chapter defines the HTTP protocol binding for the ebXML Registry abstract service interfaces. The HTTP binding enables access to the registry over the HTTP 1.1 protocol.

The HTTP interface provides multiple options for accessing RegistryObjects and RepositoryItems via the HTTP protocol. These options are:

- RPC Encoding URL: Allows client access to objects via a URL that is based on encoding a Remote Procedure Call (RPC) to a registry interface as an HTTP protocol request.
- Submitter Defined URL: Allows client access to objects via Submitter defined URLs.
- File Path Based URL: Allows clients access to objects via a URL based upon a file path derived from membership of object in a RegistryPackage membership hierarchy.

Each of the above methods has its advantages and disadvantages and each method may be better suited for different use cases as illustrated by table below:

HTTP Access Method	Advantages	Disadvantages
RPC Encoding URL	• The URL is constant and deterministic • Submitter need not explicitly assign URL	• The URL is long and not human-friendly to remember
Submitter Defined URL	• Very human-friendly URL • Submitter may assign any URL • The URL is constant and deterministic	• Submitter must explicitly assign URL • Requires additional resources in the registry
File Path Based URL	• Submitter need not explicitly assign URL • Intuitive URL that is based upon a familiar file / folder metaphor	• The URL is NOT constant and deterministic • Requires placing objects as members in RegistryPackages

Table 2: Comparison of HTTP Access Methods

4.1 HTTP Interface URL Pattern

The HTTP URLs used by the HTTP Binding MUST conform to the pattern *<base URL>/http/<url suffix>* where *<base URL>* MUST be the same as the value of the *home* attribute of the instance of the Registry class defined by [ebRIM] that represents this registry. The *<url suffix>* depends upon the HTTP Access Method and various request specific parameters that will be described later in this chapter.

4.2 RPC Encoding URL

The RPC Encoding URL method of the HTTP interface maps the operations defined by the abstract registry interfaces to the HTTP protocol using an RPC style. It defines how URL parameters are used to specify the interface, method and invocation parameters needed to invoke an operation on a registry interface such as the QueryManager interface.

The RPC Encoding URL method also defines how an HTTP response is used to carry the response generated by the operation specified in the request.

4.2.1 Standard URL Parameters

The following table specifies the URL parameters supported by RPC Encoding URLs. A Registry MAY implement additional URL parameters in addition to these parameters. Note that the URL Parameter names MUST be processed by the registry in a case-insensitive manner while the parameter values MUST be processed in a case-sensitive manner.

URL Parameter	Required	Description	Example
interface	YES	Defines the service interface that is the target of the request.	QueryManager
method	YES	Defines the method (operation)within the interface that is the target of the request.	getRegistryObject
param-<key>	NO	Defines named parameters to be passed into a method call. Note that some methods require specific parameters.	param-id= *urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription*

Table 3: Standard URL Parameters

4.2.2 QueryManager Binding

A registry MUST support a RPC Encoded URL HTTP binding to QueryManager service interface. To specify the QueryManager interface as its target, the *interface* parameter of the URL MUST be "QueryManager." In addition the following URL parameters are defined by the QueryManager HTTP Interface.

Method	Parameter	Return Value	HTTP Request Type
getRegistryObject	id	The RegistryObject that matches the specified id.	GET
getRepositoryItem	id	The RepositoryItem that matches the specified id. Note that a RepositoryItem may be arbitrary content (e.g. a GIF image).	GET

Table 4: RPC Encoded URL: Query Manager Methods

Note that in the examples that follow, name space declarations are omitted to conserve space. Also note that some lines may be wrapped due to lack of space.

4.2.2.1 Sample getRegistryObject Request

The following example shows a getRegistryObject request.

```
GET /http?interface=QueryManager&method=getRegistryObject&param-
id= urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription
HTTP/1.1
```

4.2.2.2 Sample getRegistryObject Response

The following example shows an ExtrinsicObject, which is a concrete sub-class of RegistryObject being returned as a response to the getRegistryObject method invocation.

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<ExtrinsicObject
   id =
"urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription"
   objectType="${OBJECT_TYPE}">
...
</ExtrinsicObject>
```

4.2.2.3 Sample getRepositoryItem Request

The following example shows a getRepositoryItem request.

```
GET /http?interface=QueryManager&method=getRepositoryItem&param-
id= urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription
HTTP/1.1
```

4.2.2.4 Sample getRepositoryItem Response

The following example assumes that the repository item was a Collaboration Protocol Profile as defined by [ebCPP]. It could return any type of content (e.g. a GIF image).

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: 555

<?xml version="1.0"?>
<CollaborationProtocolProfile>
...
</CollaborationProtocolProfile>
```

4.2.3 LifeCycleManager HTTP Interface

The RPC Encoded URL mechanism of the HTTP Binding does not support the LifeCycleManager interface. The reason is that the LifeCycleManager operations require HTTP POST which is already supported by the SOAP binding.

4.3 Submitter Defined URL

A Submitter MAY specify zero or more Submitter defined URLs for a RegistryObject or RepositoryItem. These URLs MAY then be used by clients to access the object using the GET request of the HTTP protocol. Submitter defined URLs serve as an alternative to the RPC Encoding URL defined by the HTTP binding for the QueryManager interface. The benefit of Submitter defined URLs is that objects are made accessible via a URL that is meaningful and memorable to the user. The cost of Submitter defined URLs is that the Submitter needs to specify the Submitter defined URL and that the Submitter defined URL takes additional storage resources within the registry.

Consider the examples below to see how Submitter defined URLs compare with the URL defined by the HTTP binding for the QueryManager interface.

Following is a sample URL defined by the HTTP binding for the QueryManager interface to access a RegistryObject that is an ExtrinsicObject describing a GIF image:

```
http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&met
hod=getRegistryObject&param-
id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription
```

The same RegistryObject (an ExtrinsicObject) may be accessed via the following Submitter defined URL:

```
http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.xml
```

Following is a sample URL defined by the HTTP binding for the QueryManager interface to access a repository item that is a GIF image:

```
http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&met
hod=getRepositoryItem&param-
id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription
```

The same repository item may be accessed via the following Submitter defined URL:

```
http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.jpg
```

4.3.1 Submitter defined URL Syntax

A Submitter MUST specify a Submitter defined URL as a URL suffix that is relative to the base URL of the registry. The URL suffix for a Submitter defined URL MUST be unique across all Submitter defined URLs defined for all objects within a registry.

The use of relative URLs is illustrated as follows:

- **Base URL for Registry:** http://localhost:8080/ebxml/registry
- **Implied Prefix URL for HTTP interface:** http://localhost:8080/ebxml/registry/http
- **Submitter Defined URL suffix:** /pictures/nikola/zeus
- **Complete URL:** http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus

4.3.2 Assigning URL to a RegistryObject

A Submitter MAY assign one or more Submitter defined URLs to a RegistryObject.

The Submitter defined URL(s) MAY be assigned by the Submitter using a canonical slot on the RegistryObject. The Slot is identified by the name:

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:locator
```

Each value in the collection of values for this Slot specifies a Submitter defined URL suffix for that RegistryObject. The registry MUST return the RegistryObject when the HTTP client sends an HTTP GET request whose URL matches any of the URLs specified within the locator Slot (if any) for that RegistryObject.

4.3.3 Assigning URL to a Repository Item

A Submitter MAY assign one or more Submitter defined URLs to a Repository Item.

The Submitter defined URL(s) may be assigned by the Submitter using a canonical slot on the ExtrinsicObject for the repository item. The Slot is identified by the name:

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:contentLocator
```

Each value in the collection of values for this Slot specifies a Submitter defined URL suffix for the RepositoryItem associated with the ExtrinsicObject. The registry MUST return the RepositoryItem when the HTTP client sends an HTTP GET request whose URL matches any of the URLs specified within the contentLocator slot (if any) for the ExtrinsicObject for that RepositoryItem.

4.4 File Path Based URL

The File Path Based URL mechanism enables HTTP clients to access RegistryObjects and RepositoryItems using a URL that is derived from the RegistryPackage membership hierarchy for the RegistryObject or RepositoryItem.

4.4.1 File Folder Metaphor

The RegistryPackage class as defined by [ebRIM] enables objects to be structurally organized by a RegistryPackage membership hierarchy. As such, a RegistryPackage serves a role similar to that of a Folder within the File and Folder metaphor that is common within filesystems in most operating systems. Similarly, the members of a RegistryPackage serve a role similar to the files within a folder in the File and Folder metaphor.

In this file-folder metaphor, a Submitter creates a RegistryPackage to create the functional equivalent of a folder and creates a RegistryObject to create the functional equivalent of a file. The Submitter adds a RegistryObjects as a member of a RegistryPackage to create the functional equivalent of adding a file to a folder.

4.4.2 File Path of a RegistryObject

Each RegistryObject has an implicit *file path*. The file path of a RegistryObject is a path structure similar to the Unix file path structure. The file path is composed of file path segments. Analogous to the Unix file path, the last segment within the file path represents the RegistryObject, while preceding segments represent the RegistryPackage(s) within the membership hierarchy of the RegistryObject. Each segment consists of the *name* of the RegistryPackage or the RegistryObject. Because the name attribute is of type InternationalString the path segment matches the name of an object within a specific locale.

4.4.2.1 File Path Example

Consider the example where a registry has a RegistryPackage hierarchy as illustrated below using the name of the objects in locale “en_US”:

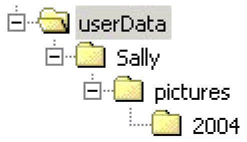


Figure 3: Example Registry Package Hierarchy

Now let us assume that the RegistryPackage named “2004” has an ExtrinsicObject named “baby.gif” for a repository item that is a photograph in the GIF format. In this example the file paths for various objects in locale “en_US” are shown in table below:

Object Name	File Path
userData	/userData
Sally	/userData/Sally
pictures	/userData/Sally/pictures
2004	/userData/Sally/pictures/2004
baby.gif	/userData/Sally/pictures/2004/baby.gif

Table 5: File Path Examples

Note that above example assumes that the RegistryPackage named userData is a root level package (not contained within another RegistryPackage).

4.4.3 Matching URL To Objects

A registry client MAY access RegistryObjects and RepositoryItems over the HTTP GET request using URL patterns that are based upon the File Path for the target objects. This section describes how a registry resolves File Path URLs specified by an HTTP client.

The registry MUST process each path segment from the beginning of the path to the end and for each path segment match the segment to the value attribute of a LocalizedString in the name attribute of a RegistryObject. For all but the last path segment, the matched RegistryObject MUST be a RegistryPackage. The last path segment MAY match any RegistryObject including a RegistryPackage. If any path segment fails to be matched then the URL is not resolvable by the File Path based URL method. When matching any segment other than the first segment the registry MUST also ensure that the matched RegistryObject is a member of the RegistryPackage that matches the previous segment.

4.4.4 URL Matches a Single Object

When a File Path based URL matches a single object the there are two possible responses.

- If the URL pattern does not end in a '/' character or the last segment does not match a RegistryPackage then the Registry MUST send as response an XML document that is the XML representation of the RegistryObject that matches the last segment. If the last segment matches an ExtrinsicObject then if the URL specifies the HTTP GET parameter with name 'getRepositoryItem' and value of 'true' then the registry MUST return as response the repository item associated with the ExtrinsicObject.

- If the URL pattern ends in a '/' character and the last segment matches a RegistryPackage then the Registry MUST send as response an HTML document that is the directory listing (section 4.4.6) of all RegistryObjects that are members of the RegistryPackage that matches the last segment.

4.4.5 URL Matches Multiple Object

A registry MUST show a partial Directory Listing of a Registry Package when a File Path based URL matches multiple objects.

A File Path based URL may match multiple objects if:

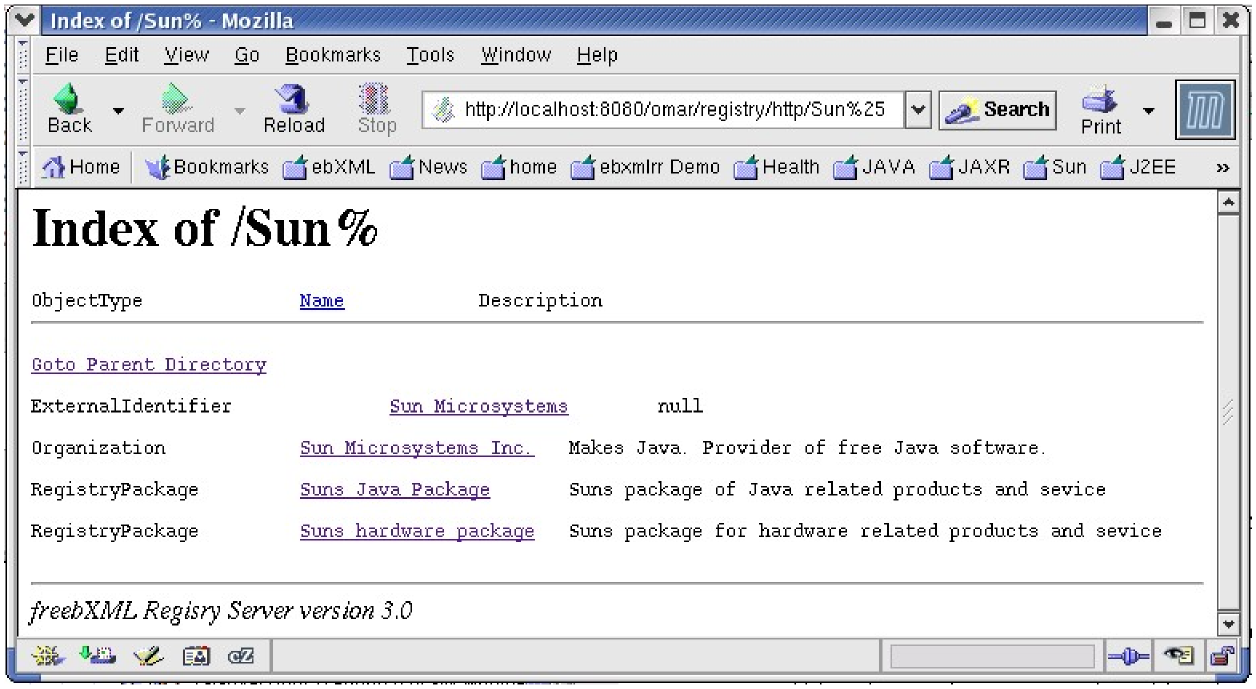
- Multiple objects with the same name exist in the same RegistryPackage
- The segment contains wildcard characters such as '%' or '?' to match the names of multiple objects within the same RegistryPackage. Note that wildcard characters must be URL encoded as defined by the HTTP protocol. For example the '%' character is encoded as '%25'.

4.4.6 Directory Listing

A registry MUST return a directory listing as a response under certain circumstances as describes earlier. The directory listing MUST show a list of objects within a specific RegistryPackage.

A registry SHOULD structure a directory listing such that each item in the listing provides information about a RegistryObject within the RegistryPackage. A registry MAY format its directory listing page in a registry specific manner. However, it is suggested that a registry SHOULD format it as an HTML page that minimally includes the objectType, name and description attributes for each RegistryObject in the directory listing.

Figure 4 shows a non-normative example of a directory listing that matches all root level objects that have a name that begins with ‘Sun’ (path /Sun%25).


Index of /Sun% - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/omar/registry/http/Sun%25
Search
Print
Home Bookmarks ebXML News home ebxmlrr Demo Health JAVA JAXR Sun J2EE

Index of /Sun%

```
ObjectType              Name              Description

Goto Parent Directory

ExternalIdentifier          Sun Microsystems        null

Organization            Sun Microsystems Inc.   Makes Java. Provider of free Java software.

RegistryPackage         Suns Java Package       Suns package of Java related products and sevice

RegistryPackage         Suns hardware package   Suns package for hardware related products and sevice
```

freebXML Regisry Server version 3.0

Figure 4: Example of a Directory Listing

4.4.7 Access Control In RegistryPackage Hierarchy

The ability to control who can add files and sub-folders to a folder is important in a file system. The same is true for the File Path Based URL mechanism.

A Submitter MAY assign a custom Access Control Policy to a Registry Package to create the functional equivalent of assigning access control to a folder in the file-folder metaphor. The custom Access Control Policy SHOULD use the "*reference*" action to control who can add RegistryObjects as members of the folder as described in [ebRIM].

4.5 URL Resolution Algorithm

Since the HTTP Binding supports multiple mechanisms to resolve an HTTP URL a registry SHOULD implement an algorithm to determine the correct HTTP Binding mechanism to resolve a URL.

This section gives a non-normative URL resolution algorithm that a registry SHOULD use to determine which of the various HTTP Binding mechanisms to use to resolve an HTTP URL.

Upon receiving an HTTP GET request a registry SHOULD first check if the URL is an RPC Encoded URL. This MAY be done by checking if the *interface* URL parameter is specified in the URL. If specified the registry SHOULD resolve the URL using the RPC Encoded URL method as defined by section 4.2. If the *interface* URL parameter is not specified then the registry SHOULD use the Submitter specified URL method to check if the URL is resolvable. If the URL is still unresolvable then the registry SHOULD check if the URL is resolvable using the File Path based URL method. If the URL is still unresolvable then the registry should return an HTTP 404 (NotFound) error as defined by the HTTP protocol.

4.6 Security Consideration

A registry MUST enforce all Access Control Policies including restriction on the READ action when processing a request to the HTTP binding of a service interface. This implies that a Registry MUST not resolve a URL to a RegistryObject or RepositoryItem if the client is not authorized to read that object.

4.7 Exception Handling

If a service interface method generates an Exception it MUST be reported in a `RegistryErrorList`, and sent back to the client within the HTTP response for the HTTP request.

When errors occur, the HTTP status code and message SHOULD correspond to the error(s) being reported in the `RegistryErrorList`. For example, if the `RegistryErrorList` reports that an object wasn't found, therefore cannot be returned, an appropriate error code SHOULD be 404, with a message of "ObjectNotFoundException". A detailed list of HTTP status codes can be found in [RFC2616]. The mapping between registry exceptions and HTTP status codes is currently unspecified.

5 Lifecycle Management Protocols

This section defines the protocols supported by Lifecycle Management service interface of the Registry. The Lifecycle Management protocols provide the functionality required by RegistryClients to manage the lifecycle of RegistryObjects and RepositoryItems within the registry.

The XML schema for the Lifecycle Management protocols is described in [RR-LCM-XSD].

5.1 Submit Objects Protocol

This SubmitObjects allows a RegistryClient to submit one or more RegistryObjects and/or repository items.

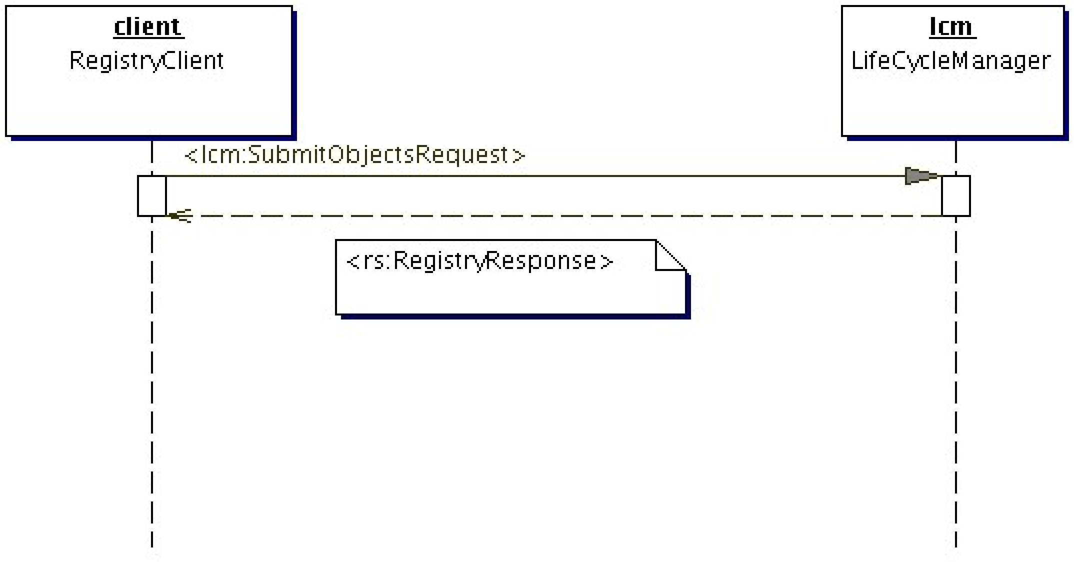


Figure 5: Submit Objects Protocol

5.1.1 SubmitObjectsRequest

The SubmitObjectsRequest is used by a client to submit RegistryObjects and/or repository items to the registry.

5.1.1.1 Syntax:

```
<element name="SubmitObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element ref="rim:RegistryObjectList"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

5.1.1.2 Parameters:

RegistryObjectList: This parameter specifies a collection of RegistryObject instances that are being submitted to the registry. The RegistryObjects in the list may be brand new objects being submitted to the registry or they may be current objects already existing in the registry. In case of existing objects the registry MUST treat them in the same manner as UpdateObjectsRequest and simply update the existing objects.

5.1.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4for details.

5.1.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

UnresolvedReferenceException: Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

UnsignedRepositoryItemException: Indicates that the requestor attempted to submit a RepositoryItem that was not signed.

QuotaExceededException: Indicates that the requestor attempted to submit more content than the quota allowed for them by the registry.

5.1.2 Unique ID Generation

As specified by [ebRIM], all RegistryObjects MUST have a unique id contained within the value of the id attribute. The id MUST be a valid URN and MUST be unique across all other RegistryObjects in the home registry for the RegistryObject.

A Submitter MAY optionally supply the id attribute for submitted objects. If the Submitter supplies the id and it is a valid URN and does not conflict with the id of an existing RegistryObject within the home registry then the registry MUST honor the Submitter-supplied id value and use it as the value of the id attribute of the object in the registry. If the id is not a valid URN then the registry MUST return an InvalidRequestException. If the id conflicts with the id of an existing RegistryObject within the home registry then the registry MUST return InvalidRequestException for an UpdateObjectsRequest and treat it as an Update action for a SubmitObjectsRequest.

If the client does not supply an id for a submitted object then the registry MUST generate a universally unique id. A registry generated id value MUST conform to the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID]:

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

5.1.3 ID Attribute And Object References

The id attribute of an object MAY be used by other objects to reference that object. Within a SubmitObjectsRequest, the id attribute MAY be used to refer to an object within the same 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 valid URN, in which case the id is permanently assigned to the object within the registry. Alternatively, the submitter MAY assign an arbitrary id that is not a valid URN as long as the id is a unique anyURI value within the request document. In this case the id serves as a linkage mechanism within the request document but MUST be 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 MAY contain an ObjectRef whose id attribute is the id of the object in the registry. This id is by definition a valid URN. An ObjectRef MAY be viewed as a proxy within the request for an object that is in the registry.

5.1.4 Audit Trail

The registry MUST create a single AuditableEvent object with eventType *Created* for all the RegistryObjects created by a SubmitObjectsRequest.

5.1.5 Sample SubmitObjectsRequest

The following example shows a simple SubmitObjectsRequest that submits a single Organization object to the registry. It does not show the complete SOAP Message with the message header and additional payloads in the message for the repository items.

```
<lcm:SubmitObjectsRequest>
  <rim:RegistryObjectList>
    <rim:Organization lid="${LOGICAL_ID}"
     id="${ID}"
     primaryContact="${CONTACT_USER_ID}">
      <rim:Name>
        <rim:LocalizedString value="Sun Microsystems Inc."
xml:lang="en-US"/>
      </rim:Name>
      <rim:Address city="Burlington" country="USA" postalCode="01867"
stateOrProvince="MA" street="Network Dr." streetNumber="1"/>
      <rim:TelephoneNumber areaCode="781" countryCode="1" number="123-
456" phoneType="office"/>
    </rim:Organization>
  </rim:RegistryObjectList>
</SubmitObjectsRequest>
```

5.2 The Update Objects Protocol

The UpdateObjectsRequest protocol allows a Registry Client to update one or more existing RegistryObjects and/or repository items in the registry.

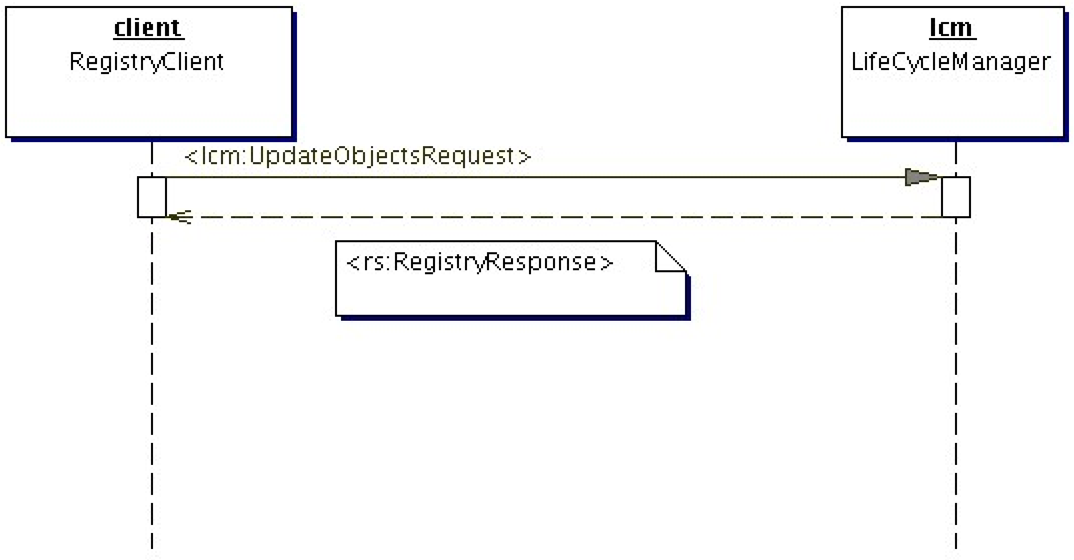


Figure 6: Update Objects Protocol

5.2.1 UpdateObjectsRequest

The UpdateObjectsRequest is used by a client to update RegistryObjects and/or repository items that already exist within the registry.

5.2.1.1 Syntax:

```
<element name="UpdateObjectsRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element ref="rim:RegistryObjectList"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

5.2.1.2 Parameters:

RegistryObjectList: This parameter specifies a collection of RegistryObject instances that are being updated within the registry. All immediate RegistryObject children of the RegistryObjectList MUST be current RegistryObjects already in the registry. RegistryObjects MUST include all required attributes, even those the user does not intend to change. A missing attribute MUST be interpreted as a request to set that attribute to NULL or in case it has a default value, the default value will be assumed. If this collection contains an immediate child RegistryObject that does not already exists in the registry, then the registry MUST return an InvalidRequestException. If the user wishes to submit a mix of new and updated objects then he or she SHOULD use a SubmitObjectsRequest.
If an ExtrinsicObject is being updated and no RepositoryItem is provided in the UpdateObjectsRequest then the registry MUST maintain any previously existing RepositoryItem associated with the original ExtrinsicObject with the updated ExtrinsicObject. If the client wishes to remove the RepositoryItem from an existing ExtrinsicObject they MUST use a RemoveObjectsRequest with deletionScope=DeleteRepositoryItemOnly.

5.2.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

5.2.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

UnresolvedReferenceException: Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

UnsignedRepositoryItemException: Indicates that the requestor attempted to submit a RepositoryItem that was not signed.

QuotaExceededException: Indicates that the requestor attempted to submit more content than the quota allowed for them by the registry.

5.2.2 Audit Trail

The registry MUST create a single AuditableEvent object with eventType *Updated* for all RegistryObjects updated via an UpdateObjectsRequest.

5.3 The Approve Objects Protocol

The Approve Objects protocol allows a client to approve one or more previously submitted RegistryObject objects using the LifeCycleManager service interface.

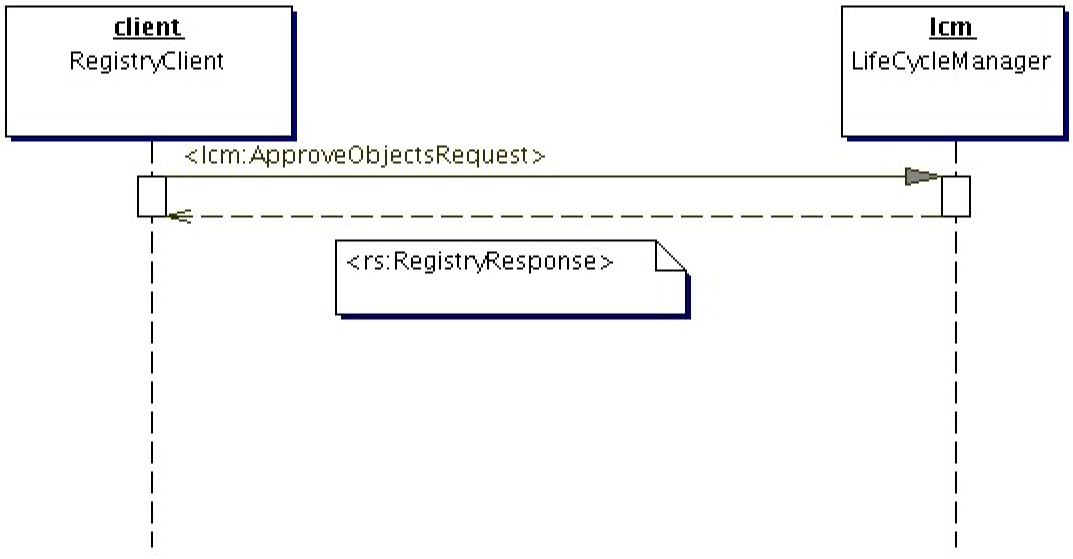


Figure 7: Approve Objects Protocol

5.3.1 ApproveObjectsRequest

The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject instances in the registry.

5.3.1.1 Syntax:

```
  <element name="ApproveObjectsRequest">
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
/>
            <element ref="rim:ObjectRefList" minOccurs="0"
maxOccurs="1" />
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

5.3.1.2 Parameters:

AdhocQuery: This parameter specifies a query. A registry MUST approve all objects that match the specified query in addition to any other objects identified by other parameters.

***ObjectRefList**:* This parameter specifies a collection of references to existing RegistryObject instances in the registry. A registry MUST approve all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

5.3.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

5.3.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

ObjectNotFoundException: Indicates that the requestor requested an object within the request that was not found.

5.3.2 Audit Trail

The registry MUST create a single AuditableEvent object with eventType *Approved* for all RegistryObject instance approved via an ApproveObjectsRequest.

5.4 The Deprecate Objects Protocol

The Deprecate Object protocol allows a client to deprecate one or more previously submitted RegistryObject instances using the LifeCycleManager service interface. Once a RegistryObject 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.

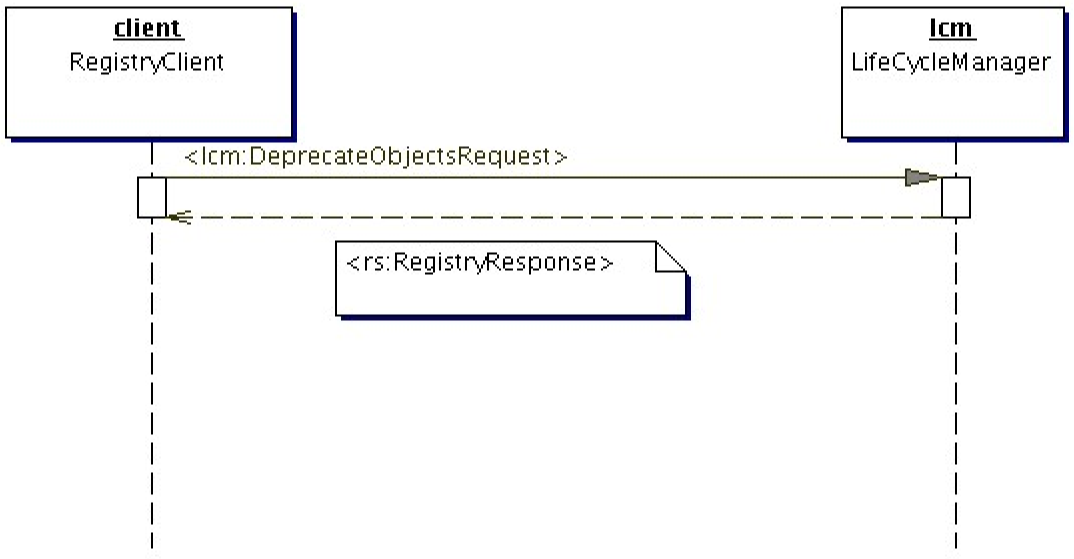


Figure 8: Deprecate Objects Protocol

5.4.1 DeprecateObjectsRequest

The DeprecateObjectsRequest is used by a client to deprecate one or more existing RegistryObject instances in the registry.

5.4.1.1 Syntax:

```
  <element name="DeprecateObjectsRequest">
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
/>
            <element ref="rim:ObjectRefList" minOccurs="0"
maxOccurs="1" />
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

5.4.1.2 Parameters:

AdhocQuery*:* This parameter specifies a query. A registry MUST deprecate all objects that match the specified query in addition to any other objects identified by other parameters.

ObjectRefList*:* This parameter specifies a collection of references to existing RegistryObject instances in the registry. A registry MUST deprecate all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

5.4.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

5.4.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

UnresolvedReferenceException: Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

5.4.2 Audit Trail

The registry MUST create a single AuditableEvent object with eventType *Deprecated* for all RegistryObject deprecated via a DeprecateObjectsRequest.

5.5 The Undeprecate Objects Protocol

The Undeprecate Objects protocol of the LifeCycleManager service interface allows a client to undo the deprecation of one or more previously deprecated RegistryObject instances. When a RegistryObject is undeprecated, it goes back to the Submitted status and new references (e.g. new Associations, Classifications and ExternalLinks) to that object can now again be submitted.

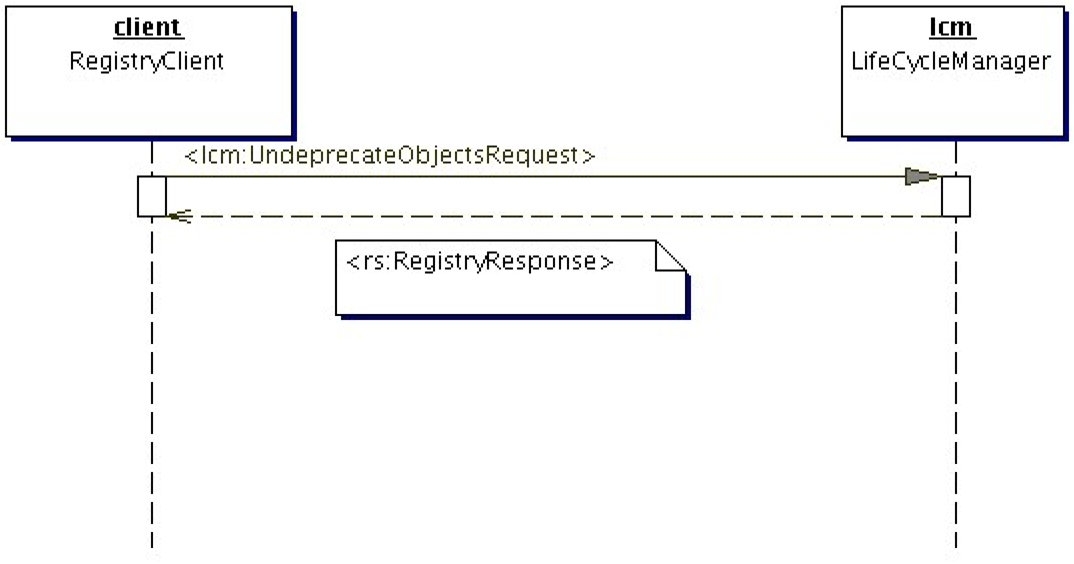


Figure 9: Undeprecate Objects Protocol

5.5.1 UndeprecateObjectsRequest

The UndeprecateObjectsRequest is used by a client to undeprecate one or more existing RegistryObject instances in the registry. The registry MUST silently ignore any attempts to undeprecate a RegistryObject that is not deprecated.

5.5.1.1 Syntax:

```
  <element name="UndeprecateObjectsRequest">
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
/>
            <element ref="rim:ObjectRefList" minOccurs="0"
maxOccurs="1" />
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
 </element>
```

5.5.1.2 Parameters:

AdhocQuery: This parameter specifies a query. A registry MUST undeprecate all objects that match the specified query in addition to any other objects identified by other parameters.

ObjectRefList: *This parameter specifies a collection of references to existing RegistryObject instances in the registry.* A registry MUST undeprecate all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

5.5.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

5.5.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

> *UnresolvedReferenceException:* Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

5.5.2 Audit Trail

The Registry Service MUST create a single AuditableEvent object with eventType *Undeprecated* for all RegistryObjects undeprecated via an UndeprecateObjectsRequest.

5.6 The Remove Objects Protocol

The Remove Objects protocol allows a client to remove one or more RegistryObject instances and/or repository items using the LifeCycleManager service interface.

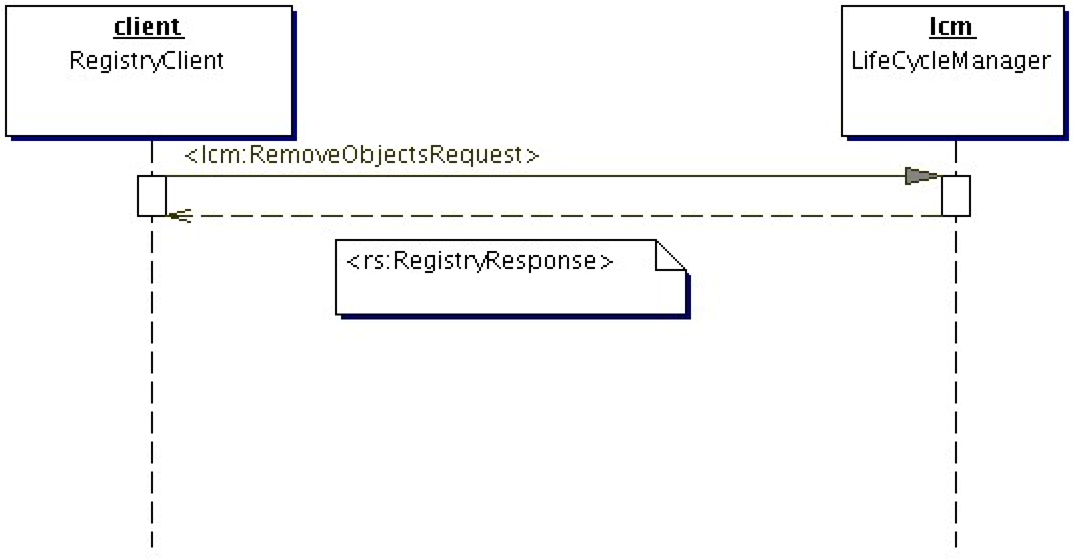


Figure 10: Remove Objects Protocol

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

5.6.1 RemoveObjectsRequest

The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject and/or repository items from the registry.

5.6.1.1 Syntax:

```
  <element name="RemoveObjectsRequest">
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
/>
```

```
            <element ref="rim:ObjectRefList" minOccurs="0"
maxOccurs="1" />
          </sequence>
          <attribute name="deletionScope"
default="urn:oasis:names:tc:ebxml-regrep:DeletionScopeType:DeleteAll"
type="rim:referenceURI" use="optional"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

5.6.1.2 Parameters:

deletionScope*:* This parameter indicates the scope of impact of the RemoveObjectsRequest. The value of the deletionScope attribute MUST be a reference to a ClassificationNode within the canonical DeletionScopeType ClassificationScheme as described in appendix A of [ebRIM]. A Registry MUST support the deletionScope types as defined by the canonical DeletionScopeType ClassificationScheme. The canonical DeletionScopeType ClassificationScheme may easily be extended by adding additional ClassificationNodes to it.

The following canonical ClassificationNodes are defined for the DeletionScopeType ClassificationScheme:

DeleteRepositoryItemOnly*:* This deletionScope specifies that the registry MUST delete the RepositoryItem for the specified ExtrinsicObjects but MUST NOT delete the specified ExtrinsicObjects. This is useful in keeping references to the ExtrinsicObjects valid. A registry MUST set the status of the ExtrinsicObject instance to *Withdrawn* in this case.

DeleteAll*:* This deletionScope specifies that the request MUST delete both the RegistryObject and the RepositoryItem (if any) for the specified objects. A RegistryObject can be removed using a RemoveObjectsRequest with deletionScope DeleteAll only if all references (e.g. Associations, Classifications, ExternalLinks) to that RegistryObject have been removed.

AdhocQuery*:* This parameter specifies a query. A registry MUST remove all objects that match the specified query in addition to any other objects identified by other parameters.

ObjectRefList*: This parameter specifies a collection of references to existing RegistryObject instances in the registry.* A registry MUST remove all objects that are referenced by this parameter in addition to any other objects identified by other parameters.

5.6.1.3 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

5.6.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

UnresolvedReferenceException*:* Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

ReferencesExistException: Indicates that the requestor attempted to remove a RegistryObject while references to it still exist. Note that it is valid to remove a RegistryObject and all RegistryObjects that refer to it within the same request. In such cases the ReferencesExistException MUST not be thrown.

5.7 Registry Managed Version Control

This section describes the version control features of the ebXML Registry. This feature is based upon [DeltaV]. The ebXML Registry provides a simplified façade that provides a small subset of [DeltaV] functionality.

5.7.1 Version Controlled Resources

All repository items in an ebXML Registry are implicitly version-controlled resources as defined by section 2.2.1 of [DeltaV]. No explicit action is required to make them a version-controlled resource.

In addition RegistryObject instances are also implicitly version-controlled resources. However, a registry may limit version-controlled resources to a sub-set of RegistryObject classes based upon registry specific policies.

Minimally, a registry implementing the version control feature SHOULD make the following types as version-controlled resources:

ClassificationNode

ClassificationScheme

Organization

ExtrinsicObject

RegistryPackage

Service

The above list is chosen to exclude all composed types and include most of remaining RegistryObject types for which there are known use cases requiring versioning.

5.7.2 Versioning and Object Identification

Each version of a RegistryObject is a unique object and as such has its own unique value for its id attribute as defined by [ebRIM].

5.7.3 Logical ID

All versions of a RegistryObject are logically the same object and are referred to as the `logical` RegistryObject. A logical RegistryObject is a tree structure where nodes are specific versions of the RegistryObject.

A specific version of a logical RegistryObject is referred to as a `RegistryObject instance`.

A RegistryObject instance MUST have a *Logical ID (LID)* to identify its membership in a particular logical RegistryObject. Note that this is in contrast with the `id` attribute that MUST be unique for each version of the same logical RegistryObject. A client may refer to the logical RegistryObject in a version independent manner using its LID.

A RegistryObject is assigned a LID using the `lid` attribute of the RegistryObject class. If the submitter assigns the lid attribute, she must guarantee that it is a globally unique URN. A registry MUST honor a valid submitter-supplied LID. If the submitter does not specify a LID then the registry MUST assign a LID and the value of the LID attribute MUST be identical to the value of the id attribute of the first (originally created) version of the logical RegistryObject.

5.7.4 Version Identification

An ebXML Registry supports independent versioning of both RegistryObject metadata as well as repository item content. It is therefore necessary to keep distinct version information for a RegistryObject instance and its repository item if it happens to be an ExtrinsicObject instance.

5.7.4.1 Version Identification for a RegistryObject

A RegistryObject MUST have a versionInfo attribute whose type is the VersionInfo class defined by ebRIM. The versionInfo attributes identifies the version information for that RegistryObject instance. A registry MUST not allow two versions of the same RegistryObject to have the same versionInfo.versionName attribute value.

5.7.4.2 Version Identification for a RepositoryItem

When a RegistryObject is an ExtrinsicObject with an associated repository item, the version identification for the repository item is distinct from the version identification for the ExtrinsicObject.

An ExtrinsicObject that has an associated repository item MUST have a contentVersionInfo attribute whose type is the VersionInfo class defined by ebRIM. The contentVersionInfo attributes identifies the version information for that repository item instance.

An ExtrinsicObject that does not have an associated repository item MUST NOT have a contentVersionInfo attribute defined.

A registry MUST allow two versions of the same ExtrinsicObject to have the same contentVersionInfo.versionName attribute value because multiple ExtrinsicObject versions MAY share the same RepositoryItem version.

5.7.5 Versioning of ExtrinsicObject and Repository Items

An ExtrinsicObject and its associated repository item may be updated independently and therefore versioned independently.

A registry MUST maintain separate version trees for an ExtrinsicObject and its associated repository item as described earlier.

Table 6 shows all the combinations for versioning an ExtrinsicObject and its repository item. After eliminating invalid or impossible combinations as well as those combinations where no action is needed, the only combinations that require versioning are showed in gray background rows. Of these there are only two unique cases (referred to as case A and B). Note that it is not possible to version a repository item without versioning its ExtrinsicObject.

ExtrinsicObject Exists	**RepositoryItem Exists**	**ExtrinsicObject Updated**	**RepositoryItem Updated**	**Comment**
No	No			Do nothing
No	Yes			Not possible
Yes	No	No	No	Do nothing
		No	Yes	Not possible
		Yes	No	Version ExtrinsicObject (case A)
		Yes	Yes	Not possible

Yes	Yes	No	No	Do nothing
		No	Yes	Not possible
		Yes	No	Version ExtrinsicObject (case A)
		Yes	Yes	Version ExtrinsicObject and RepositoryItem (case B)

Table 6: Versioning of ExtrinsicObject and Repository Item

5.7.5.1 ExtrinsicObject and Shared RepositoryItem

Because an ExtrinsicObject and its repository item are versioned independently (case B) it is possible for multiple versions of the ExtrinsicObject to share the same version of the repository item. In such cases the contentVersionInfo attributes MUST be the same across multiple version of the ExtrinsicObject.

5.7.6 Versioning and Composed Objects

When a registry creates a new version of a RegistryObject it MUST create copies of all composed[1] objects as new objects that are composed within the new version. This is because each version is a unique object and composed objects by definition are not shareable across multiple objects. Specifically, each new copy of a composed object MUST have a new id since it is a different object than the original composed object in the previous version.

A registry MUST not version composed objects.

5.7.7 Versioning and References

An object reference from a RegistryObject references a specific version of the referenced RegistryObject. When a registry creates a new version of a referenced RegistryObject it MUST NOT move refrences from other objects from the previous version to the new version of the referenced object. Clients that wish to always reference the latest versions of an object MAY use the Event Notification feature to update references when new versions are created and thus always reference the latest version.

A special case is when a SubmitObjectsRequest or an UpdateObjectRequest contains an object that is being versioned by the registry and the request contains other objects that reference the object being versioned. In such case, the registry MUST update all references within the submitted objects to the object being versioned such that those objects now reference the new version of the object being created by the request.

5.7.8 Versioning and Audit Trail

The canonical EventType ClassificationScheme used by the Audit Trail feature defines an Updated event type and then defines a Versioned event type as a child of the Updated event type ClassificationNode. The semantic are that a Versioned event type is specialization of the Updated event type.

A registry MUST use the Updated event type in the AuditableEvent when it updates a RegistryObject

[1] Composed object types are identified in figure 1 in [ebRIM] figure 1 as classes with composition or "solid diamond" relationship with RegistryObject type.

without creating a new version.

A registry MUST use the Versioned event type in the AuditableEvent when it creates a new version of a logical RegistryObject.

A registry MUST NOT use the Created event type in the AuditableEvent when it creates a new version of a logical RegistryObject.

5.7.9 Inter-versions Association

Within any single branch within the version tree for an object any given version implicitly supersedes the version immediately prior to it. Sometimes it may be necessary to explicitly indicate which version supersedes another version for the same object. This is especially true when two versions are siblings branch roots of the version tree for the same object.

A client MAY specify an Association between any two versions of an object within the objects version tree using the canonical associationType “Supersedes” to indicate that the sourceObject supersedes the target targetObject within the Association.

A client MUST NOT specify an Association between two version of an object using the canonical associationType “Supersedes” if the sourceObject is an earlier version within the same branch in the version tree than the targetObject as this violates the implicit “Supersedes” association between the two version.

Note that this section is functionally equivalent to the predecessor-set successor-set elements of the Version Properties as defined by [DeltaV].

5.7.10 Client Initiated Version Removal

An ebXML Registry MAY allow clients to remove specified versions of a RegistryObject. A client MAY delete older version of an object using the RemoveObjectsRequest by specifying the version by its unique id. Removing an ExtrinsicObject instance MUST remove its repository item if no other version references that repository item.

5.7.11 Registry Initiated Version Removal

The registry MAY prune older versions based upon registry specific administrative policies in order to manage storage resources.

5.7.12 Locking and Concurrent Modifications

This specification does not define a workspace feature with explicit checkin and checkout capabilities as defined by [DeltaV]. An ebXML Registry MAY support such features in an implementation specific manner.

This specification does not prescribe a locking or branching model. An implementation may choose to support an optimistic (non-locking) model. Alternatively or in addition, an implementation may support a locking model that supports explicit checkout and checkin capability. A future technical note or specification may address some of these capabilities.

5.7.13 Version Creation

The registry manages creation of new version of a RegistryObject or a repository item automatically. A registry that supports versioning MUST implicitly create a new version for a repository item if the repository item is updated via a SubmitObjectsRequest or UpdateObjectsRequest. In such cases it MUST also create a new version of its ExtrinsicObject.

If the client only wishes to update and version the ExtrisnicObject it may do so using an UpdateObjectsRequest without providing a repository item. In such cases the registry MUST assign the repository item version associated with the previous version of the ExtrinsicObject.

5.7.14 Versioning Override

A client MAY specify a *dontVersion* hint on a per RegistryObject basis when doing a submit or update of a RegistryObject. A registry SHOULD not create a new version for that RegistryObject when the dontVersion hint has value of “true”. The dontVersion hint MAY be specified as a canonical Slot with the following name:

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersion
```

The value of the dontVersion Slot, if specified, MUST be either “true” or “false”.

A client MAY specify a dontVersionContent hint on a per ExtrinsicObject basis when doing a submit or update of an ExtrinsicObject with a repository item. A registry SHOULD not create a new version for that repository item when the dontVersionContent hint has value of “true”. The dontVersionContent hint MAY be specified as a canonical Slot with the following name:

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersionContent
```

The value of the dontVersionContent Slot, if specified, MUST be either “true” or “false”.

A client MAY also specify the dontVersion and dontVersionContent Slots on the RegistryRequest using the <rs:RequstSlotList> element. A registry MUST treat these Slots when specified on the request as equivalent to being specified on every RegistryObject within the request. The value of these Slots as specified on the request take precedence over value of these Slots as specified on RegistryObjects within the request.

6 Query Management Protocols

This section defines the protocols supported by QueryManager service interface of the Registry. The Query Management protocols provide the functionality required by RegistryClients to query the registry and discover RegistryObjects and RepositoryItems.

The XML schema for the Query Management protocols is described in [RR-QUERY-XSD].

6.1 Ad Hoc Query Protocol

The Ad hoc Query protocol of the QueryManager service interface allows a client to query the registry and retrieve RegistryObjects and/or RepositoryItems that match the specified query.

A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that specifies a query in one of the query syntaxes supported by the registry.

The QueryManager sends an AdhocQueryResponse back to the client as response. The AdhocQueryResponse returns a collection of objects that match the query. The collection is potentially heterogeneous depending upon the query expression and request options.

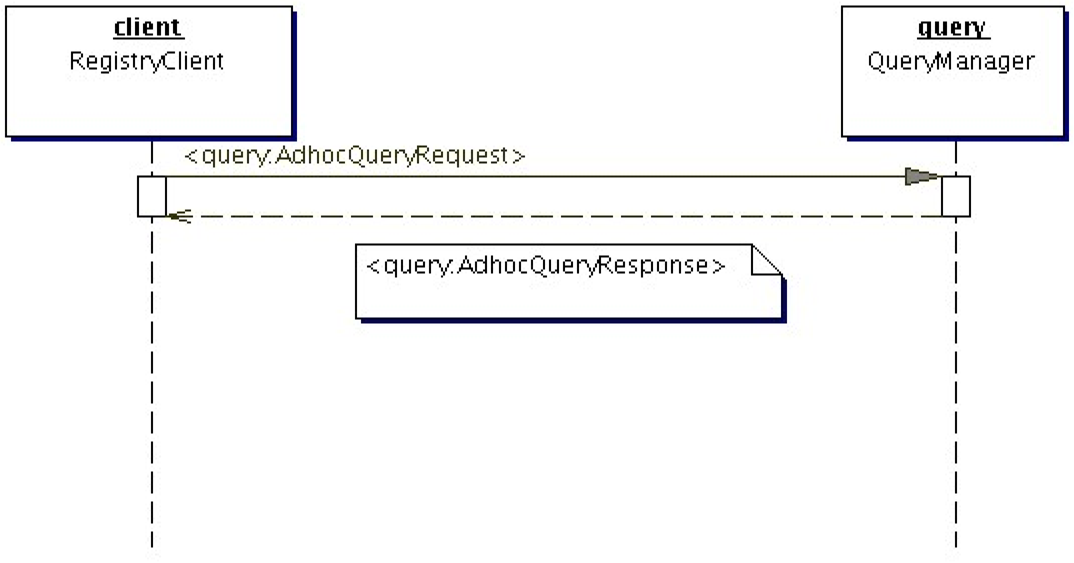


Figure 11: Ad Hoc Query Protocol

6.1.1 AdhocQueryRequest

The AdhocQueryRequest is used to submit a query to the registry.

6.1.1.1 Syntax:

```
<element name="AdhocQueryRequest">
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element maxOccurs="1" minOccurs="1"
                  ref="tns:ResponseOption"/>
          <element ref="rim:AdhocQuery" />
        </sequence>
        <attribute default="false" name="federated"
            type="boolean" use="optional"/>
        <attribute name="federation" type="anyURI" use="optional"/>
        <attribute default="0" name="startIndex" type="integer"/>
        <attribute default="-1" name="maxResults" type="integer"/>
      </extension>
    </complexContent>
  </complexType>
</element>
```

6.1.1.2 Parameters:

***AdhocQuery**:* This parameter specifies the actual query. It is decsribed in detail in section 6.1.3.

***federated**:* This optional parameter specifies that the registry must process this query as a federated query. By default its value is *false*. This value MUST be false when a registry routes a federated query to another registry in order to avoid an infinite loop in federated query processing.

***federation**:* This optional parameter specifies the id of the target Federation for a federated query in case the registry is a member of multiple federations. In the absence of this parameter a registry must route the federated query to all federations of which it is a member. This value MUST be unspecified when a registry routes a federated query to another registry in order to avoid an infinite loop in federated query processing.

maxResults: This optional parameter specifies a limit on the maximum number of results the client wishes the query to return. If unspecified, the registry SHOULD return either all the results, or in case the result set size exceeds a registry specific limit, the registry SHOULD return a sub-set of results that are within the bounds of the registry specific limit. See section 6.2.1 for an illustrative example.

ResponseOption: This required parameter allows the client to control the format and content of the AdhocQueryResponse generated by the registry in response to this request. See section 6.1.4 for details.

startIndex: This optional integer value is used to indicate which result *must* be returned as the first result when iterating over a large result set. The default value is 0, which returns the result set starting with index 0 (first result). See section 6.2.1 for an illustrative example.

6.1.1.3 Returns:

This request returns an AdhocQueryResponse. See section 6.1.2 for details.

6.1.1.4 Exceptions:

In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be returned:

InvalidQueryException: signifies that the query syntax or semantics was invalid. Client must fix the query syntax or semantic error and re-submit the query.

6.1.2 AdhocQueryResponse

The AdhocQueryResponse is sent by the registry as a response to an AdhocQueryRequest.

6.1.2.1 Syntax:

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

```
      <complexContent>
        <extension base="rs:RegistryResponseType">
          <sequence>
            <element ref="rim:RegistryObjectList" />
          </sequence>
          <attribute default="0" name="startIndex" type="integer"/>
          <attribute name="totalResultCount" type="integer"
use="optional"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

6.1.2.2 Parameters:

***RegistryObjectList**:* This is the element that contains the RegistryObject instances that matched the specified query.

startIndex: This optional integer value is used to indicate the index for the first result in the result set returned by the query, within the complete result set matching the query. By default, this value is 0. See section 6.2.1 for an illustrative example.

totalResultCount: This optional parameter specifies the size of the complete result set matching the query within the registry. When this value is unspecified, the client should assume it is the size of the result set contained within the result. See section 6.2.1 for an illustrative example.

6.1.3 AdhocQuery

A client specifies a <rim:AdhocQuery> element within an AdhocQueryRequest to specify the actual query being submitted.

6.1.3.1 Syntax:

```
  <complexType abstract="true" name="AdhocQueryType">
    <complexContent>
      <extension base="tns:RegistryObjectType">
        <sequence>
          <element ref="tns:QueryExpression"
             minOccurs="0" maxOccurs="1" />
        </sequence>
      </extension>
    </complexContent>
  </complexType>
  <element name="AdhocQuery" type="tns:AdhocQueryType"
      substitutionGroup="tns:RegistryObject" />
```

6.1.3.2 Parameters:

***queryExpression**:* This element contains the actual query expression. The schema for queryExpression is extensible and can support any query syntax supported by the registry.

6.1.4 ReponseOption

A client specifies a ResponseOption structure within an AdhocQueryRequest to indicate the format of the results within the corresponding AdhocQueryResponse.

6.1.4.1 Syntax:

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

6.1.4.2 Parameters:

returnComposedObjects*:* This optional parameter specifies whether the RegistryObjects returned should include composed objects as defined by Figure 1 in [ebRIM]. The default is to return all composed objects.

returnType*:* This optional enumeration parameter specifies the type of RegistryObject to return within the response. Values for returnType are as follows:

- ***ObjectRef*** - This option specifies that the AdhocQueryResponse MUST contain a collection of <rim:ObjectRef> elements. The purpose of this option is to return references to registry objects rather than the actual objects.
- ***RegistryObject*** - This option specifies that the AdhocQueryResponse MUST contain a collection of <rim:RegistryObject> elements.
- ***LeafClass*** - This option specifies that the AdhocQueryResponse MUST contain a collection of elements that correspond to leaf classes as defined in [RR-RIM-XSD].
- ***LeafClassWithRepositoryItem*** - This option is same as LeafClass option with the additional requirement that the response include the RepositoryItems, if any, for every <rim:ExtrinsicObject> element in the response.

If “returnType” specified does not match a result returned by the query, then the registry *must* use the closest matching semantically valid returnType that matches the result.

To illustrate, consider a case where OrganizationQuery is asked to return LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass option instead.

6.2 Iterative Query Support

The AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large result set matching a logical query by allowing multiple AdhocQueryRequest requests to be submitted such that each query requests a different subset of results within the result set. This feature enables the registry to handle queries that match a very large result set, in a scalable manner. The iterative query feature is accessed via the startIndex and maxResults parameters of the AdhocQueryRequest and the startIndex and totalResultCount parameters of the AdhocQueryResponse as described earlier.

The iterative queries feature is not a true Cursor capability as found in databases. The registry is not required to maintain transactional consistency or state between iterations of a query. Thus it is possible for new objects to be added or existing objects to be removed from the complete result set in between iterations. As a consequence it is possible to have a result set element be skipped or duplicated between iterations.

Note that while it is not required, an implementations MAY implement a transactionally consistent iterative query feature.

6.2.1 Query Iteration Example

Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a query that matches all 1007 Organizations. The user wishes to do the query iteratively such that Organizations are retrieved in chunks of 100. The following table illustrates the parameters of the AdhocQueryRequest and those of the AdhocQueryResponses for each iterative query in this example.

AdhocQueryRequest Parameters		**AdhocQueryResponse Parameters**		
startIndex	maxResults	startIndex	totalResultCount	# of Results
0	100	0	1007	100
100	100	100	1007	100
200	100	200	1007	100
300	100	300	1007	100
400	100	400	1007	100
500	100	500	1007	100
600	100	600	1007	100
700	100	700	1007	100
800	100	800	1007	100
900	100	900	1007	100
1000	100	1000	1007	7

6.3 Stored Query Support

The AdhocQuery protocol allow clients to submit queries that may be as general or as specific as the use case demands. As the queries get more specific they also get more complex. In these situations it is desirable to hide the complexity of the query from the client using parameterized queries stored in the registry. When using parameterized stored queries the client is only required to specify the identity of the query and the parameters for the query rather than the query expression itself.

Parameterized stored queries are useful to Registry Administrators because they provide a system wide mechanism for the users of the registry to share a set of commonly used queries.

Parameterized stored queries are useful to vertical standards because the standard can define domain specific parameterized queries and require that they be stored within the registry.

An ebXML Registry MUST support parameterized stored queries as defined by this section.

6.3.1 Submitting a Stored Query

A stored query is submitted using the standard SubmitObjectsRequest protocol where the object submitted is an AdhocQueryType instance.

6.3.1.1 Declaring Query Parameters

When submitting a stored query, the submitter MAY declare zero or more parameters for that query. A parameter MUST be declared using a parameter name that begins with the ‘$’ character followed immediately by a letter and then followed by any combination of letters and numbers. The following BNF defines how a parameter name MUST be declared.

```
QueryParameter := '$' [a-zA-Z] ( [a-zA-Z] | [0-9] )*
```

A query parameter MAY be used as a placeholder for any part of the stored query.

The following example illustrates how a parameterized stored query may be submitted:

```
<SubmitObjectsRequest>
  <rim:RegistryObjectList>
    <rim:AdhocQuery id="${QUERY_ID}">
      <rim:QueryExpression queryLanguage="${SQL_QUERY_LANG_ID}">
        SELECT * from $tableName ro, Name_ nm, Description d
        WHERE
        objectType = ''$objectType''
        AND (nm.parent = ro.id AND UPPER ( nm.value ) LIKE UPPER
( ''$name'' ) )
        AND (d.parent = ro.id AND UPPER ( d.value ) LIKE UPPER
( ''$description'' ) )
        AND (ro.id IN ( SELECT classifiedObject FROM Classification
WHERE classificationNode IN (  SELECT id
        FROM ClassificationNode WHERE path LIKE
''$classificationPath1%'' ) ))
      </rim:QueryExpression>
    </rim:AdhocQuery>
  </rim:RegistryObjectList>
</SubmitObjectsRequest>
```

Listing 1: Example of Stored Query Submission

The above query takes parameters *$objectType, $name, $description* and *$classificationPath1* and find all objects for that match specified objectType, name, description and classification.

6.3.1.2 Canonical Context Parameters

A query MAY contain one or more context parameters as defined in this section. Context parameters are special query parameters whose value does not need to be supplied by the client. Instead the value for a context parameter is supplied by the registry based upon the context within which the client request is being processed.

When processing a query, a registry MUST replace all context parameters present in the query with the context sensitive value for the parameter. A registry MUST ignore any context parameter values supplied by the client.

Context Parameter	Replacement Value
$currentUser	Must be replaced with the id attribute of the user associated with the query.
$currentTime	Must be replaced with the currentTime. The time format is same as the format defined for the timestamp attribute of AuditableEvent class.

6.3.2 Invoking a Stored Query

A stored query is invoked using the AdhocQueryRequest with the following constraints:

- The <rim:AdhocQuery> element MUST not contain a <rim:queryExpression> element.

- The <rim:AdhocQuery> element's id attribute value MUST match the id attribute value of the stored query.
- The <rim:AdhocQuery> element MAY have a Slot for each non-context parameter defined for the stored query being invoked. These Slots provide the value for the query parameters.

6.3.2.1 Specifying Query Invocation Parameters

A stored query MAY be defined with zero or more parameters. A client may specify zero or more of the parameters defined for the stored query when submitting the AdhocQueryRequest for the stored query. It is important to note that the client MAY specify fewer parameters than those declared for the stored query. A registry MUST prune any predicates of the stored query that contain parameters that were not supplied by the client during invocation of the stored query.

In essence, the client may narrow or widen the specificity of the search by supplying more or less parameters.

A client specifies a query invocation parameter by using a Slot whose name matches the parameter name and whose value MUST be a single value that matches the specified value for the parameter.

A registry MUST ignore any parameters specified by the client for a stored query that do not match the parameters defined by the stored query.

The following listing shows an example of how the stored query shown earlier is invoked. It shows:

- The stored query being identified by the value of the id attribute of the <rim:AdhocQuery> element.
- The value for the $name parameter being supplied
- The value of other parameters defined by the query not being supplied. This indicates that the client does not wish to use those parameters as serach criterea.

```
<AdhocQueryRequest>
  <query:ResponseOption returnComposedObjects="true"
returnType="LeafClassWithRepositoryItem"/>

  <rim:AdhocQuery id="${STORED_QUERY_ID}">
    <rim:Slot name="$name">
      <rim:ValueList>
        <rim:Value>%ebXML%</rim:Value>
      </rim:ValueList>
    </rim:Slot>
  </rim:AdhocQuery>
</AdhocQueryRequest>
```

Listing 2: Example of Stored Query Invocation

6.3.3 Response to Stored Query Invocation

A registry MUST send a standard AdhocQueryResponse when a client invokes a stored query using an AdhocQueryRequest.

6.3.4 Access Control on a Stored Query

A stored query is a RegistryObject. Like all RegistryObjects, access to the stored query is governed by the Access Control Policy defined the stored query. By default a stored query is assigned the default Access Control Policy that allows any client to read and invoke that query and only the owner of the query and the Registry Administrator role to update or delete the query. The owner of the query may define a custom Access Control Policy for the query that restricts the visibility of the query, and ability to invoke it, to specific users, roles or groups. Thus the owner of the query or the Registry Administrator may control *who* gets to invoke *which* stored queries.

6.3.5 Canonical Query: Get Client's User Object

A registry MUST support a canonical stored query with `id="urn:oasis:names:tc:ebxml-regrep:query:GetCallersUser"`.

This query MUST return the User object associated with the client invoking the stored query. The client MUST not provide any parameters for this query. The stored query SHOULD use the canonical context parameter $currentUser.

The following is a non-normative example of a stored SQL query that MAY be used by a registry for this canonical stored query:

```
<rim:AdhocQuery id="urn:oasis:names:tc:ebxml-
regrep:query:GetCallersUser">
  <rim:QueryExpression
    queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:SQL-
92">
    SELECT u.* FROM User u WHERE u.id = $currentUser;
  </rim:QueryExpression>
</rim:AdhocQuery>
```

Note that a registry MAY use an equivalent stored filter query instead of a stored SQL query.

6.4 SQL Query Syntax

An ebXML Registry MAY support SQL as a supported query syntax within the <rim:queryExpression> element of AdhocQueryRequest. This section normatively defines the SQL syntax that an ebXML Registry MAY support. Note that the support for SQL syntax within a registry does not imply a requirement that the registry must use a relational database in its implementation.

The registry SQL syntax is a proper subset of the "SELECT" statement of Intermediate level SQL as defined by ISO/IEC 9075:1992, Database Language SQL [SQL].

The terms below enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax conforms to the <query specification> with the following additional restrictions:

1. A **<derived column>** MAY NOT have an **<as clause>**.
2. **A <table expression>** does not contain the optional **<group by clause>** and **<having clause>** clauses.
3. A **<table reference>** can only consist of **<table name>** and **<correlation name>.**
4. A **<table reference>** does not have the optional AS between **<table name>** and **<correlation name>.**
5. 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.

As defined by [SQL], a registry MUST process table names and attribute names in a case insensitive manner.

6.4.1 Relational Schema for SQL Queries

The normative Relational Schema definition that is the target of registry SQL queries can be found at the following location on the web:

http://www.oasis-open.org/committees/regrep/documents/3.0/sql/database.sql

6.4.2 SQL Query Results

The result of an SQL query resolves to a collection of objects within the registry. It never resolves to partial attributes. The objects related to the result set may be returned as an ObjectRef, RegistryObject or leaf class depending upon the returnType attribute of the responseOption parameter specified by the client on the AdHocQueryRequest. The entire result set is returned as an <rim:RegistryObjectList>.

6.5 Filter Query Syntax

This section normatively defines an XML syntax for querying an ebXML Registry called *Filter Query* syntax. An ebXML Registry MUST support the Filter Query syntax as a supported query syntax within the <rim:queryExpression> element of AdhocQueryRequest.

The Filter Query syntax is defined in [RR-QUERY-XSD] and is derived from a mapping from [ebRIM] to XML Schema following certain mapping patterns.

The Filter Query operational model views the network of RegistryObjects in the registry as a virtual XML document and a query traverses a specified part of the tree and prunes or filters objects from the virtual document using filter expressions and ultimately returns a collection of objects that are left after filtering out all objects that do not match the filters specified in the query.

Unlike SQL query syntax, the filter query syntax does not support joins across classes. This constrains the expressive capabilities of the query and may also be somehat less efficient in processing.

6.5.1 Filter Query Structure

The <rim:queryExpression> element of AdhocQueryRequest MUST contain a *Query* element derived from the <query:RegistryObjectQueryType> type.

A Query element MAY contain a <query:PrimaryFilter> element and MAY contain additional Filter, Branch and Query elements within it as shown in the asbtract example below. The normative schema is defined by [RR-QUERY-XSD].

```
<${QueryElement}>
  <PrimaryFilter ... />
  <${OtherFilterElement} ... />
  <${BranchElement} .../>
  <${QueryElement} ... />
</${QueryElement}>
```

The role of Query, Filter and Branch elements will be defined next.

6.5.2 Query Elements

A Query element is the top level element in the Filter Query syntax to query the registry. The [RR-QUERY-XSD] XML Schema defines a Query element for the RegistryObject class and all its descendant classes as defined by [ebRIM] using the following pattern:

- For each class in model descendant from RegistryObject class define a complexType with name <class>QueryType. For example there is an OrganizationQueryType complexType defined for the Organization class in [ebRIM].
- The QueryType of a descendant of RegistryObject class MUST extend the QueryType for its super class. For example the OrganizationQueryType extends the RegistryObjectQueryType.
- For RegistryObject class and each of its descendants define an element with name <class>Query and with type <class>QueryType. For example the OrganizationQuery element is defined with type OrganizationQueryType.

The class associated with a Query element is referred to as the *Query domain class*.

The following example shows the Query syntax where the Query domain class is the Organization class defined by [ebRIM]:

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

```
      <extension base="tns:RegistryObjectQueryType">
        ...Relevant Filters, Queries and Branches are defined here...
      </extension>
    </complexContent>
  </complexType>
  <element name="OrganizationQuery" type="tns:OrganizationQueryType"/>
```

A Query element MAY have Filter, Branch or nested Query Elements. These are described in subsequent sections.

6.5.3 Filter Elements

A Query element MAY contain one or more Filter sub-elements. A Filter element is used to *filter* or select a subset of instances of a specific [ebRIM] class. The class that a Filter filters is referred to as the *Filter domain class*. A Filter element specifies a restricted predicate clause over the attributes of the Filter domain class.

[RR-QUERY-XSD] XML Schema defines zero or more Filter elements within a Query element definition using the following pattern:

- ***PrimaryFilter***: A Filter element is defined within the RegistryObjectQueryType with name *PrimaryFilter*. This Filter is used to filter the instances of the Query domain class based upon the value of its primitive attributes. The cardinality of the Filter element is zero or one. The *PrimaryFilter* element is inherited by all descendant QueryTypes of RegistryObjectQueryType.
- ***Additional Filters:*** Additional Filters in a Query element used to filter the instances of the Query domain class based upon whether the candidate domain class instance has a referenced object that satisfies the additional filter.
  Additional filter elements are defined for those attributes of the Query domain class that satisfy all of the following criterea:
  - The attribute's domain is not a primitive type (e.g. string, float, dateTime, int etc.).
  - The attribute's domain class is not RegistryObject or its descendant.
  - The attribute's domain class does not have any reference attributes (use Branch or sub-Query if attribute's domain class has reference attributes).

    The attribute for which the Filter is defined is referred to as the Filter domain attribute. The domain class of the Filter domain attribute is the Filter domain class for such Filters. This type of Filter is used to filter the instances of the Query domain class based upon the attribute values within the Filter domain class.
  - The name of the Filter element is <Filter Domain Attribute Name>Filter.
  - The type of the Filter element is the FilterType complex type that is decsribed in 6.5.3.1.
  - The cardinality of the Filter element matches the cardinality of the Filter domain attribute in the Query domain class.

The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to define Filters for the OrganizationQueryType for the Organization class defined by [ebRIM].

```
  <complexType name="OrganizationQueryType">
    <complexContent>
      <extension base="tns:RegistryObjectQueryType">
        <sequence>
          <element maxOccurs="unbounded" minOccurs="0"
            name="AddressFilter" type="tns:FilterType"/>
          <element maxOccurs="unbounded" minOccurs="0"
            name="TelephoneNumberFilter" type="tns:FilterType"/>
          <element maxOccurs="unbounded" minOccurs="0"
            name="EmailAddresseFilter" type="tns:FilterType"/>
          ...Branches and sub-Queries go here...
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

The following UML class diagram describing the Filter class structure as defined in [RR-QUERY-XSD] XML Schema. Note that the classes whose name ends in “*Type”* map to complexTypes and other Filter classes map to elements in the [RR-QUERY-XSD] XML Schema.

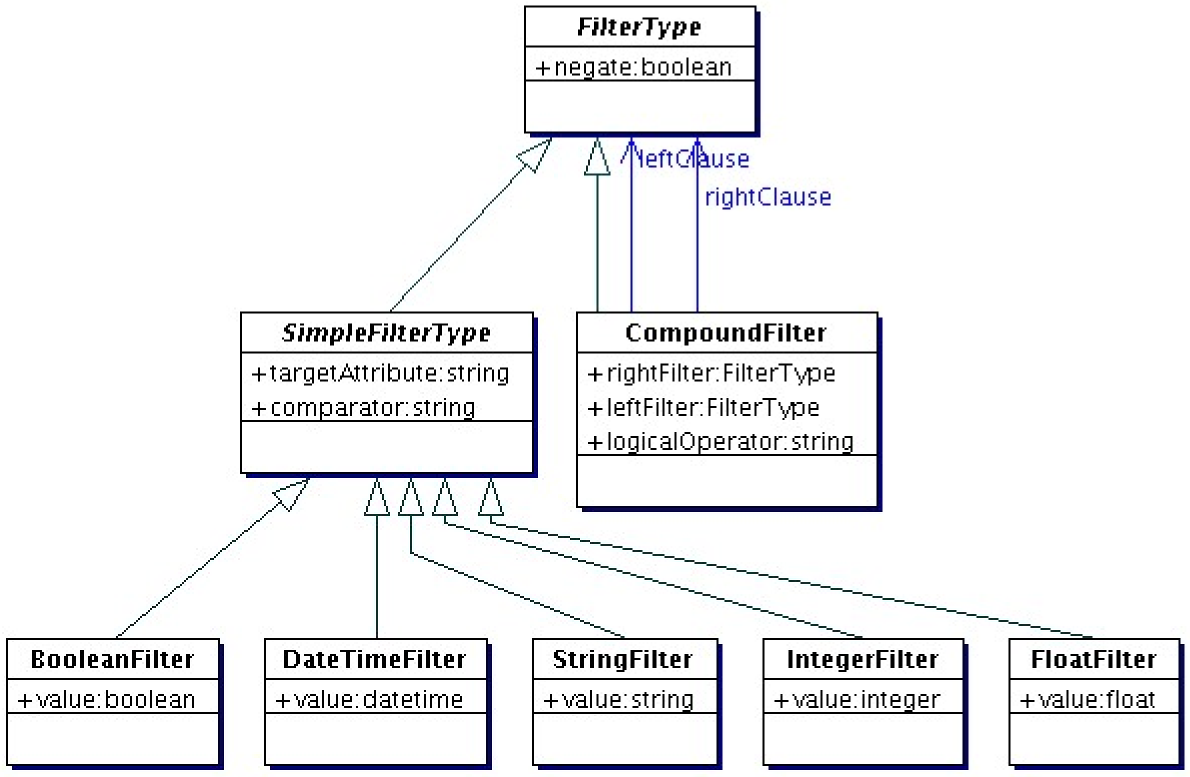


Figure 12: Filter Type Hierarchy

6.5.3.1 FilterType

The FilterType is an abstract complexType that is the root type in the inheritence hierarchy for all Filter types.

6.5.3.1.1 Parameters:

negate*:* This parameter specifies that the boolean value that the Filter evaluates to MUST be negated to complete the evaluation of the filter. It is functionally equivalent to the NOT operator in SQL syntax.

6.5.3.2 SimpleFilterType

The SimpleFilter is the abstract base type for several concrete Filter types defined for primitive type such as boolean, float, integer and string.

6.5.3.2.1 Parameters:

***domainAttribute**:* This parameter specifies the attribute name of a primitive attribute within the Filter domain class. A registry MUST return an InvalidQueryException if this parameter's value does not match the name of primitive attribute within the Filter domain class. A registry MUST perform the attribute name match in a case insensitive manner.

***comparator**:* This parameter specifies the comparison operator for comparing the value of the attribute with the value supplied by the filter. The following comparators are defined:

- LE: abbreviation for LessThanOrEqual
- LT: abbreviation for LessThan
- GE: abbreviation for GreaterThanOrEqual
- GT: abbreviation for GreaterThan
- EQ: abbreviation for Equal
- NE: abbreviation for NotEqual
- Like: Same as LIKE operator in SQL-92. MUST only be used in StringFilter.
- NotLike: Same as NOT LIKE operator in SQL-92. MUST only be used in StringFilter.

6.5.3.3 BooleanFilter

The BooleanFilter MUST only be used for matching primitive attributes whose domain is of type boolean.

6.5.3.3.1 Parameters:

***value**:* This parameter specifies the value that MUST be compared with the attribute value being tested by the Filter. It MUST be a boolean value.

The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the ClassificationScheme class defined by [ebRIM]:

```
<BooleanFilter
      domainAtribute="isInternal" comparator="EQ" value="true"/>
```

6.5.3.4 FloatFilter

The FloatFilter MUST only be used for matching primitive attributes whose domain is of type float.

6.5.3.4.1 Parameters:

***value**:* This parameter specifies the value that MUST be compared with the attribute value being tested by the Filter. It MUST be a float value.

The following example shows the use of a FloatFilter to match fictitious *amount* float attribute since [ebRIM] currently has no float attributes defined:

```
<FloatFilter
      domainAtribute="amount" comparator="GT" value="9.99"/>
```

6.5.3.5 IntegerFilter

The IntegerFilter MUST only be used for matching primitive attributes whose domain is of type integer.

6.5.3.5.1 Parameters:

***value**:* This parameter specifies the value that MUST be compared with the attribute value being tested by the Filter. It MUST be an integer value.

The following example shows the use of a BooleanFilter to match a fictitious *count* integer attribute since [ebRIM] currently has no integer attributes defined:

```
<IntegerFilter
      domainAtribute="amount" comparator="LT" value="100"/>
```

6.5.3.6 DateTimeFilter

The DateTimeFilter MUST only be used for matching primitive attributes whose domain is of type datetime.

6.5.3.6.1 Parameters:

***value**:* This parameter specifies the value that MUST be compared with the attribute value being tested by the Filter. It MUST be a datetime value.

The following example shows the use of a DateTimeFilter to match a the *timestamp* attribute of the Auditable class defined by [ebRIM] where the timestamp value is greater than (later than) the specified datetime value:

```
<DateTimeFilter
      domainAtribute="timestamp"
      comparator="GT" value="1997-07-16T19:20+01:00"/>
```

6.5.3.7 StringFilter

The StringFilter MUST only be used for matching primitive attributes whose domain is of type string.

6.5.3.7.1 Parameters:

***value**:* This parameter specifies the value that MUST be compared with the attribute value being tested by the Filter. It MUST be a string value.

The following example shows the use of a StringFilter to match a the *firstName* attribute of the Person class defined by [ebRIM] where the firstName value matches the pattern specified by the value:

```
<StringFilter
      domainAtribute="firstName"
      comparator="Like" value="Farid%"/>
```

6.5.3.8 CompoundFilter

The CompoundFilter MAY be used to specify a boolean conjunction (AND) or disjunction (OR) between two Filters. It allows a query to express a combination of predicate clauses within a Filter Query.

6.5.3.8.1 Parameters:

***LeftFilter**:* This parameter specifies the first of two Filters for the CompoundFilter.

***RightFilter**:* This parameter specifies the second of two Filters for the CompoundFilter.

***logicalOperator**:* This parameter specifies the logical operator. The value of this parameter MUST be “AND” or “OR”

The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the ClassificationScheme class defined by [ebRIM]:

```
<CompoundFilter logicalOperator="AND">
  <LeftFilter domainAttribute="targetObject" comparator="EQ"
    value="${REGISTRY_OBJECT_ID}" type="StringFilter"/>
  <RightFilter domainAttribute="associationType" comparator="EQ"
    value="${HAS_MEMBER_ASSOC_TYPE_NODE_ID}" type="StringFilter"/>
</CompoundFilter>
```

6.5.4 Nested Query Elements

A Query element MAY contain one or more nested Query sub-elements. The purpose of the nested Query element is to allow traversal of the branches within the network of relationships defined by the information model and prune or filter those branches that do not meet the predicates specified in the corresponding Branch element.

The [RR-QUERY-XSD] XML Schema defines zero or more nested Query elements within a Query element definition using the following pattern:

- A nested Query element is defined for each attribute of the Query domain class that satisfy all of the following criterea:
  - The attribute's domain class is a descendant type of the RegistryObjectType.
  - The attribute's domain class contains reference attributes that link the domain class to some third class via the reference.

  The attribute for which the nested Query is defined is referred to as the Nested Query domain attribute. The domain class of the nested Query domain attribute is the Query domain class for the nested Query element.
- The name of the nested Query element is <Nested Query Domain Attribute Name>Query.
- The type of the nested Query element matches the QueryType for the domain class for the Query domain attribute.
- The cardinality of the nested Query element matches the cardinality of the nested Query domain attribute in the Query domain class.

The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to define nested Query elements for the OrganizationQueryType for the Organization class defined by [ebRIM].

```
  <complexType name="OrganizationQueryType">
    <complexContent>
      <extension base="tns:RegistryObjectQueryType">
        <sequence>
          ...Filters and Branches go here ...
          <element maxOccurs="1" minOccurs="0"
            name="ParentQuery" type="tns:OrganizationQueryType"/>
          <element maxOccurs="unbounded" minOccurs="0"
            name="ChildOrganizationQuery" type="tns:OrganizationQueryType"/>
          <element maxOccurs="1" minOccurs="0"
            name="PrimaryContactQuery" type="tns:PersonQueryType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

6.5.5 Branch Elements

A Query element MAY contain one or more Branch sub-elements. A Branch element is similar to the nested Query element as it too can have sub-elements that are Filter, Branch and subQuery elements.

However, it is different from Query elements because its type is not a descendant type of RegistryObjectQueryType. The purpose of the branch element is to allow traversal of the branches within the network of relationships defined by the information model and prune or filter those branches that do not meet the predicates specified in the corresponding Branch element.

The [RR-QUERY-XSD] XML Schema defines zero or more Branch elements within a Query element definition using the following pattern:

- A Branch element is defined for each attribute of the Query domain class that satisfies all of the following criterea:
  - The attribute's domain is not a primitive type (e.g. String, float, dateTime, int etc.).
  - The attribute's domain class contains reference attributes that link the domain class to some third class via the reference.

  The attribute for which the Branch is defined is referred to as the Branch domain attribute. The domain class of the Branch domain attribute is the Branch domain class for the Branch element.
- The name of the Branch element is <Branch Domain Attribute Name>Branch.
- The cardinality of the Branch element matches the cardinality of the Branch domain attribute in the Query domain class.

The following example shows how the [RR-QUERY-XSD] XML Schema uses the above pattern to define Branches for the RegistryObjectQueryType for the RegistryObject class defined by [ebRIM].

```
<complexType name="RegistryObjectQueryType">
  <complexContent>
    <extension base="tns:FilterQueryType">
      <sequence>
        <element maxOccurs="unbounded" minOccurs="0"
          name="SlotBranch" type="tns:SlotBranchType"/>
        <element maxOccurs="1" minOccurs="0" name="NameBranch"
          type="tns:InternationalStringBranchType"/>
        <element maxOccurs="1" minOccurs="0" name="DescriptionBranch"
          type="tns:InternationalStringBranchType"/>
        ... Relevant Filters, queries go here...
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

6.6 Query Examples

This section provides examples in both SQL and Filter Query syntax for some common query use cases. Each example gives the SQL syntax for the query followed by blank line followed by the equivalent Filter Query syntax for it.

6.6.1 Name and Description Queries

The following queries matches all RegistryObject instances whose name contains the word ‘Acme’ and whose description contains the word “bicycle”.

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

<RegistryObjectQuery>
  <NameBranch>
    <LocalizedStringFilter comparator="Like" domainAttribute="value"
      value="%Acme%" xsi:type="StringFilterType"/>
  </NameBranch>
  <DescriptionBranch>
    <LocalizedStringFilter comparator="Like" domainAttribute="value"
      value="%bicycle%" xsi:type="StringFilterType"/>
  </DescriptionBranch>
</RegistryObjectQuery>
```

6.6.2 Classification Queries

This section describes various classification related queries.

6.6.2.1 Retrieving ClassificationSchemes

The following query retrieves the collection of all ClassificationSchemes. Note that the above query may also specify additional Filters, Querys and Branches as search criterea if desired.

```
SELECT scheme.* FROM ClassificationScheme scheme;

<ClassificationSchemeQuery/>
```

6.6.2.2 Retrieving Children of Specified ClassificationNode

The following query retrieves the children of a ClassificationNode given the “id” attribute of the parent ClassificationNode:

```
SELECT cn.* FROM ClassificationNode cn WHERE parent = ${PARENT_ID};

<ClassificationNodeQuery>
  <PrimaryFilter comparator="Like" domainAttribute="parent"
    value="${PARENT_ID}" xsi:type="StringFilterType"/>
</ClassificationNodeQuery>
```

6.6.2.3 Retrieving Objects Classified By a ClassificationNode

The following query retrieves the collection of ExtrinsicObjects that are classified by the Automotive Industry and the Japan Geography. Note that the query does not match ExtrinsicObjects classified by descendant ClassificationNodes of the Automotive Industry and the Japan Geography. That would require a slightly more complex query.

```
SELECT eo.* FROM ExtrinsicObject eo WHERE
   id IN (SELECT classifiedObject FROM Classification
         WHERE
              classificationNode IN (SELECT id FROM ClassificationNode
              WHERE path = '/${GEOGRAPHY_SCHEME_ID}/Asia/Japan'))
  AND
   id IN (SELECT classifiedObject FROM Classification
         WHERE
              classificationNode IN (SELECT id FROM ClassificationNode
              WHERE path = '/${INDUSTRY_SCHEME_ID}/Automotive'))

<ExtrinsicObjectQuery>
  <ClassificationQuery>
    <ClassificationNodeQuery>
      <PrimaryFilter comparator="EQ" domainAttribute="path"
```

```
        value="/${GEOGRAPHY_SCHEME_ID}/Asia/Japan"
        xsi:type="StringFilterType"/>
    </ClassificationNodeQuery>
  </ClassificationQuery>
  <ClassificationQuery>
    <ClassificationNodeQuery>
      <PrimaryFilter comparator="EQ" domainAttribute="path"
        value="/${INDUSTRY_SCHEME_ID}/Automotive"
        xsi:type="StringFilterType"/>
    </ClassificationNodeQuery>
  </ClassificationQuery>
</ExtrinsicObjectQuery>
```

6.6.2.4 Retrieving Classifications that Classify an Object

The following query retrieves the collection of Classifications that classify a object with id matching ${ID}:

```
SELECT c.* FROM Classification c
       WHERE c.classifiedObject = ${ID};

<ClassificationQuery>
  <PrimaryFilter comparator="EQ" domainAttribute="classifiedObject"
    value="${ID}" xsi:type="StringFilterType"/>
</ClassificationQuery>
```

6.6.3 Association Queries

This section describes various Association related queries.

6.6.3.1 Retrieving All Associations With Specified Object As Source

The following query retrieves the collection of Associations that have the object with id matching ${SOURCE_ID} as their source:

```
SELECT a.* FROM Association a WHERE sourceObject = ${SOURCE_ID}

<AssociationQuery>
  <PrimaryFilter comparator="EQ" domainAttribute="sourceObject"
    value="${SOURCE_ID}" xsi:type="StringFilterType"/>
</AssociationQuery>
```

6.6.3.2 Retrieving All Associations With Specified Object As Target

The following query retrieves the collection of Associations that have the object with id matching ${TARGET_ID} as their target:

```
SELECT a.* FROM Association a WHERE targetObject = ${TARGET_ID}

<AssociationQuery>
  <PrimaryFilter comparator="EQ" domainAttribute="targetObject"
    value="${TARGET_ID}" xsi:type="StringFilterType"/>
</AssociationQuery>
```

6.6.3.3 Retrieving Associated Objects Based On Association Type

Select Associations whose associationType attribute value matches the value specified by the ${ASSOC_TYPE_ID}. The ${ASSOC_TYPE_ID} value MUST reference a ClassificationNode that is a descendant of the canonical AssociationType ClassificationScheme.

```
SELECT a.* FROM Association a WHERE
      associationType = ${ASSOC_TYPE_ID}

<AssociationQuery>
  <PrimaryFilter comparator="EQ" domainAttribute="associationType"
    value="${ASSOC_TYPE_ID}" xsi:type="StringFilterType"/>
</AssociationQuery>
```

6.6.3.4 Complex Association Query

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

```
SELECT a.* FROM Association a WHERE
      sourceObject = ${SOURCE_ID} AND
      targetObject = ${TARGET_ID} AND
      associationType = ${ASSOC_TYPE_ID};

<AssociationQuery>
  <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
    <LeftFilter comparator="EQ" domainAttribute="sourceObject"
      xsi:type="StringFilterType" value="${SOURCE_ID}"/>
    <RightFilter logicalOperator="AND" xsi:type="CompoundFilterType">
      <LeftFilter comparator="EQ" domainAttribute="targetObject"
        xsi:type="StringFilterType" value="${TARGET_ID}"/>
      <RightFilter comparator="EQ" domainAttribute="associationType"
        xsi:type="StringFilterType" value="${ASSOC_TYPE_ID}"/>
    </RightFilter>
  </PrimaryFilter>
</AssociationQuery>
```

6.6.4 Package Queries

The following query retrieves all Packages that have as member the RegistryObject specified by ${REGISTRY_OBJECT_ID}:

```
SELECT p.* FROM Package p, Association a WHERE
      a.sourceObject = p.id AND
      a.targetObject = ${REGISTRY_OBJECT_ID} AND
      a.associationType = ${HAS_MEMBER_ASSOC_TYPE_NODE_ID};

<RegistryPackageQuery>
  <SourceAssociationQuery>
    <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
      <LeftFilter comparator="EQ" domainAttribute="targetObject"
        value="${REGISTRY_OBJECT_ID}"
        xsi:type="StringFilterType"/>
      <RightFilter comparator="EQ" domainAttribute="associationType"
        value="${HAS_MEMBER_ASSOC_TYPE_NODE_ID}"
        xsi:type="StringFilterType"/>
    </PrimaryFilter>
  </SourceAssociationQuery>
</RegistryPackageQuery>
```

Note that the ${HAS_MEMBER_ASSOC_TYPE_NODE_ID} is a placeholder for the value of the id attribute of the canonical HasMember AssociationType ClassificationNode.

6.6.5 ExternalLink Queries

The following query retrieves all ExternalLinks that serve as ExternalLink for the RegistryObject specified by ${REGISTRY_OBJECT_ID}:

```
SELECT el.* From ExternalLink el, Association a WHERE
      a.sourceObject = el.id AND
      a.targetObject = ${REGISTRY_OBJECT_ID} AND
      a.associationType = ${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};

<ExternalLinkQuery>
  <SourceAssociationQuery>
    <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
      <LeftFilter comparator="EQ" domainAttribute="targetObject"
        value="${REGISTRY_OBJECT_ID}"
        xsi:type="StringFilterType"/>
      <RightFilter comparator="EQ" domainAttribute="associationType"
        value="${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
        xsi:type="StringFilterType"/>
    </PrimaryFilter>
  </SourceAssociationQuery>
</ExternalLinkQuery>
```

Note that the ${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID} is a placeholder for the value of the id attribute of the canonical ExternallyLinks AssociationType ClassificationNode.

The following query retrieves all ExtrinsicObjects that are linked to an ExternalLink specified by ${EXTERNAL_LINK_ID}:

```
SELECT eo.* From ExtrinsicObject eo, Association a WHERE
      a.sourceObject = ${EXTERNAL_LINK_ID} AND
      a.targetObject = eo.id AND
      a.associationType = ${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};

<ExtrinsicObjectQuery>
  <TargetAssociationQuery>
    <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
      <LeftFilter comparator="EQ" domainAttribute="sourceObject"
        value="${EXTERNAL_LINK_ID}"
        xsi:type="StringFilterType"/>
      <RightFilter comparator="EQ" domainAttribute="associationType"
        value="${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
        xsi:type="StringFilterType"/>
    </PrimaryFilter>
  </TargetAssociationQuery>
</ExtrinsicObjectQuery>
```

6.6.6 Audit Trail Queries

The following query retrieves all the AuditableEvents for the RegistryObject specified by ${REGISTRY_OBJECT_ID}:

```
SELECT ae.* FROM AuditableEvent ae, AffectedObject ao WHERE
      ao.eventId = ae.id AND
      ao.id = ${REGISTRY_OBJECT_ID}

<AuditableEventQuery>
  <AffectedObjectQuery>
    <PrimaryFilter comparator="EQ" domainAttribute="id"
      value="${REGISTRY_OBJECT_ID}" xsi:type="StringFilterType"/>
  </AffectedObjectQuery>
</AuditableEventQuery>
```

7 Event Notification Protocols

This chapter defines the Event Notification feature of the OASIS ebXML Registry.

Event Notification feature allows OASIS ebXML Registries to notify its users and / or other registries about events of interest. It allows users to stay informed about registry events without being forced to periodically poll the registry. It also allows a registry to propagate internal changes to other registries whose content might be affected by those changes.

ebXML registries support content-based Notification where interested parties express their interest in form of a query. This is different from subject–based (sometimes referred to as topic-based) notification, where information is categorized by subjects and interested parties express their interests in those predefined subjects.

7.1 Use Cases

The following use cases illustrate different ways in which ebXML registries notify users or other registries.

7.1.1 CPP Has Changed

A user wishes to know when the CPP [ebCPP] of a partner is updated or superseded by another CPP. When that happens he may wish to create a CPA [ebCPP] based upon the new CPP.

7.1.2 New Service is Offered

A user wishes to know when a new plumbing service is offered in her town and be notified every 10 days. When that happens, she might try to learn more about that service and compare it with her current plumbing service provider’s offering.

7.1.3 Monitor Download of Content

User wishes to know whenever his CPP [ebCPP] is downloaded in order to evaluate on an ongoing basis the success of his recent advertising campaign. He might also want to analyze who the interested parties are.

7.1.4 Monitor Price Changes

User wishes to know when the price of a product that she is interested in buying drops below a certain amount. If she buys it she would also like to be notified when the product has been shipped to her.

7.1.5 Keep Replicas Consistent With Source Object

In order to improve performance and availability of accessing some registry objects, a local registry MAY make replicas of certain objects that are hosted by another registry. The registry would like to be notified when the source object for a replica is updated so that it can synchronize the replica with the latest state of the source object.

7.2 Registry Events

Activities within a registry result in meaningful events. Typically, registry events are generated when a registry processes client requests. In addition, certain registry events may be caused by administrative actions performed by a registry operator. [ebRIM] defines the AuditableEvent class, instances of which represent registry events. When such an event occurs, an AuditableEvent instance is generated by the registry.

7.3 Subscribing to Events

A user MAY create a subscription with a registry if he or she wishes to receive notification for a specific type of event. A user creates a subscription by submitting a Subscription instance to a registry using the SubmitObjectsRequest. If a Subscription is submitted to a registry that does not support event notification then the registry MUST return an UnsupportedCapabilityException.

The listing below shows a sample Subscription using a pre-defined SQL query as its selector that will result in an email notification to the user whenever a Service is created that is classified as a “Plumbing” service and located in “A Little Town.”

The SQL query within the selector in plain English says the following:

Find all Services that are Created AND classified by ClassificationNode where ClassificationNode's Path ends with string "Plumbing", AND classified by ClassificationNode where ClassificationNode's Code contains string "A Little Town.”

```
<rim:Subscription id="${SUBSCRIPTION_ID}" selector="${QUERY_ID}">
  <!--
       The selector is a reference to a query object that has the
following query defined
       SELECT * FROM Service s, AuditableEvent e, AffectectedObject ao,
       Classification c1, Classification c2
       ClassificationNode cn1, ClassificationNode cn2 WHERE
       e.eventType = 'Created' AND ao.id = s.id AND ao.parent=e.id AND
       c1.classifiedObject = s.id AND c1.classificationNode = cn1.id
AND
       cn1.path LIKE '%Plumbing' AND
       c2.classifiedObject = s.id AND c2.classificationNode = cn2.id
AND
       cn2.path LIKE '%A Little Town%'
  -->
  <!-- Next endPoint is an email address -->
  <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-
regrep:NotificationOptionType:Objects"
endPoint="mailto:farrukh.najmi@sun.com"/>
  <!-- Next endPoint is a service via reference to its ServiceBinding
object -->
  <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-
regrep:NotificationOptionType:ObjectRefs"
endPoint="urn:freebxml:registry:demoDB:serviceBinding:EpidemicAlertList
enerServiceBinding"/>
</rim:Subscription>
```

7.3.1 Event Selection

In order to only be notified of specific events of interest, the user MUST specify a reference to a stored AdHocQuery object via the selector attribute within the Subscription instance. The query determines whether an event qualifies for that Subscription or not. For details on query syntax see chapter 6.

7.3.2 Notification Action

When creating a Subscription, a user MAY also specify Actions within the subscription that specify what the registry must do when an event matching the Subscription (subscription event) transpires.

A user MAY omit specifying an Action within a Subscription if he does not wish to be notified by the registry. A user MAY periodically poll the registry and pull the pending Notifications.

[ebRIM] defines two standard ways that a NotifyAction may be used:

- Email NotifyAction that allows delivery of event notifications via email to a human user or to an email end point for a software component or agent.
- Service NotifyAction that allows delivery of event notifications via a programmatic interface by invoking a specified listener web service.

If the registry supports event notification, at some time after the successful processing of each request, it MUST check all registered and active Subscriptions and see if any Subscriptions match the event. If a match is found then the registry performs the Notification Actions required for the Subscription. A registry MAY periodically perform such checks and corresponding notification actions in a batch mode based upon registry specific policies.

7.3.3 Subscription Authorization

A registry operator or content owner MAY use custom Access Control Policies to decide which users are authorized to create a subscription and to what events. A Registry MUST return an AuthorizationException in the event that an unauthorized user submits a Subscription to a registry. It is up to registry implementations whether to honour the existing subscription if an access control policy governing subscriptions becomes more restrictive after subscription have already been created based on the older policy.

7.3.4 Subscription Quotas

A registry MAY use registry specific policies to decide an upper limit on the number of Subscriptions a user is allowed to create. A Registry MUST return a QuotaExceededException in the event that an authorized user submits more Subscriptions than allowed by their registry specific quota.

7.3.5 Subscription Expiration

Each subscription defines a startTime and and endTime attribute which determines the period within which a Subscription is active. Outside the bounds of the active period, a Subsription MAY exist in an expired state within the registry. A registry MAY remove an expired Subscription at any time. In such cases the identity of a RegistryOperator user MUST be used for the request in order to have sufficient authorization to remove a user’s Subscription.

A Registry MUST NOT consider expired Subscriptions when delivering notifications for an event to its Subscriptions. An expired Subscription MAY be renewed by submitting a new Subscription.

7.3.6 Subscription Rejection

A Registry MAY reject a Subscription if it is too costly to support. For instance a Subscription that wishes to be notified of any change in any object may be too costly for most registries. A Registry MUST return a SubscriptionTooCostlyException in the event that an Authorized User submits a Subscription that is too costly for the registry to process.

7.4 Unsubscribing from Events

A user MAY terminate a Subscription with a registry if he or she no longer wishes to be notified of events related to that Subscription. A user terminates a Subscription by deleting the corresponding Subscription object using the RemoveObjectsRequest to the registry.

Removal of a Subscription object follows the same rules as removal of any other object.

7.5 Notification of Events

A registry performs the *Actions* for a Subscription in order to actually deliver the events information to the subscriber. However, regardless of the specific delivery Action, the registry MUST communicate the Subscription events. The Subscription events are delivered within a Notification instance as described by [ebRIM]. In case of Service NotifyAction, the Notification is delivered to a handler service conformant to the RegistryClient interface. In case of an Email NotifyAction the notification is delivered an email address.

The listing below shows a sample Notification matching the subscription example in section 7.3:

```
<rim:Notification subscription="${SUBSCRIPTION_ID}">
  <rim:RegistryObjectList>
    <rim:Service id="f3373a7b-4958-4e55-8820-d03a191fb76a">
      <rim:Name>
        <rim:LocalizedString value="A Little Town Plumbing"/>
      </rim:Name>
      <rim:Classification id="a3373a7b-4958-4e55-8820-d03a191fb76a"
classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
      <rim:Classification id="b3373a7b-4958-4e55-8820-d03a191fb76a"
classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
    </rim:Service>
  </rim:RegistryObjectList>
</rim:Notification>
```

A Notification MAY contain actual RegistryObjects or ObjectRefs to RegistryObjects within the <rim:RegistryObjectList>. A client MAY specify the whether they wish to receive RegistryObjects or ObjectRefs to RegistryObjects using the notificationOption attribute of the Action within the Subscription. The registry MAY override this notificationOption based upon registry specific operational policies.

7.6 Retrieval of Events

The registry provides asynchronous PUSH style delivery of Notifications via notify Actions as described earlier. However, a client MAY also use a PULL style to retrieve any pending events for their Subscriptions. Pulling of events is done using the AdHocQuery protocol and querying the Notification class. A registry SHOULD buffer undelivered notifications for some period to allow clients to PULL those notifications. The period that a registry SHOULD buffer undelivered notifications MAY be defined using registry specific policies.

7.7 Pruning of Events

A registry MAY periodically prune AuditableEvents in order to manage its resources. It is up to the registry when such pruning occurs. It is up to the registry to determine when undelivered events are purged. A registry SHOULD perform such pruning by removing the older information in its Audit Trail content. However, it MUST not remove the original Create Event at the beginning of the audit trail since the Create Event establishes the owner of the RegistryObject.

8 Content Management Services

This chapter describes the Content Management services of the ebXML Registry. Examples of Content Management Services include, but are not limited to, content validation and content cataloging. Content Management Services result in improved quality and integrity of registry content and metadata as well as improved ability for clients to discover that content and metadata.

The Content Management Services facility of the registry is based upon a pluggable architecture that allows clients to publish and discover new Content Management Services as Service objects that conform to a normative web service interface specified in this chapter. Clients MAY configure a Content Management Service that is specialized for managing a specific type of content.

8.1 Content Validation

The Content Validation feature provides the ability to enforce domain specific validation rules upon submitted content and metadata in a content specific manner.

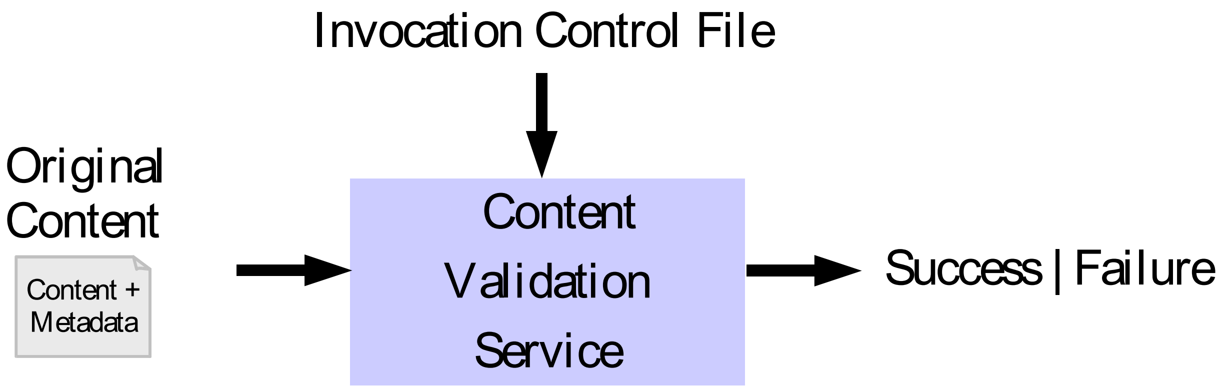


Figure 13: Content Validation Service

A registry uses one or more Content Validation Services to automatically validate the RegistryObjects and repository items when they are submitted to the registry. A registry MUST reject a submission request in its entirety if it contains invalid data. In such cases a ValidationException MUST be returned to the client.

Content Validation feature improves the quality of data in the registry.

8.1.1 Content Validation: Use Cases

The following use cases illustrate the Content Validation feature:

8.1.1.1 Validation of HL7 Conformance Profiles

The Healthcare Standards organization HL7 uses content validation to enforce consistency rules and semantic checks whenever an HL7 member submits an HL7 Conformance Profile. HL7 is also planning to use the feature to improve the quality of other types of HL7 artifacts.

8.1.1.2 Validation of Business Processes

Content validation may be used to enforce consistency rules and semantic checks whenever a Business Process is submitted to the registry. This feature may be used by organizations such as UN/CEFACT, OAGi, and RosettaNet.

8.1.1.3 Validation of UBL Business Documents

Content validation may be used by the UBL technical committee to enforce consistency rules and semantic checks whenever a UBL business document is submitted to the registry.

8.2 Content Cataloging

The Content Cataloging feature provides the ability to selectively convert submitted RegistryObject and repository items into metadata defined by [ebRIM], in a content specific manner.

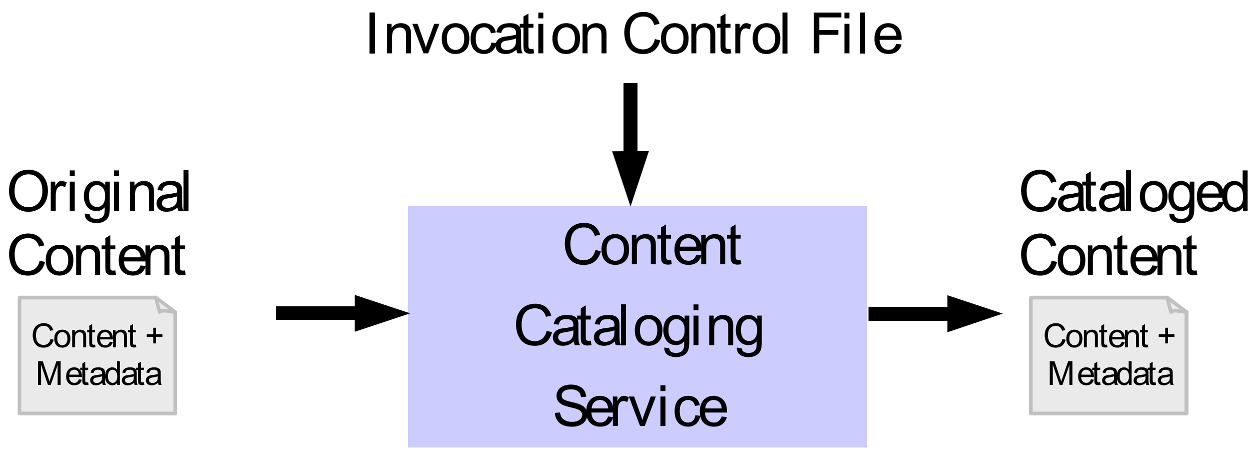


Figure 14: Content Cataloging Service

A registry uses one or more Content Cataloging Services to automatically catalog RegistryObjects and repository items. Cataloging creates and/or updates RegistryObject metadata such as ExtrinsicObject or Classification instances. The cataloged metadata enables clients to discover the repository item based upon content from the repository item, using standard query capabilities of the registry. This is referred to as *Content-based Discovery*.

The main benefit of the Content Cataloging feature is to enable Content-based Discovery.

8.2.1 Content-based Discovery: Use Cases

There are many scenarios where content-based discovery is necessary.

8.2.1.1 Find All CPPs Where Role is “Buyer”

A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to find CPPs for other companies where the Role element of the CPP is that of “Buyer”.

8.2.1.2 Find All XML Schema’s That Use Specified Namespace

A client may wish to discover all XML Schema documents in the registry that use an XML namespace containing the word “oasis”.

8.2.1.3 Find All WSDL Descriptions with a SOAP Binding

An ebXML registry client is attempting to discover all repository items that are WSDL descriptions that have a SOAP binding defined. Note that SOAP binding related information is content within the WSDL document and not metadata.

8.3 Abstract Content Management Service

This section describes in abstract terms how the registry supports pluggable, user-defined Content Management Services. A Content Management Service is invoked in response to content being submitted to the registry via the standard Submit/UpdateObjectsRequest method. The Service invocation is on a per request basis where one request may result in many invocations, one for each RegistryObject for which a Content Management Service is configured within the registry.

The registry may perform such invocation in one of two ways.

- ***Inline Invocation Model***: Content Management Service may be invoked inline with the processing of the Submit/UpdateObjectsRequest and prior to committing the content. This is referred to as Inline Invocation Model.
- ***Decoupled Invocation Model***: Content Management Service may be invoked decoupled from the processing of the Submit/UpdateObjectsRequest and some time after committing the content. This is referred to as Decoupled Invocation Model.

8.3.1 Inline Invocation Model

In an inline invocation model a registry MUST invoke a Content Management Service inline with Submit/UpdateObjectsRequest processing and prior to committing the Submit/UpdateObjectsRequest. All metadata and content from the original Submit/UpdateObjectsRequest request or from the Content Management Service invocation MUST be committed as an atomic transaction.

Figure 15 shows an abstract Content Management Service and how it is used by an ebXML Registry using an inline invocation model. The steps are as follows:

1. A client submits a Content Management Service S1 to an ebXML Registry. The client typically belongs to an organization responsible for defining a specific type of content. For example the client may belong to RosettaNet.org and submit a Content Validation Service for validating RosettaNet PIPs. The client uses the standard Submit/UpdateObjectsRequest interface to submit the Service. This is a one-time step to configure this Content Management Service in the registry.
2. Once the Content Management Service has been submitted, a potentially different client may submit content to the registry that is of the same object type for which the Content Management Service has been submitted. The client uses the standard Submit/UpdateObjectsRequest interface to submit the content.
3. The registry determines there is a Content Management Service S1 configured for the object type for the content submitted. It invokes S1 using a ContentManagementServiceRequest and passes it the content.
4. The Content Management Service S1 processes the content and sends back a ContentManagementServiceResponse.
5. The registry then commits the content to the registry if there are no errors encountered.
6. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest in step 2.

Figure 15: Content Management Service: Inline Invocation Model

8.3.2 Decoupled Invocation Model

In a decoupled invocation model a registry MUST invoke a Content Management Service independent of or decoupled from the Submit/UpdateObjectsRequest processing. Any errors encountered during Content Management Service invocation MUST NOT have any impact on the original Submit/UpdateObjectsRequest processing.

All metadata and content from the original Submit/UpdateObjectsRequest request MUST be committed as an atomic transaction that is decoupled from the metadata and content that may be generated by the Content Management Service invocation.

Figure 16 shows an abstract Content Management Service and how it is used by an ebXML Registry using a decoupled invocation model. The steps are as follows:

1. Same as in inline invocation model (Content Management Service is submitted).
2. Same as in inline invocation model (client submits content using Submit/UpdateObjectsRequest).
3. The registry processes the Submit/UpdateObjectsRequest and commits it to persistent store.
4. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest in step 2.
5. The registry determines there is a Content Management Service S1 configured for the object type for the content submitted. It invokes S1 using a ContentManagementServiceRequest and passes it the content.
6. The Content Management Service S1 processes the content and sends back a ContentManagementServiceResponse.

7. If the ContentManagementServiceResponse includes any generated or modified content it is committed to the persistent store as separate transaction. If there are any errors encountered during decoupled invocation of a Content Management Service then these errors are logged by the registry in a registry specific manner and MUST NOT be reported back to the client.

Figure 16: Content Management Service: Decoupled Invocation Model

8.4 Content Management Service Protocol

This section describe the abstract Content Management Service protocol that is the base- protocol for other concrete protocols such as Validate Content protocol and Catalog Content protocol. The concrete protocols will be defined later in this document.

8.4.1 ContentManagementServiceRequestType

The ContentManagementServiceRequestType MUST be the abstract base type for all requests sent from a registry to a Content Management Service.

8.4.1.1 Syntax:

```
<complexType name="ContentManagementServiceRequestType">
   <complexContent>
     <extension base="rs:RegistryRequestType">
       <sequence>
         <element name="OriginalContent"
type="rim:RegistryObjectListType"/>
         <element name="InvocationControlFile"
type="rim:ExtrinsicObjectType" maxOccurs="unbounded" minOccurs="0"/>
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

8.4.1.2 Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

InvocationControlFile: This parameter specifies the ExtrinsicObject for a repository item that the caller wishes to specify as the Invocation Control File. This specification does not specify the format of this file. There MUST be a corresponding repository item as an attachment to this request. The corresponding repository item SHOULD follow the same rules as attachments in Submit/UpdateObjectsRequest.

OriginalContent: This parameter specifies the RegistryObjects that will be processed by the content management service. In case of ExtrinsicObject instances within the OriginalContent there MAY be repository items present as attachments to the ContentManagementServiceRequest. This specification does not specify the format of such repository items. The repository items SHOULD follow the same rules as attachments in Submit/UpdateObjectsRequest.

8.4.1.3 Returns:

This request returns a ContentManagementServiceResponse. See section 8.4.2 for details.

8.4.1.4 Exceptions:

In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

MissingRepositoryItemException: signifies that the caller did not provide a repository item as an attachment to this request when the Service requires it.

InvocationControlFileException: signifies that the InvocationControlFile(s) provided by the caller do not match the InvocationControlFile(s) expected by the Service.

UnsupportedContentException: signifies that this Service does not support the content provided by the caller.

8.4.2 ContentManagementServiceResponseType

The ContentManagementServiceResponseType is sent by a Content Management Service as a response to a ContentManagementServiceRequestType. The ContentManagementServiceResponseType is the abstract base type for all responses sent to a registry from a Content Management Service. It extends the RegistryResponseType and does not define any new parameters.

8.4.2.1 Syntax:

```
<complexType name="ContentManagementServiceResponseType">
   <complexContent>
     <extension base="rs:RegistryResponseType">
       <sequence>
       </sequence>
     </extension>
   </complexContent>
 </complexType>
```

8.4.2.2 Parameters:

No new parameters are defined other than those inherited from RegistryResponseType.

8.5 Publishing / Configuration of a Content Management Service

Any Submitter MAY submit an arbitrary Content Management Service to an ebXML Registry. The Content Management Service MUST be published using the standard LifeCycleManager interface.

The Submitter MUST use the standard Submit/UpdateObjectsRequest to publish:

- A Service instance for the Content Management Service. In Figure 17 this is exemplified by the defaultXMLCatalogingService in the upper-left corner. The Service instance MUST have an Association with a ClassificationNode in the canonical ObjectType ClassificationScheme as defined by [ebRIM]. The Service MUST be the sourceObject while a ClassificationNode MUST be the targetObject. This association binds the Service to that specific ObjectType. The associationType for this Association instance MUST be “ContentManagementServiceFor.” The Service MUST be classified by the canonical ContentManagementService ClassificationScheme as defined by [ebRIM]. For example it may be classified as a “ContentValidationService” or a “ContentCatalogingService.”
- The Service instance MAY be classified by a ClassificationNode under the canonical InvocationModel ClassificationScheme as defined by [ebRIM], to determine whether it uses the Inline Invocation model or the Decoupled Invocation model.
- The Service instance MAY be classified by a ClassificationNode under the canonical ErrorHandlingModel ClassificationScheme as defined by [ebRIM], to determine whether the Service should fail on first error or simply log the error as a warning and continue. See section 8.6.4 for details.
- A ServiceBinding instance contained within the Service instance that MUST provide the accessURI to the Cataloging Service.
- An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page describing:

    The format of the supported content to be Cataloged

    The format of the supported Invocation Control File

  Note that no SpecificationLink is required since this specification [ebRS] is implicit for Content Cataloging Services.
- One or more Invocation Control File(s) consisting of an ExtrinsicObject and a repository item pair. The ExtrinsicObject for the Invocation Control File MUST have a required Association with associationType value that references a descendant ClassificationNode of the canonical ClassificationNode “InvocationControlFileFor.” This is exemplified by the cppCatalogingServiceXSLT and the oagBODCatalogingServiceXSLT objects in Figure 17 (left side of picture). The Invocation Control File MUST be the sourceObject while a ClassificationNode in the canonical ObjectType ClassificationScheme MUST be the targetObject.

    ○

Figure 17: Cataloging Service Configuration

Figure 17 shows an example of the configuration of the Canonical XML Cataloging Service associated with the objectType for XML content. This Cataloging Service may be used with any XML content that has its objectType attribute hold a reference to the xmlObjectType ClassificationNode or one of its descendants.

The figure also shows two different Invocation Control Files, cppCatalogingServiceXSLT and oagBODCatalogingServiceXSLT that may be used to catalog ebXML CPP and OAG Business Object Documents (BOD) respectively.

8.5.1 Multiple Content Management Services and Invocation Control Files

This specification allows clients to submit multiple Content Management Services of the same type (e.g. validation, cataloging) and multiple Invocation Control Files for the same objectType. Content Management Services of the same type of service for the same ObjectType are referred to as peer Content Management Services.

When there are multiple Content Management Services and Invocation Control Files for the same ObjectType there MUST be an unambiguous association between a Content Management Service and its Invocation Control File(s). This MUST be defined by an Association instance with associationType value that references a ClassificationNode that is a descendant of the canonical ClassificationNode “InvocationControlFileFor” where the ExtrinsicObject for each Invocation Control File is the sourceObject and the Service is the targetObject.

The order of invocation of peer Content Management Services is undefined and MAY be determined in a registry specific manner.

8.6 Invocation of a Content Management Service

This section describes how a registry invokes a Content Management Service.

8.6.1 Resolution Algorithm For Service and Invocation Control File

When a registry receives a submission of a RegistryObject, it MUST use the following algorithm to determine or resolve the Content Management Services and Invocation Control Files to be used for dynamic content management for the RegistryObject:

1. Get the objectType attribute of the RegistryObject.
2. Query to see if the ClassificationNode referenced by the objectType is the targetObject of an Association with associationType of *ContentManagementServiceFor*. If the desired Association is not found for this ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired Association is found or until the parent is the ClassificationScheme. If desired Association(s) is found then repeat following steps for each such Association instance.
3. Check if the sourceObject of the desired Association is a Service instance. If not, log an InvalidConfigurationException. If it is a Service instance, then use this Service as the Content Management service for the RegistryObject.
4. Query to see if the objectType ClassificationNode is the targetObject of one or more Associations whose associationType value references a ClassificationNode that is a descendant of the canonical ClassificationNode *InvocationControlFileFor*. If desired Association is not found for this ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired Association is found or until the parent is the ClassificationScheme.
5. If desired Association(s) is found then check if the sourceObject of the desired Association is an ExtrinsicObject instance. If not, log an InvalidConfigurationException. If sourceObject is an ExtrinsicObject instance, then use its repository item as an Invocation Control File. If there are multiple InvocationControlFiles then all of them MUST be provided when invoking the Service.

The above algorithm allows for objectType hierarchy to be used to configure Content Management Services and Invocation Control Files with varying degrees of specificity or specialization with respect to the type of content.

8.6.2 Audit Trail and Cataloged Content

The Cataloged Content generated as a result of the invocation of a Content Management Service has an audit trail consistent with RegistryObject instances that are submitted by Registry Clients. However, since a Registry Client does not submit Cataloged Content, the user attribute of the AuditableEvent instances for such Cataloged Content references the Service object for the Content Management Service that generated the Cataloged Content. This allows an efficient way to distinguish Cataloged Content from content submitted by Registry Clients.

8.6.3 Referential Integrity

A registry MUST maintain referential integrity between the RegistryObjects and repository items invocation of a Content Management Service.

8.6.4 Error Handling

If the Content Management Service is classified by the “FailOnError” ClassificationNode under canonical ErrorHandlingModel ClassificationScheme as defined by [ebRIM], then the registry MUST stop further processing of the Submit/UpdateObjectsRequest and return status of “Failure” upon first error returned by a Content Management Service Invocation.

If the Content Management Service is classified by the “LogErrorAndContinue” ClassificationNode under ErrorHandlingModel then the registry MUST continue to process the Submit/UpdateObjectsRequest and not let any Content Management Service invocation error affect the storing of the RegistryObjects and repository items that were submitted. Such errors SHOULD be logged as Warnings within the RegistryResponse returned to the client. In this case a registry MUST return a normal response with status of “Success” if the submitted content and metadata is stored successfully even when there are errors encountered during dynamic invocation of one or more Content Management Services.

8.7 Validate Content Protocol

The interface of a Content Validation Service MUST implement a single method called validateContent. The validateContent method accepts a ValidateContentRequest as parameter and returns a ValidateContentResponse as its response if there are no errors.

The OriginalContent element within a ValidateContentRequest MUST contain exactly one RegistryObject that needs to be cataloged. The resulting ValidateContentResponse contains the status attribute that communicates whether the RegistryObject (and its content) are valid or not.

The Validate Content protocol does not specify the implementation details of any specific Content Validation Service.

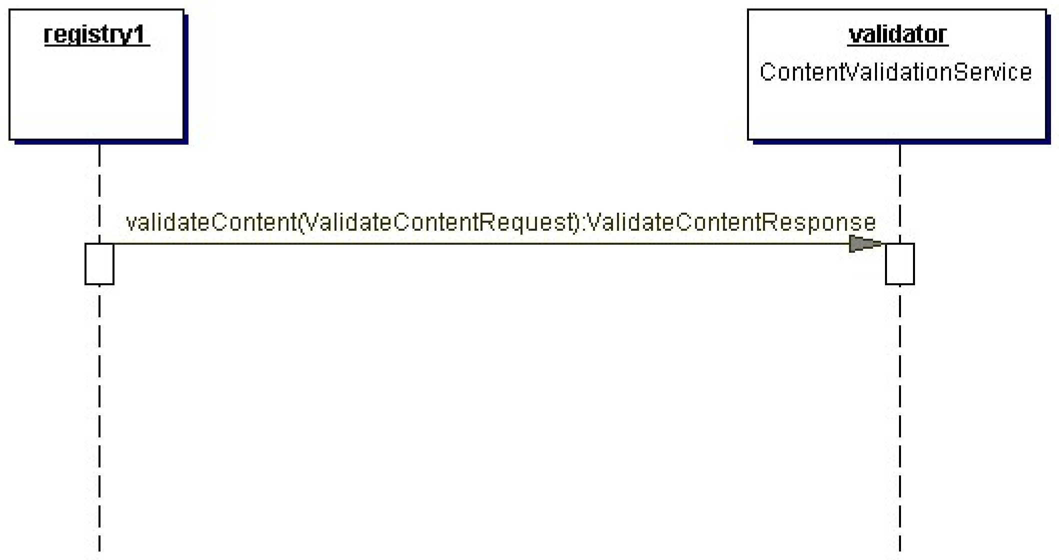

registry1
validator
ContentValidationService
validateContent(ValidateContentRequest):ValidateContentResponse

Figure 18: Validate Content Protocol

8.7.1 ValidateContentRequest

The ValidateContentRequest is used to pass content to a Content Validation Service so that it can validate the specified RegistryObject and any associated content. The RegistryObject typically is an ExternalLink (in the case of external content) or an ExtrinsicObject. The ValidateContentRequest extends the base type ContentManagementServiceRequestType.

8.7.1.1 Syntax:

```
<element name="ValidateContentRequest">
   <complexType>
     <complexContent>
       <extension base="cms:ContentManagementServiceRequestType">
         <sequence>
         </sequence>
       </extension>
```

```
    </complexContent>
  </complexType>
</element>
```

8.7.1.2 Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

InvocationControlFile: Inherited from base type. This parameter may not be present. If present its format is defined by the Content Validation Service.

OriginalContent: Inherited from base type. This parameter MUST contain exactly one RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content. This specification does not specify the format of the content. If it is an ExtrinsicObject then there MAY be a corresponding repository item as an attachment to this request that is the content. The corresponding repository item SHOULD follow the same rules as attachments in Submit/UpdateObjectsRequest.

8.7.1.3 Returns:

This request returns a ValidateContentResponse. See section 8.7.2 for details.

8.7.1.4 Exceptions:

In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

InvalidContentException: signifies that the specified content was found to be invalid. The exception SHOULD include enough detail for the client to be able to determine how to make the content valid.

8.7.2 ValidateContentResponse

The ValidateContentResponse is sent by the Content Validation Service as a response to a ValidateContentRequest.

8.7.2.1 Syntax:

```
<element name="ValidateContentResponse">
   <complexType>
     <complexContent>
       <extension base="cms:ContentManagementServiceResponseType">
         <sequence>
         </sequence>
       </extension>
     </complexContent>
   </complexType>
 </element>
```

8.7.2.2 Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

status: Inherited attribute. This enumerated value is used to indicate the status of the request. Values for status are as follows:

- Success - This status specifies that the content specified in the ValidateContentRequest was valid.
- Failure - This status specifies that the request failed. If the error returned is an InvalidContentException then the content specified in the ValidateContentRequest was invalid. If there was some other failure encountered during the processing of the request then a different error MAY be returned.

8.8 Catalog Content Protocol

The interface of the Content Cataloging Service MUST implement a single method called catalogContent. The catalogContent method accepts a CatalogContentRequest as parameter and returns a CatalogContentResponse as its response if there are no errors.

The CatalogContentRequest MAY contain repository items that need to be cataloged. The resulting CatalogContentResponse contains the metadata and possibly content that gets generated or updated by the Content Cataloging Service as a result of cataloging the specified repository items.

The Catalog Content protocol does not specify the implementation details of any specific Content Cataloging Service.

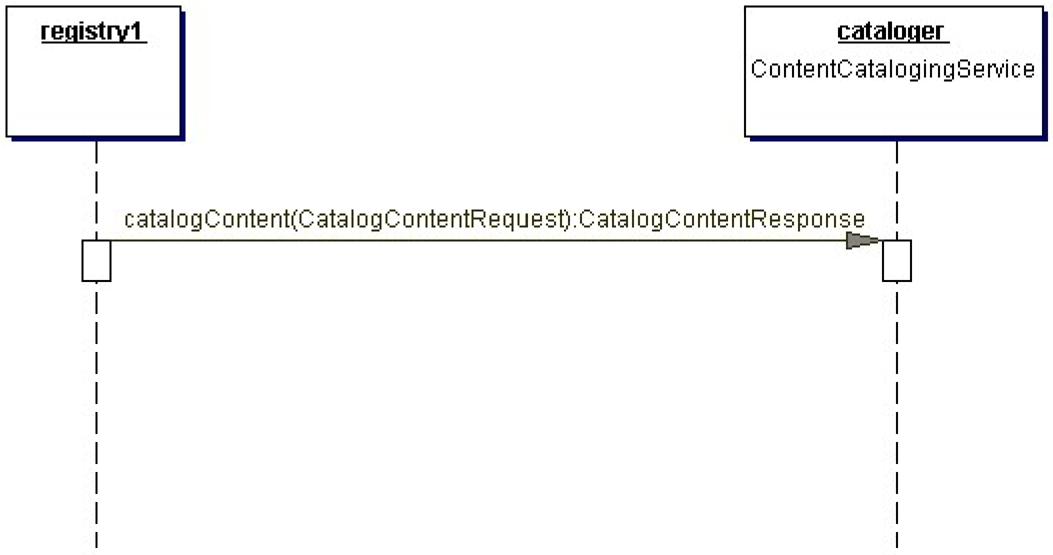

registry1
cataloger
ContentCatalogingService
catalogContent(CatalogContentRequest):CatalogContentResponse

Figure 19: Catalog Content Protocol

8.8.1 CatalogContentRequest

The CatalogContentRequest is used to pass content to a Content Cataloging Service so that it can create catalog metadata for the specified RegistryObject and any associated content. The RegistryObject typically is an ExternalLink (in case of external content) or an ExtrinsicObject. The CatalogContentRequest extends the base type ContentManagementServiceRequestType.

8.8.1.1 Syntax:

```
<element name="CatalogContentRequest">
```

```
   <complexType>
     <complexContent>
       <extension base="cms:ContentManagementServiceRequestType">
         <sequence>
         </sequence>
       </extension>
     </complexContent>
   </complexType>
 </element>
```

8.8.1.2 Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

InvocationControlFile: Inherited from base type. If present its format is defined by the Content Cataloging Service.

OriginalContent: Inherited from base type. This parameter MUST contain exactly one RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content. This specification does not specify the format of the content. If it is an ExtrinsicObject then there MAY be a corresponding repository item as an attachment to this request that is the content. The corresponding repository item SHOULD follow the same rules as attachments in Submit/UpdateObjectsRequest.

8.8.1.3 Returns:

This request returns a CatalogContentResponse. See section 8.8.2 for details.

8.8.1.4 Exceptions:

In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

CatalogingException: signifies that an exception was encountered in the Cataloging algorithm for the service.

8.8.2 CatalogContentResponse

The CatalogContentResponse is sent by the Content Cataloging Service as a response to a CatalogContentRequest.

8.8.2.1 Syntax:

```
<element name="CatalogContentResponse">
   <complexType>
     <complexContent>
       <extension base="cms:ContentManagementServiceResponseType">
         <sequence>
           <element name="CatalogedContent"
type="rim:RegistryObjectListType"/>
         </sequence>
       </extension>
     </complexContent>
```

```
    </complexType>
  </element>
```

8.8.2.2 Parameters:

The following parameters are parameters that are either newly defined for this type or are inherited and have additional semantics beyond those defined in the base type description.

CatalogedContent: This parameter specifies a collection of RegistryObject instances that were created or updated as a result of dynamic content cataloging by a content cataloging service. The Content Cataloging Service may add metadata such as Classifications, ExternalIdentifiers, name, description etc. to the CatalogedContent element. There MAY be an accompanying repository item as an attachment to this response message if the original repository item was modified by the request.

8.9 Illustrative Example: Canonical XML Cataloging Service

Figure 20 shows a UML instance diagram to illustrate how a Content Cataloging Service is used. This Content Cataloging Service is the normative Canonical XML Cataloging Service described in section 8.10.

- In the center we see a Content Cataloging Service name defaultXMLCataloger Service.
- On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP being input as Original Content to the defaultXMLCataloging Service.
- On top we see an XSLT style sheet repository item and its ExtrinsicObject that is configured as an Invocation Control File for the defaultXMLCataloger Service.
- On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for the CPP. We also see a Classification roleClassification, which classifies the CPP by the Role element within the CPP. These are the Cataloged Content generated as a result of the Cataloging Service cataloging the CPP.

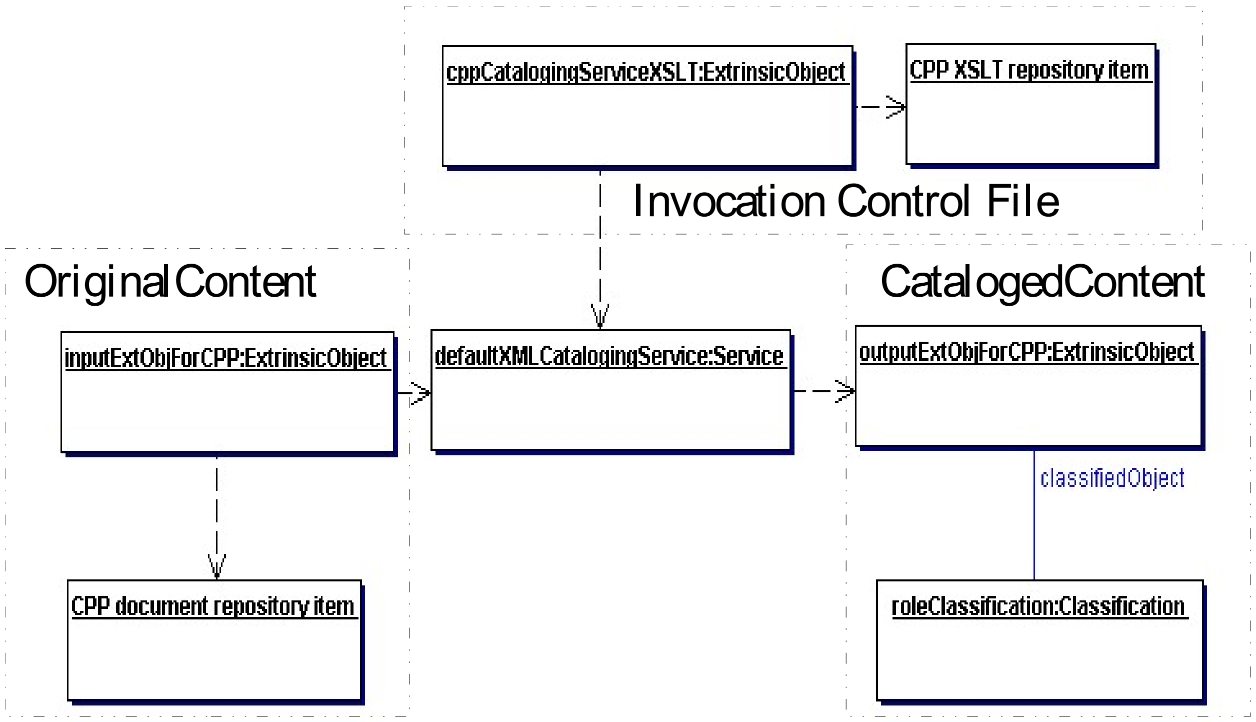

cppCatalogingServiceXSLT:ExtrinsicObject
CPP XSLT repository item
Invocation Control File
Original Content
Cataloged Content
inputExtObjForCPP:ExtrinsicObject
defaultXMLCatalogingService:Service
outputExtObjForCPP:ExtrinsicObject
classifiedObject
CPP document repository item
roleClassification:Classification

Figure 20: Example of CPP cataloging using Canonical XML Cataloging Service

8.10 Canonical XML Content Cataloging Service

An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in service with the following constraints:

- There is exactly one Service instance for the Canonical XML Content Cataloging Service
- The Service is an XSLT engine
- The Service may be invoked with exactly one Invocation Control File
- The Original Content for the Service MUST be XML document(s)
- The Cataloged Content for the Service MUST be XML document(s)
- The Invocation Control File MUST be an XSLT style sheet
- Each invocation of the Service MAY be with different Invocation Control File (XSLT style sheet) depending upon the objectType of the RegistryObject being cataloged. Each objectType SHOULD have its own unique XSLT style sheet. For example, ebXML CPP documents SHOULD have a specialized ebXML CPP Invocation Control XSLT style sheet.
- The Service MUST have at least one input XML document that is a RegistryObject. Typically this is an ExtrinsicObject or an ExternalLink.
- The Service MAY have at most one additional input XML document that is the content represented by the RegistryObject (e.g. a CPP document or an HL7 Conformance Profile). The optional second input MUST be referenced within the XSLT Style sheet by a using the “document” function with the document name specified by variable “repositoryItem” as in “document($repositoryItem).” A registry MUST define the variable “repositoryItem” when invoking the Canonical XML Cataloging Service.
- The canonical XML Content Cataloging Service MUST apply the XSLT style sheet to the input XML instance document(s) in an XSLT transformation to generate the Cataloged Output.

The Canonical XML Content Cataloging Service is a required normative feature of an ebXML Registry.

8.10.1 Publishing of Canonical XML Content Cataloging Service

An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in service. This built-in service MUST be published to the registry as part of the intrinsic bootstrapping of required canonical data within the registry.

9 Cooperating Registries Support

This chapter describes the capabilities and protocols that enable multiple ebXML registries to cooperate with each other to meet advanced use cases.

9.1 Cooperating Registries Use Cases

The following is a list of use cases that illustrate different ways that ebXML registries cooperate with each other.

9.1.1 Inter-registry Object References

A Submitting Organization wishes to submit a RegistryObject to a registry such that the submitted object references a RegistryObject in another registry.

An example might be where a RegistryObject in one registry is associated with a RegistryObject in another registry.

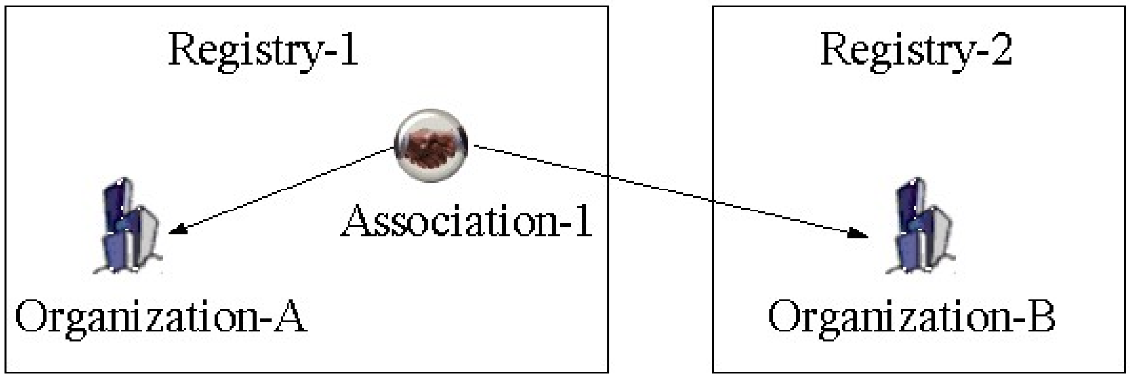


Figure 21: Inter-registry Object References

9.1.2 Federated Queries

A client wishes to issue a single query against multiple registries and get back a single response that contains results based on all the data contained in all the registries. From the client's perspective it is issuing its query against a single logical registry that has the union of all data within all the physical registries.

9.1.3 Local Caching of Data from Another Registry

A destination registry wishes to cache some or all the data of another source registry that is willing to share its data. The shared dataset is copied from the source registry to the destination registry and is visible to queries on the destination registry even when the source registry is not available.

Local caching of data may be desirable in order to improve performance and availability of accessing that object.

An example might be where a RegistryObject in one registry is associated with a RegistryObject in another registry, and the first registry caches the second RegistryObject locally.

9.1.4 Object Relocation

A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from the registry where it was submitted to another registry.

9.2 Registry Federations

A registry federation is a group of registries that have voluntarily agreed to form a loosely coupled union. Such a federation may be based on common business interests and specialties that the registries may share. Registry federations appear as a single logical registry to registry clients.

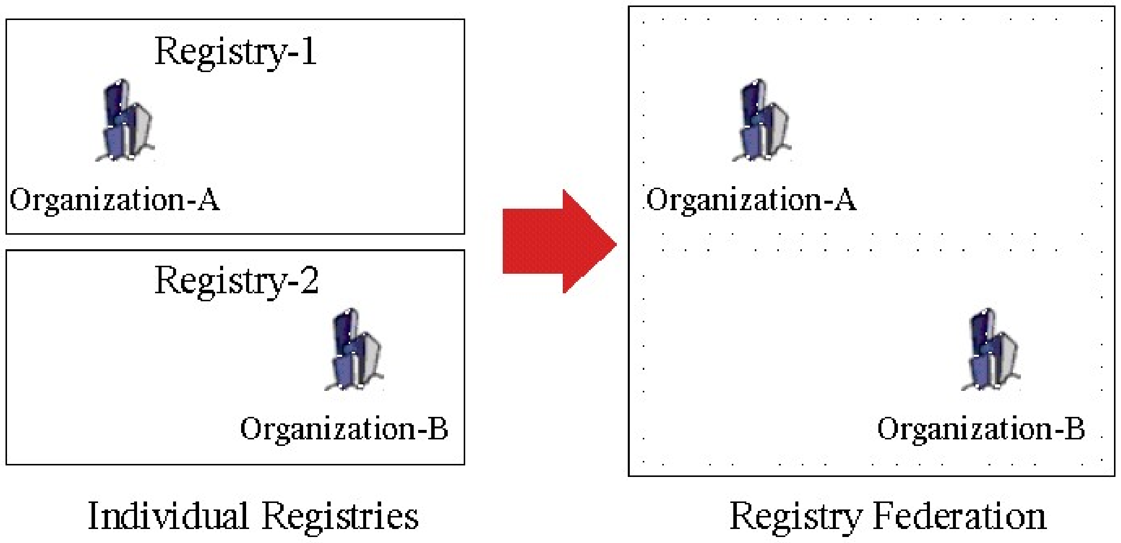


Figure 22: Registry Federations

Registry federations are based on a peer-to-peer (P2P) model where all participating registries are equal. Each participating registry is called a *registry peer*. There is no distinction between the registry operator that created a federation and those registry operators that joined that Federation later.

Any registry operator MAY form a registry federation at any time. When a federation is created it MUST have exactly one registry peer which is the registry operated by the registry operator that created the federation.

Any registry MAY choose to voluntarily join or leave a federation at any time.

9.2.1 Federation Metadata

The Registry Information model defines the Registry and Federation classes. Instances of these classes and the associations between these instances describe a federation and its members. Such instance data is referred to as Federation Metadata. The Registry and Federation classes are described in detail in [ebRIM].

The Federation information model is summarized here as follows:

- A Federation instance represents a registry federation.
- A Registry instance represents a registry that is a member of the Federation.
- An Association instance with associationType of *HasFederationMember* represents membership of the registry in the federation. This Association links the Registry instance and the Federation instance.

Figure 23: Federation Metadata Example

9.2.2 Local Vs. Federated Queries

A federation appears to registry clients as a single unified logical registry. An AdhocQueryRequest sent by a client to a federation member MAY be local or federated. A new boolean attribute named *federated* is added to AdhocQueryRequest to indicate whether the query is federated or not.

9.2.2.1 Local Queries

When the federated attribute of AdhocQueryRequest has the value of *false* then the query is a local query. In the absence of a *federated* attribute the default value of *federated* attribute is *false*.

A local AdhocQueryRequest is only processed by the registry that receives the request. A local AdhocQueryRequest does not operate on data that belongs to other registries.

9.2.2.2 Federated Queries

When the *federated* attribute of AdhocQueryRequest has the value of *true* then the query is a federated query.

A federation member MUST route a federated query received by it to all other federation member registries on a best attempt basis. If a member is not reachable for any reason then it MAY be skipped.

When a registry routes a federated query to other federation members it MUST set the federated attribute value to *false* and the *federation* attribute value to null to avoid infinite loops.

A federated query operates on data that belongs to all members of the federation.

When a client submits a federated query to a registry such that the query specifies no federation and no federations exist in the registry, then the registry MUST treat it as a local query.

When a client submits a federated query that invokes a parameterized stored query, the registry MUST resolve the parameterized stored query into its non-stored formed and MUST replace all variables with user-supplied parameters on registry supplied contextual parameters before routing it to a federation member.

When a client submits a federated iterative query, the registry MUST use the *startIndex* attribute value of the original request as the *startIndex* attribute value of the routed request sent to each federation member. The response to the original request MUST be the *union* of the results from each routed query. In such cases the registry MUST return a *totalResultCount* attribute value on the federated query response to be equal to the *maximum* of all *totalResultCount* attribute values returned by each federation member.

9.2.2.3 Membership in Multiple Federations

A registry MAY be a member of multiple federations. In such cases if the *federated* attribute of AdhocQueryRequest has the value of *true* then the registry MUST route the federated query to *all* federations that it is a member of.

Alternatively, the client MAY specify the id of a specific federation that the registry is a member of, as the value of the *federation* parameter. The type of the federation parameter is anyURI and identifies the "id" attribute of the desired Federation.

In such cases the registry MUST route the federated query to the specified federation only.

9.2.3 Federated Lifecycle Management Operations

Details on how to create and delete federations and how to join and leave a federation are described in 9.2.8.

All lifecycle operations SHOULD be performed on a RegistryObject within its home registry using the operations defined by the LifeCycleManager interface. Unlike query requests, lifecycle management requests do not support any federated capabilities.

9.2.4 Federations and Local Caching of Remote Data

A federation member is not required to maintain a local cache of replicas of RegistryObjects and repository items that belong to other members of the federation.

A registry MAY choose to locally cache some or all data from any other registry whether that registry is a federation member or not. Data caching is orthogonal to registry federation and is described in section 9.3.

Since by default there is minimal replication in the members of a federation, the federation architecture scales well with respect to memory and disk utilization at each registry.

Data replication is often necessary for performance, scalability and fault-tolerance reasons.

9.2.5 Caching of Federation Metadata

A special case for local caching is the caching of the Federation and Registry instances and related Associations that define a federation and its members. Such data is referred to as federation metadata. A federation member is required to locally cache the federation metadata, from the federation home for each federation that it is a member of. The reason for this requirement is consistent with a Peer-to-Peer (P2P) model and ensures fault-tolerance in case the Federation home registry is unavailable.

The federation member MUST keep the cached federation metadata synchronized with the master copy in the Federation home, within the time period specified by the replicationSyncLatency attribute of the Federation. Synchronization of cached Federation metadata may be done via synchronous polling or asynchronous event notification using the event notification feature of the registry.

9.2.6 Time Synchronization Between Registry Peers

Federation members are not required to synchronize their system clocks with each other. However, each Federation member SHOULD keep its clock synchronized with an atomic clock server within the latency described by the replicationSyncLatency attribute of the Federation.

9.2.7 Federations and Security

Federated operations abide by the same security rules as standard operations against a single registry. However, federation operations often require registry-to-registry communication. Such communication is governed by the same security rules as a Registry Client to registry communication. The only difference is that the requesting registry plays the role of Registry Client. Such registry-to-registry communication SHOULD be conducted over a secure channel such as HTTP/S. Federation members SHOULD be part of the same SAML Federation if member registries implement the Registry SAML Profile described in chapter 11.

9.2.8 Federation Lifecycle Management Protocols

This section describes the various operations that manage the lifecycle of a federation and its membership. Federation lifecycle operations are done using standard LifeCycleManager interface of the registry in a stylized manner. Federation lifecycle operations are privileged operations. A registry SHOULD restrict Federation lifecycle operations to registry User's that have the RegistryAdministrator role.

9.2.8.1 Joining a Federation

The following rules govern how a registry joins a federation:

- Each registry SHOULD have exactly one Registry instance within that registry for which it is a home. The Registry instance is owned by the RegistryOperator and may be placed in the registry using any operator specific means. The Registry instance SHOULD never change its home registry.
- A registry MAY request to join an existing federation by submitting an instance of an Extramural Association that associates the Federation instance as sourceObject, to its Registry instance as targetObject, using an associationType of *HasFederationMember*. The home registry for the Association and the Federation objects MUST be the same.

9.2.8.2 Creating a Federation

The following rules govern how a federation is created:

- A Federation is created by submitting a Federation instance to a registry using SubmitObjectsRequest.
- The registry where the Federation is submitted is referred to as the federation home.
- The federation home may or may not be a member of that Federation.
- A federation home MAY contain multiple Federation instances.

9.2.8.3 Leaving a Federation

The following rules govern how a registry leaves a federation:

A registry MAY leave a federation at any time by removing its *HasFederationMember* Association instance that links it with the Federation instance. This is done using the standard RemoveObjectsRequest.

9.2.8.4 Dissolving a Federation

The following rules govern how a federation is dissolved:

- A federation is dissolved by sending a RemoveObjectsRequest to its home registry and removing its Federation instance.
- The removal of a Federation instance is controlled by the same Access Control Policies that govern any RegistryObject.
- The removal of a Federation instance is controlled by the same lifecycle management rules that govern any RegistryObject. Typically, this means that a federation MUST NOT be dissolved while it has federation members. It MAY however be deprecated at any time. Once a

Federation is deprecated no new members can join it.

9.3 Object Replication

RegistryObjects within a registry MAY be replicated in another registry. A replicated copy of a remote object is referred to as its replica. The remote object MAY be an original object or it MAY be a replica. A replica from an original is referred to as a first-generation replica. A replica of a replica is referred to as a second-generation replica (and so on).

The registry that replicates a remote object locally is referred to as the destination registry for the replication. The registry that contains the remote object being replicated is referred to as the source registry for the replication.

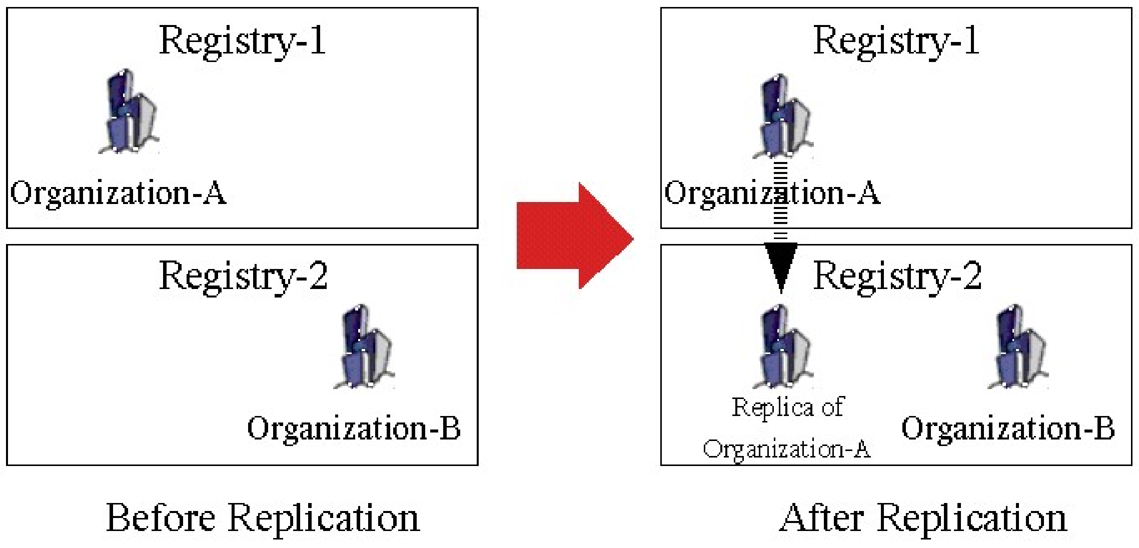


Figure 24: Object Replication

9.3.1 Use Cases for Object Replication

A registry MAY create a local replica of a remote object for a variety of reasons. A few sample use cases follow:

- Improve access time and fault tolerance by locally caching remote objects. For example, a registry MAY automatically create a local replica when a remote ObjectRef is submitted to the registry.
- Improve scalability by distributing access to hotly contested objects, such as NAICS scheme, across multiple replicas.
- Enable cooperating registry features such as hierarchical registry topology and local caching of federation metadata.

9.3.2 Queries And Replicas

A registry MUST support client queries to consider a local replica of remote object as if it were a local object. Local replicas are considered within the extent of the data set of a registry as far as local queries are concerned.

When a client submits a local query that retrieves a remote object by its id attribute, if the registry contains a local replica of that object then the registry SHOULD return the state defined by the local replica.

9.3.3 Lifecycle Operations And Replicas

LifeCycle operations on an original object MUST be performed at the home registry for that object. LifeCycle operations on replicas of an original object should result in an InvalidRequestException.

9.3.4 Object Replication and Federated Registries

Object replication capability is orthogonal to the registry federation capability. Objects MAY be replicated from any registry to any other registry without any requirement that the registries belong to the same federation.

9.3.5 Creating a Local Replica

Any Submitting Organization can create a replica by using the standard SubmitObjectsRequest. If a registry receives a SubmitObjectsRequest that has a RegistryObjectList containing a remote ObjectRef, then it MUST create a replica for that remote ObjectRef. In such cases the User that submitted the ObjectRef (via a SubmitObjectsRequest) owns the replica while the original RegistryObject is owned by its original owner.

In addition to Submitting Organizations, a registry itself MAY create a replica under specific situations in a registry specific manner.

Creating a local replica requires the destination registry to read the state of the remote object from the source registry and then create a local replica of the remote object.

A registry SHOULD use standard QueryManager interface to read the state of a remote object (whether it is an original or a replica). No new APIs are needed to read the state of a remote object. Since query functionality does not need prior registration, no prior registration or contract is needed for a registry to read the state of a remote object.

Once the state of the remote object has been read, a registry MAY use registry specific means to create a local replica of the remote object. Such registry specific means MAY include the use of the LifeCycleManager interface.

A replica of a RegistryObject may be distinguished from an original since a replica MUST have its home attribute point to the remote registry where the original for the replica resides.

9.3.6 Transactional Replication

Transactional replication enables a registry to replicate events in another registry in a transactionally consistent manner. This is typically the case when entire registries are replicated to another registry.

This specification defines a more loosely coupled replication model as an alternative to transactional replication for the following reasons:

- Transactional replication requires a tight coupling between registries participating in the replication
- Transactional replication is not a typical use case for registries
- Loosely coupled replication as defined by this specification typically suffices for most use cases
- Transaction replication is very complex and error prone

Registry implementations are not required to implement transactional replication.

9.3.7 Keeping Replicas Current

A registry MUST keep its replicas current within the latency specified by the value of the *replicationSyncLatency* attribute defined by the registry. This includes removal of the replica when its original is removed from its home registry.

Replicas MAY be kept current using the event notification feature of the registry or via periodic polling.

9.3.8 Lifecycle Management of Local Replicas

Local Replicas are read-only objects. Lifecycle management actions are not permitted on local replicas with the exception of the Delete action which is used to remove the replica. All other lifecycle management actions MUST be performed on the original RegistryObject in the home registry for the object.

9.3.9 Tracking Location of a Replica

A local replica of a remote RegistryObject instance MUST have exactly one ObjectRef instance within the local registry. The home attribute of the ObjectRef associated with the replica tracks its home location. A RegistryObject MUST have exactly one home. The home for a RegistryObject MAY change via Object Relocation as described in section 9.4. It is optional for a registry to track location changes for replicas within it.

9.3.10 Remote Object References to a Replica

It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another RegistryObject. In such cases the home attribute of the ObjectRef contains the base URI to the home registry for the replica.

9.3.11 Removing a Local Replica

A client can remove a replica by using the RemoveObjectsRequest. If a registry receives a RemoveObjectsRequest that has an ObjectRefList containing a remote ObjectRef, then it MUST remove the local replica for that remote ObjectRef assuming that the client was authorized to remove the replica.

9.4 Object Relocation Protocol

Every RegistryObject has a home registry and a User within the home registry that is the Submitter or owner of that object. Initially, the home registry is the where the object is originally submitted. Initially, the owner is the User that submitted the object.

A RegistryObject MAY be relocated from one home registry to another home registry using the Object Relocation protocol.

Within the Object Relocation protocol, the new home registry is referred to as the *destination* registry while the previous home registry is called the *source* registry.

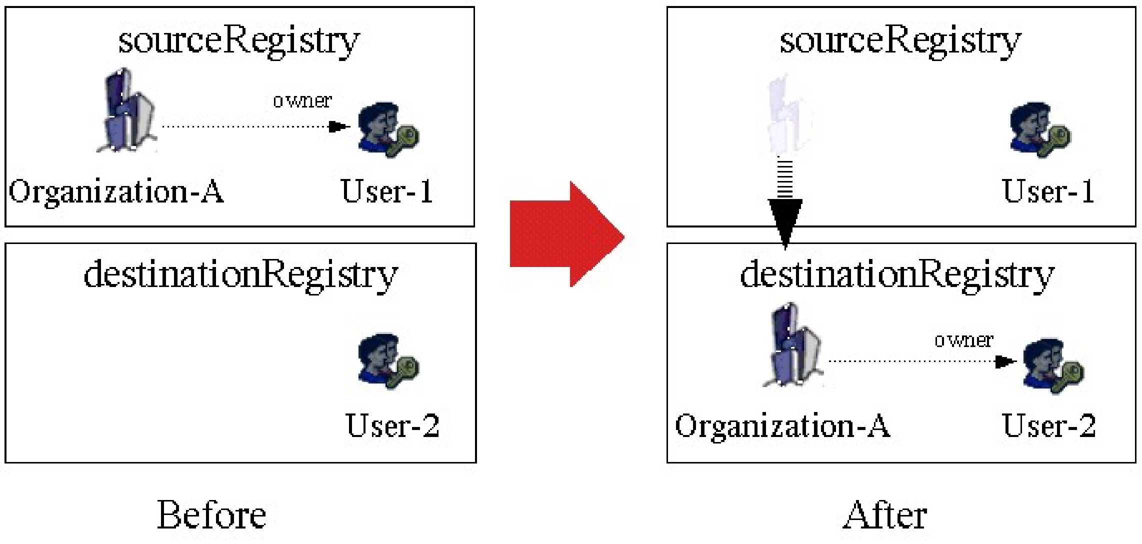


Figure 25: Object Relocation

The User at the source registry who owns the objects being relocated is referred to as the *ownerAtSource*. The User at the destination registry, who is the new owner of the objects, is referred to as the *ownerAtDestination*. While the ownerAtSource and the ownerAtDestination may often be the same, the Object Relocation protocol treats them as two distinct identities.

A special case usage of the Object Relocation protocol is to transfer ownership of RegistryObjects from one User to another within the same registry. In such cases the protocol is the same except for the fact that the source and destination registries are the same.

Following are some notable points regarding object relocation:

- Object relocation does not require that the source and destination registries be in the same federation or that either registry have a prior contract with the other.
- Object relocation MUST preserve object id. While the home registry for a RegistryObject MAY change due to object relocation, its id never changes.
- ObjectRelocation MUST preserve referential integrity of RegistryObjects. Relocated objects that have references to an object that did not get relocated MUST preserve their reference. Similarly objects that have references to a relocated object MUST also preserve their reference. Thus, relocating an object may result in making the value of a reference attribute go from being a local reference to being a remote reference or vice versa.
- AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque transactonId identifying the relocateObjects transaction.
- The requests defined by the Relocate Objects protocol MUST be sent to the source or destination registry only.
- When an object is relocated an AuditableEvent of type “Relocated” MUST be recorded by the sourceRegistry. Relocated events MUST have the source and destination registry’s base URIs recorded as two Slots on the Relocated event. The names of these Slots are:
    - `urn:oasis:names:tc:ebxml-regrep:rs:events:sourceRegistry`
    - `urn:oasis:names:tc:ebxml-regrep:rs:events:destinationRegistry`

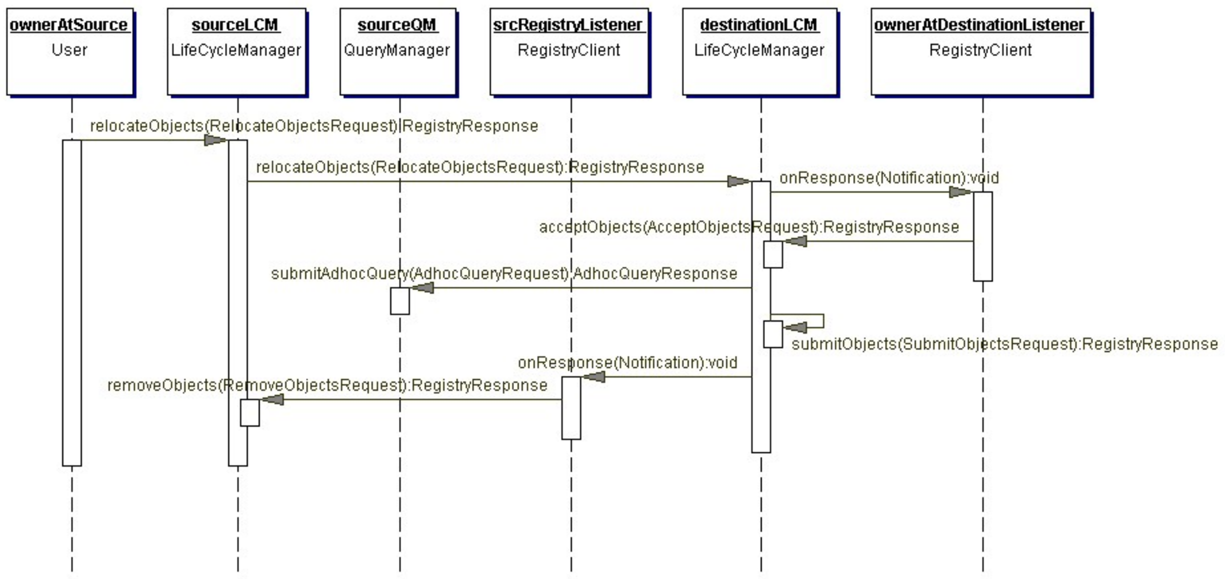

ownerAtSource
User
sourceLCM
LifeCycleManager
sourceQM
QueryManager
srcRegistryListener
RegistryClient
destinationLCM
LifeCycleManager
ownerAtDestinationListener
RegistryClient
relocateObjects(RelocateObjectsRequest):RegistryResponse
relocateObjects(RelocateObjectsRequest):RegistryResponse
onResponse(Notification):void
acceptObjects(AcceptObjectsRequest):RegistryResponse
submitAdhocQuery(AdhocQueryRequest):AdhocQueryResponse
submitObjects(SubmitObjectsRequest):RegistryResponse
onResponse(Notification):void
removeObjects(RemoveObjectsRequest):RegistryResponse

Figure 26: Relocate Objects Protocol

Figure 26 illustrates the Relocate Objects Protocol. The participants in the protocol are the ownerAtSource and ownerAtDestination User instances as well as the LifeCycleManager interfaces of the sourceRegistry and destinationRegistry.

The steps in the protocol are described next:

1. The protocol is initiated by the ownerAtSource sending a RelocateObjectsRequest message to the LifeCycleManager interface of the sourceRegistry. The sourceRegistry MUST make sure that the ownerAtSource is authorized to perform this request. The id of this RelocateObjectsRequest is used as the transaction identifier for this instance of the protocol. This RelocateObjectsRequest message MUST contain an ad hoc query that specifies the objects that are to be relocated.

2. Next, the sourceRegistry MUST relay the same RelocateObjectsRequest message to the LifeCycleManager interface of the destinationRegistry. This message enlists the detsinationRegistry to participate in relocation protocol. The destinationRegistry MUST store the request information until the protocol is completed or until a registry specific period after which the protocol times out.

3. The destinationRegistry MUST relay the RelocateObjectsRequest message to the ownerAtDestination. This notification MAY be done using the event notification feature of the registry as described in chapter 7. The notification MAY be done by invoking a listener Service for the ownerAtDestination or by sending an email to the ownerAtDestination. This concludes the first phase of the Object Relocation protocol.

4. The ownerAtDestination at a later time MAY send an AcceptObjectsRequest message to the destinationRegistry. This request MUST identify the object relocation transaction via the *correlationId*. The value of this attribute MUST be the id of the original RelocateObjectsRequest.

5. The destinationRegistry sends an AdhocQueryRequest message to the sourceRegistry. The source registry returns the objects being relocated as an AdhocQueryResponse. In the event of a large number of objects this may involve multiple AdhocQueryRequest/responses as described by the iterative query feature described in section 6.2.

6. The destinationRegistry submits the relocated data to itself assigning the identity of the ownerAtDestination as the owner. The relocated data MAY be submitted to the destination registry using any registry specific means or a SubmitObjectsRequest. However, the effect SHOULD be the same as if a SubmitObjectsRequest was used.

7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been safely committed using the Event Notification feature of the registry as described in chapter 7.

8. The sourceRegistry removes the relocated objects using any registry specific means and logging an AuditableEvent of type Relocated. This concludes the Object Relocation transaction.

9.4.1 RelocateObjectsRequest

```
<element name="RelocateObjectsRequest">
   <complexType>
     <complexContent>
       <extension base="rs:RegistryRequestType">
         <sequence>
           <element name="Query" type="rim:AdhocQueryType"/>
           <element name="SourceRegistry" type="rim:ObjectRefType"/>
           <element name="DestinationRegistry"
type="rim:ObjectRefType"/>
           <element name="OwnerAtSource" type="rim:ObjectRefType"/>
           <element name="OwnerAtDestination"
type="rim:ObjectRefType"/>
         </sequence>
       </extension>
     </complexContent>
   </complexType>
 </element>
```

9.4.1.1 Parameters:

id: the attribute id provides the transaction identifier for this instance of the protocol.

AdhocQuery: This element specifies an ad hoc query that selects the RegistryObjects that are being relocated.

sourceRegistry: This element specifies the ObjectRef to the sourceRegistry Registry instance. The value of this attribute MUST be a local reference when the message is sent by the ownerAtSource to the sourceRegistry.

destinationRegistry: This element specifies the ObjectRef to the destinationRegistry Registry instance.

ownerAtSource: This element specifies the ObjectRef to the ownerAtSource User instance.

ownerAtDestination: This element specifies the ObjectRef to the ownerAtDestination User instance.

9.4.1.2 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

9.4.1.3 Exceptions:

In addition to the exceptions common to all requests, the following exceptions MAY be returned:

ObjectNotFoundException: signifies that the specified Registry or User was not found in the registry.

9.4.2 AcceptObjectsRequest

```
<element name="AcceptObjectsRequest">
```

```
   <complexType>
     <complexContent>
       <extension base="rs:RegistryRequestType">
         <attribute name="correlationId" use="required"
type="{http://www.w3.org/2001/XMLSchema}anyURI" />
       </extension>
     </complexContent>
   </complexType>
 </element>
```

9.4.2.1 Parameters:

correlationId: Provides the transaction identifier for this instance of the protocol.

9.4.2.2 Returns:

This request returns a RegistryResponse. See section 2.1.4 for details.

9.4.2.3 Exceptions:

In addition to the exceptions common to all requests, the following exceptions MAY be returned:

InvalidRequestException: signifies that the specified correlationId was not found to match an ongoing RelocateObjectsRequest in the registry.

9.4.3 Object Relocation and Remote ObjectRefs

The following scenario describes what typically happens when a person moves:

1. When a person moves from one house to another, other persons may have their old postal addresses.
2. When a person moves, they leave their new address as the forwarding address with the post office.
3. The post office forwards their mail for some time to their new address.
4. Eventually the forwarding request expires and the post office no longer forwards mail for that person.
5. During this forwarding interval the person notifies interested parties of their change of address.

The Object Relocation feature supports a similar model for relocation of RegistryObjects. The following steps describe the expected behavior when an object is relocated.

1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other RegistryObjects may have remote ObjectRefs to O1.
2. The registry R1 MUST create an AuditableEvent of type Relocated that includes the home URI for the new registry R2.
3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it MUST forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the client. The object O1 MUST include the home URI to R2 within the optional home attribute of RegistryObject. Clients are advised to check the home attribute and update the home attribute of their local ObjectRef to match the new home URI value for the object.
4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no longer required to relay requests for O1 to R2 transparent to the client. Instead R1 MUST return an ObjectNotFoundException.

5. Clients that are interested in the relocation of O1 and being notified of its new address may choose to be notified by having a prior subscription using the event notification facility of the registry. For example a Registry that has a remote ObjectRefs to O1 may create a subscription on relocation events for O1. This however, is not required behavior.

9.4.4 Notification of Object Relocation To ownerAtDestination

This section describes how the destinationRegistry uses the event notification feature of the registry to notify the ownerAtDestination of a Relocated event.

The destinationRegistry MUST send a Notification with the following required characteristics:

- The notification MUST be an instance of a Notification element.
- The Notification instance MUST have at least one Slot as follows:
  - The Slot MUST have the name: `urn:oasis:names:tc:ebxml-regrep:rs:events:correlationId`
  - The Slot MUST have the correlationId for the Object Relocation transaction as the value of the Slot.

9.4.5 Notification of Object Commit To sourceRegistry

This section describes how the destinationRegistry uses the event notification feature of the registry to notify the sourceRegistry that it has completed committing the relocated objects.

The destinationRegistry MUST send a Notification with the following required characteristics:

- The notification MUST be an instance of a Notification element.
- The Notification instance MUST have at least one Slot as follows:
  - The Slot MUST have the name `urn:oasis:names:tc:ebxml-regrep:rs:events:objectsCommitted`
  - The Slot MUST have the value of *true*.

9.4.6 Object Ownership and Owner Reassignment

A registry MUST determine the ownership of a RegistryObject based upon the most recent AuditableEvent that has the eventType matching the canonical EventType ClassificationNode for Create or Relocate events.

A special case of Object Relocation is when an ObjectRelocationRequest to a registry specifies the same registry as sourceRegistry and destinationRegistry. In such cases the request is effectively to change the owner of the specified objects from current owner to a new owner.

In such case if the client does not have the RegistryAdministrator role then the protocol requires the ownerAtDestination to issue an AcceptObjectsRequest as described earlier.

However, if the client does have the RegistryAdministrator role then the registry MUST change the owner of the object to the user specified as ownerAtDestination without the ownerAtDestination to issue an AcceptObjectsRequest.

9.4.7 Object Relocation and Timeouts

No timeouts are specified for the Object Relocation protocol. Registry implementations MAY cleanup incomplete Object Relocation transactions in a registry specific manner as an administrative task using registry specific policies.

10 Registry Security

This chapter describes the security features of ebXML Registry. A glossary of security terms can be referenced from [RFC 2828]. The registry security specification incorporates by reference the following specifications:

- [WSI-BSP] WS-I Basic Security Profile 1.0
- [WSS-SMS] Web Services Security: SOAP Message Security 1.0
- [WSS-SWA] Web Services Security: SOAP Messages with Attachments (SwA) Profile 1.0

This chapter provides registry specific details not present in above specifications.

10.1 Security Use Cases

This section describes various use cases that require security features from the registry. Subsequent sections describe specific registry mechanisms that enable each of these use cases.

10.1.1 Identity Management

An organization deploys an ebXML Registry and needs to define the set of users and services that are authorized to use the services offered by the registry. They require that the registry provide some mechanism for registering and subsequently managing the identity and credentials associated with such authorized users and services.

10.1.2 Message Security

A Registered User sends a request message to the registry and receives a response back from the registry. The user requires that the message integrity be protected during transmission from tampering (man-in-the-middle attack). The user may also require that the message communication is not available to unauthorized parties (confidentiality).

10.1.3 Repository Item Security

A Registered User submits a repository item to the registry. The user requires that the registry provide mechanisms to protect the integrity of the repository item during transmission on the wire and as long as it is stored in the registry. The user may also require that the content of the RepositoryItem is not available to unauthorized parties (confidentiality).

10.1.4 Authentication

An organization that deploys an ebXML Registry requires that when a Registered User sends a request to the registry, the registry checks the credentials provided by the user to ensure that the user is a Registered User and to unambiguously determine the user's identity.

10.1.5 Authorization and Access Control

An organization that deploys an ebXML Registry requires that the registry provide a mechanism that protect its resources from unauthorized access. Specifically, when a Registry Requestor sends a request to the registry, the registry restricts the actions of the requestor to specific actions on specific resources for which the requestor is authorized.

10.1.6 Audit Trail

An organization that deploys an ebXML Registry requires that the registry keep a journal or Audit Trail of all significant actions performed by Registry Requestors on registry resources. This provides a basic form of non-repudiation where a Registry Requestor cannot repudiate that that they performed actions that are logged in the Audit Trail.

10.2 Identity Management

An ebXML Registry MUST provide an Identity Management mechnism that allows identities and credentials to be registered for authorized users of the registry and subsequently managed.

If a registry implements the Registry SAML Profile as described in chapter 11 then the Identity Management capability MUST be provided by an Identity Provider service that integrates with the registry using the SAML 2.0 protocols as defined by [SAMLCore].

If a registry does not implement the Registry SAML Profile then it MUST provide User Registration and Identity Management functionality in an implementation specific manner.

10.3 Message Security

A registry MUST provide mechanisms to securely exchange messages between a Registry Requestor and the registry to ensure data and source integrity as described in this section.

10.3.1 Transport Layer Security

A registry MUST support HTTP/S communication between an HTTP Requestor and its HTTP interface binding. A registry MUST also support HTTP/S communication between a SOAP Requestor and its SOAP interface binding when the underlying transport protocol is HTTP.

HTTP/S support SHOULD allow for both SSL and TLS as transport protocols.

10.3.2 SOAP Message Security

A registry MUST support signing and verification of all registry protocol messages (requests and responses) between a SOAP Requestor and its SOAP binding. Such mechanisms MUST conform to [WSI-BSP], [WSS-SMS], [WSS-SWA] and [XMLDSIG]. The reader should refer to these specifications for details on these message security mechanisms.

10.3.2.1 Request Message Signature

When a Registered User sends a request message to the registry, the requestor SHOULD sign the request message with a Message Signature. This ensures the integrity of the message and also enables the registry to perform authentication and authorization for the request. If the registry receives a request that does not include a Message signature then it MUST implicitly treat the request as coming from a Registry Guest. A Registered User need not sign a request message with a Message Signature when the SOAP communication is conducted over HTTP/S as the message security is handled by the transport layer security provided by HTTP/S in this case.

When a Registered User sends a request message to the registry that contains a RepositoryItem as a SOAP Attachment, the requestor MUST also reference and sign the RepositoryItem from the message signature. This MUST conform to [RFC2392] and [WSS-SWA].

If the registry receives a request containing an unsigned RepositoryItem then it MUST return an UnsignedRepositoryItemException.

10.3.2.2 Response Message Signature

When a Registered User sends a request message to the registry, the registry MAY use a pre-established preference policy or a default policy to determine whether the response message SHOULD be signed with a Message Signature. When a Registry Guest sends a request, the Registration Authority MAY use a default policy to determine whether the response contains a header signature. A registry need not sign a response message with a Message Signature when the SOAP communication is conducted over HTTP/S as the message security is handled by the transport layer security provided by HTTP/S in this case.

When a registry sends a signed response message to a Registry Client that contains a RepositoryItem as a SOAP Attachement, the registry MUST also reference and sign the RepositoryItem from the message signature. This MUST conform to [RFC2392] and [WSS-SWA].

If the Registry Client receives a signed response with a RepositoryItem that does not include a RepositoryItem Signature then it SHOULD not trust the integrity of the response and treat it as an error condition.

10.3.2.3 KeyInfo Requirements

The sender of a registry protocol message (Registry Requestor and Registry) SHOULD provide their public key under the <wsse:Security> element. If provided, it MUST be contained in a <wsse:BinarySecurityToken> element and MUST be referenced from the <ds:KeyInfo> element in the Message Signature. The value of wsu:Id attribute of the <wsse:BinarySecurityToken> containing the senders public key MUST be **`urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert`**.

The <wsse:BinarySecurityToken> SHOULD contain a X509 Certificate.

Listing 3 shows an example of Message signature including specifying the KeyInfo.

10.3.2.4 Message Signature Validation

Signature validation ensures message and attached RepositoryItems integrity and security, concerning both data and source.

If the registry receives a request containing a Message Signature then it MUST validate the Message Signature as defined by [WSS-SMS]. In case the request contains an attached RepositoryItem it MUST validate the RepositoryItems signature as defined by [WSS-SWA].

If the Registry Requestor receives a response containing a Message Signature then it SHOULD validate the Message Signature as defined by [WSS-SMS]. In case the response contains an attached RepositoryItem then it SHOULD validate the RepositoryItem signature as defined by [WSS-SWA].

10.3.2.5 Message Signature Example

The following example shows the format of a Message Signature:

```
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">
        lui+Jy4WYKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnW
SWkXm9jAEdsm/
        hs+f3NwvK23bh46mNmnCQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yT
cI7XU7xZT54S9
        hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrsFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI
9WzxPCfHdalN4
        rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUc9QY3VjwNALgGDaEAT7gpURkCI8
5HjdnSA5SM4cY
        7jAsYX/CIpEkRJcBULlTEFrBZIBYDPzRWlSdsJRJngF7yCoGWJ+/HYOyP8P4OM5
9FDi0kM8GwOE0
        WgYrJHH92qaVhoiPTLi7
      </wsse:BinarySecurityToken>
      <ds:Signature>
                          <!--The Message Signature -->
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
            <c14n:InclusiveNamespaces PrefixList="wsse soap"
xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#TheBody">
```

```
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#">
                <c14n:InclusiveNamespaces PrefixList=""
xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValu
e>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureVa
lue>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body wsu:Id="TheBody">
    <lcm:SubmitObjectsRequest/>
  </soap:Body>
</soap:Envelope>
```

Listing 3: Message Signature Example

10.3.2.6 Message With RepositoryItem: Signature Example

The following example shows the format of a Message Signature that also signs the attached RespositoryItem:

```
Content-Type: multipart/related; boundary=”BoundaryStr” type=”text/xml”
--BoundaryStr
Content-Type: text/xml
<soap:Envelope>
  <soap:Header>
    <wsse:Security>
      <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">
        lui+Jy4WYKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnW
SWkXm9jAEdsm/
        hs+f3NwvK23bh46mNmnCQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yT
cI7XU7xZT54S9
        hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrsFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI
9WzxPCfHdalN4
        rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUc9QY3VjwNALgGDaEAT7gpURkCI8
5HjdnSA5SM4cY
        7jAsYX/CIpEkRJcBULlTEFrBZIBYDPzRWlSdsJRJngF7yCoGWJ+/HYOyP8P4OM5
9FDi0kM8GwOE0
        WgYrJHH92qaVhoiPTLi7
      </wsse:BinarySecurityToken>
      <ds:Signature>
        <!-- The Message Signature -->
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
            <c14n:InclusiveNamespaces PrefixList="wsse soap"
xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#TheBody">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#">
                <c14n:InclusiveNamespaces PrefixList=""
xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValu
e>
          </ds:Reference>
        </ds:SignedInfo>

        <!--A reference to a RepositoryItem (one for each
RepositoryItem) -->
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
            <c14n:InclusiveNamespaces PrefixList="wsse soap"
xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="cid:${REPOSITORY_ITEM1_ID}">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
exc-c14n#">
              <ds:Transform Algorithm="http://docs.oasis-
open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-
Content-Only-Transform"/>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValu
e>
          </ds:Reference>
        </ds:SignedInfo>

        <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureVa
lue>

        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>

      </ds:Signature>
    </wsse:Security>
  </soap:Header>
  <soap:Body wsu:Id="TheBody">
    <lcm:SubmitObjectsRequest/>
  </soap:Body>
</soap:Envelope>
--BoundaryStr
Content-Type: image/png
Content-ID: <${REPOSITORY_ITEM1_ID}>
Content-Transfer-Encoding: base64
the repository item (e.g. PNG Image) goes here..
```

Listing 4: RepositoryItem Signature Example

10.3.2.7 SOAP Message Security and HTTP/S

When using HTTP/S between a Registry Client and a registry, SOAP message security MUST NOT be used. Specifically:

- The Registry Client MUST NOT sign the request message or any repository items in the request.
- The registry MUST NOT verify request or RepositoryItem signatures.
- The registry MUST NOT sign the response message or any repository items in the response.
- The Registry Client MUST NOT verify response or RepositoryItem signatures.

10.3.3 Message Confidentiality

A registry SHOULD support encryption of protocol messages as defined section 9 of [WSI-BSP] as a mechanism to support confidentiality of protocol messages during transmission on the wire.

A Registry Client MAY use encryption of RepositoryItems as defined by [WSS-SWA] as a mechanism to support confidentiality of RepositoryItems during transmission on the wire.

A registry SHOULD support the submission of encrypted repository items.

10.3.4 Key Distribution Requirements

The registry and Registered Users MUST mutually exchange their public keys. This is necessary to enable:

- Mutual Authentication of Registry Client and registry using SSL/TLS handshake for transport layer security over HTTP/S
- Validation of Message Signature and RepositoryItem Signature (described in section).
- Decryption of encrypted messages

In order to enable Message Security the following requirements MUST be met:

1. A Certificate is associated with the registry.
2. A Certificate is associated with Registry Client.
3. A Registry Client registers its public key certificate with the registry. This is typically done during User Registration and is implementation specific.
4. Registry Client obtains the registry's public key certificate and stores it in its own local key store. This is done in an implementation specific manner.

10.4 Authentication

The Registry MUST be able to authenticate the identity of the User associated with client requests in order to perform authorization and access control and to maintain an Audit Trail of registry access. In security terms a service that provides the ability to authenticate requestors is referred to as an Authentication Authority.

A registry MUST provide one or more of the following Authentication mechanisms:

- Registry as Authentication Authority
- External Authentication Authority

10.4.1 Registry as Authentication Authority

A registry MAY provide authentication capability by serving as an Authentication Authority. In this role the registry uses the <ds:KeyInfo> in the Message Signature as credentials to authenticate the requestor. This typically requires checking that the public key supplied in the <ds:KeyInfo> of the Message Signature matches the public key of a Registered User. This also requires that the registry maintain a “registry keystore” that contains the public keys of Registered Users. The remaining details of registry as an authentication authority are implementation specific.

Alternatively, if the Registry Client communicates with the registry over HTTP/S, the registry MUST authenticate the Registry Client User if a registered certificate is provided through SSL Client Authentication. If the certificate is not known to the registry then the Registry MUST assign the RegistryGuest principal with the Registry Client.

10.4.2 External Authentication Authority

A registry MAY also use an external Authentication Authority to auhenticate client requests. The use of an external Authentication Authority requires that the registry implement the Registry SAML Profile as described in chapter 11.

10.4.3 Authenticated Session Support

Once a request is authenticated a Registry SHOULD establish an authenticated session using implementation specific means to avoid having to re-authenticate subsequent request from the same requestor. When the underlying transport protocol is HTTP, a registry SHOULD implement authenticated session support based upon HTTP session capability as defined by [RFC2965].

10.5 Authorization and Access Control

Once a registry has authenticated the identity of the Registered User associated with a client request it MUST perform authorization and subsequently enforce access control rules based upon the authorization decision.

Authorization and access control is an operation conducted by the registry that decides WHO can do WHAT ACTION on WHICH RESOURCE.

- The WHO is the User determined by the authentication step.
- The WHAT ACTION is determined by the registry protocol request sent by the client.
- The WHICH RESOURCE consists of the RegistryObjects and RepositoryItems impacted by the registry protocol request.

The Access Control Policy associated with the resource that is impacted by the action determines authorization and access control.

A registry MUST provide an access control and authorization mechanism based upon chapter titled “Access Control Information Model” in [ebRIM]. This model defines a default access control policy that MUST be supported by the registry. In addition it also defines a binding to [XACML] that allows fine-grained access control policies to be defined.

10.6 Audit Trail

Once a registry has performed authorization checks, enforced access control and allowed a client request to proceed it services the client request. A registry MUST create an Audit Trail of all

LifeCycleManager operations. A registry MAY create an Audit Trail of QueryManager operations. To conserve storage resources, a registry MAY prune the Audit Trail information it stores in an implementation specific manner. A registry SHOULD perform such pruning by removing the older information in its Audit Trail content. However, it MUST not remove the original Create Event at the beginning of the audit trail since the Create Event establishes the owner of the RegistryObject.

Details of how a registry maintains an Audit Trail of client requests is described in the chapter title “Event Information Model” of [ebRIM].

11 Registry SAML Profile

This chapter defines the Registry SAML Profile that a registry MAY implement in order to support SAML 2.0 protocols defined by [SAMLCore]. A specific focus of the Registry SAML Profile is the Web Single Sign On (SSO) profile defined by [SAMLProf].

11.1 Terminology

The reader should refer to the SAML Glossary [SAMLGloss] for various terms used in the Registry SAML profile. A few terms are described here for convenience:

Term	Definition
Authentication Authority	An Authentication Authority is a system entity (typically a service) that enables other system entities (typically a user or service) to establish an authenticated session by proving their identity by providing necessary credentials (e.g. username / password, certificate alias / password). An Authentication Authority produces authentication assertions as a result of successful authentication.
Enhanced Client Proxy (ECP)	Describes a client that operates under certain constraints such as not being able to support HTTP Redirect protocol. Typically these are clients that do not have a Web Browser environment. In this document the main example of an ECP is a Registry Client that uses SOAP to communicate with the registry (SOAP Requestor).
Identity Provider (IdP)	A kind of *service provider* that creates, maintains, and manages identity information for *principals (e.g. users).* An Identity Provider is usually also an Authentication Authority.
Principal	A system entity whose identity can be authenticated. This maps to User in [ebRIM].
SAML Requestor	A *system entity* that utilizes the SAML protocol to request services from another system entity (a *SAML authority*, a *responder*). The term "client" for this notion is not used because many system entities simultaneously or serially act as both clients and servers.
Service Provider (SP)	A role donned by a system entity where the system entity provides services to principals or other system entities. The Registry Service is a SP
Single Sign On (SSO)	The ability to share a single authenticated session across multiple SSO enabled services and application. The client may establish the authenticated session by authenticating with any Authentication Authority within the system. The client may then perform secure operations with any SSO enabled service within the system using the authenticated session.
Single Logout	The ability to logout nearly simultaneously from multiple Service Providers within a federated system.

11.2 Use Cases for SAML Profile

The Registry SAML Profile is intended to address following use cases using the protocols defined by [SAMLCore].

11.2.1 Registry as SSO Participant:

A large enterprise is deploying an ebXML Registry. The enterprise already has an existing Identity Provider (e.g. an Access Manager service) where it maintains user information and credentials. The enterprise also has an existing Authentication Authority (which may be the same service as the Identity Provider) that is used to authenticate users and enable Single Sign On (SSO) across all their enterprise services applications.

The enterprise wishes to use its existing Identity Provider to manage registry users and to avoid duplicating the user database contained in the Identity Provider within the registry. The enterprise also wishes to use its existing Authentication Authority to authenticate registry users and expects the registry to participate in SSO capability provided by their Authentication Authority service.

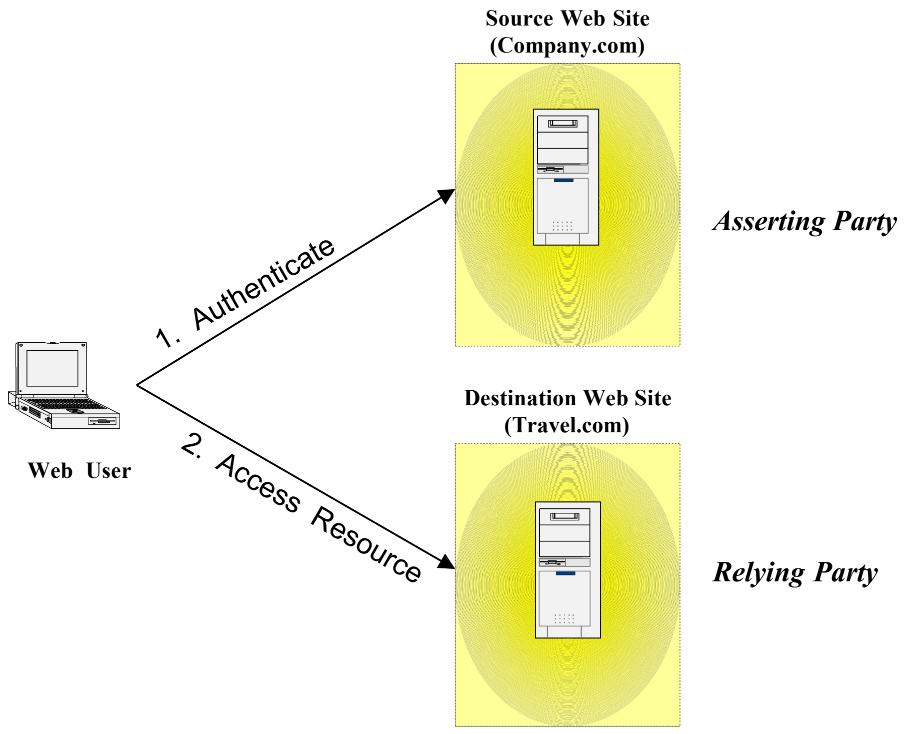


Figure 27: SAML SSO Typical Scenario

11.3 SAML Roles Played By Registry

In order to conform to the registry SAML Profile an ebXML Registry plays the Service Provider (SP) role based upon conformance with SAML 2.0 protocols.

11.3.1 Service Provider Role

The Service Provider role enables the registry to participate in SAML protocols. Specifically it allows the registry to utilize an Identity Provider to perform client authentication on its behalf.

11.3.1.1 Service Provider Requirements

The following are a list of requirements for the Service Provider role of the registry:

- MUST support the protocols, messages and bindings that are the responsibility of the Service Provider as defined by Web SSO Profile in [SAMLProf]. Specifically it MUST be able to intiate and participate in the Authentication Request Protocol with an Identity Provider.
- MUST be able to use a SAML Identity Provider to authenticate client requests.
- MUST support the ability to maintain a security context for registry clients across multiple client requests.

11.4 Registry SAML Interface

In order to conform to the registry SAML Profile an ebXML Registry MUST implement a new SAML interface in addition to its service interfaces such as QueryManager and LifeCycleManager.

Details of the registry's SAML interface are not described by this specification. Instead they are described by the SAML 2.0 specifications and MUST support SAML HTTP and SOAP requests.

A registry uses its SAML interface to participate in SAML protocols with SAML Clients and SAML Identity Providers. Specifically, an IdentityProvider uses the registry's SAML Service Provider interface to deliver the Response to an Authentication Request.

11.5 Requirements for Registry SAML Profile

In order to conform to the Registry SAML Profile a registry MUST implement specific SAML protocol that support specific SAML protocol message exchanges using specific protocol bindings.

Table 7 lists the matrix of SAML Profiles, Protocols Messages and their Bindings that a registry MUST support in order to conform to the registry SAML Profile.

The reader should refer to:

- [SAMLProf] for description of profiles listed
- [SAMLCore] for description of Message Flows listed
- [SAMLBind] for description of Bindings listed

Profile	Message Flows	Binding	Implementation Requirement
Web SSO	`<AuthnRequest>` from Registry to IdentityProvider	HTTP redirect	MUST
	IdentityProvider `<Response>` to Registry	HTTP POST	MUST
		HTTP artifact	MUST
Single Logout	`<LogoutRequest>`	HTTP redirect	MUST
		SOAP	MAY
	`<LogoutResponse>`	HTTP redirect	MUST
		SOAP	MAY
Artifact Resolution	`<ArtifactResolve>`,	SOAP	MUST
	`<ArtifactResponse>`	SOAP	MUST
Enhanced Client/Proxy SSO	ECP to Registry, Registry to ECP to IdentityProvider	PAOS	MUST
	IdentityProvider to ECP to Registry, Registry to ECP	PAOS	MUST

Table 7: Required SAML Profiles, Protocols and Bindings

11.6 SSO Operation

This section describes the interaction sequnce for various types of SSO operations.

11.6.1 Scenario Actors

The following are the actors that will be participating the various SSO Operation scenarios described in subsequent section:

- HTTP Requestor: This represents a Registry Client that accesses the registry using the HTTP binding of the registry protocols typically through a User Agent such as a Web Browser.
- SOAP Requestor: This represents a Registry Client that accesses the registry using the SOAP binding of the registry protocols.
- Registry: This represents a Registry and includes all Registry interfaces such as QueryManager, LifeCycleManager and the registry's SAML Service Provider. The Registry participates in ebXML Registry protocols as well as SAML protocols.
- IdentityProvider: This represents the IdentityProvider used by the registry to perform Authentication on its behalf.

11.6.2 SSO Operation – Unauthenticated HTTP Requestor

Figure 28 shows a high level view of the Single Sign On (SSO) operation when the SOAP Requestor is unauthenticated and accesses the registry over HTTP via a User Agent such as a Web Browser.

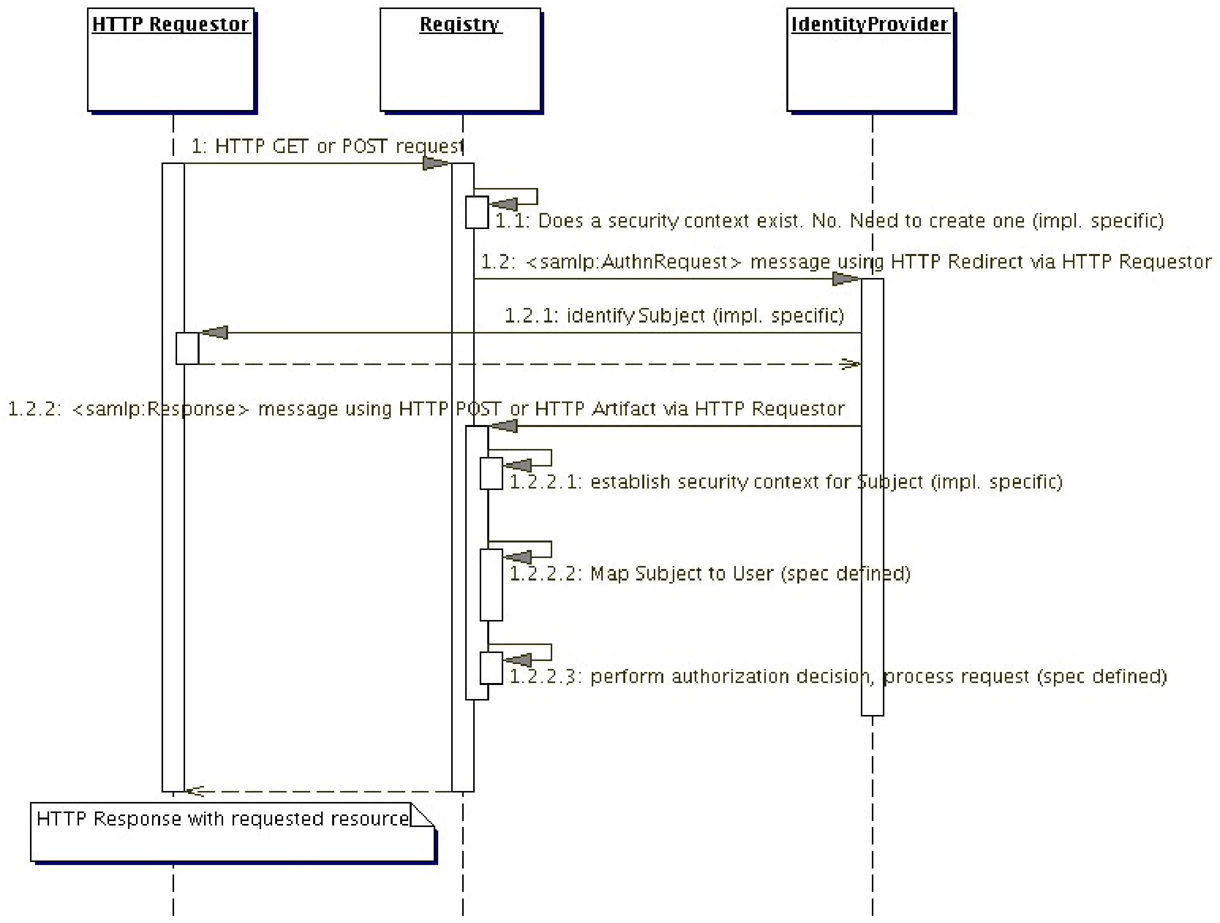

HTTP Requestor
Registry
IdentityProvider
1: HTTP GET or POST request
1.1: Does a security context exist. No. Need to create one (impl. specific)
1.2: <samlp:AuthnRequest> message using HTTP Redirect via HTTP Requestor
1.2.1: identifySubject (impl. specific)
1.2.2: <samlp:Response> message using HTTP POST or HTTP Artifact via HTTP Requestor
1.2.2.1: establish security context for Subject (impl. specific)
1.2.2.2: Map Subject to User (spec defined)
1.2.2.3: perform authorization decision, process request (spec defined)
HTTP Response with requested resource

Figure 28: SSO Operation – Unauthenticated HTTP Requestor

11.6.2.1 Scenario Sequence

Figure 28 shows the following sequence of steps for the operation:

1 The HTTP Requestor sends a HTTP GET or POST request to a Registry interface such as the QueryManager or LifeCycleManager.

1.1 The Registry checks to see if it already has a security context established for the Subject associated with the request. It determines that there is no pre-existing security context.

1.2 In order to establish a security context, the Registry therefor initiates the <samlp:AuthnRequest> protocol with the IdentityProvider. The <AuthnRequest> is sent using HTTP Redirect via the User Agent (e.g. Web Browser) used by the HTTP Requestor.

1.2.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this requires communicating with the User Agent being used by the HTTP Requestor to get the credentials associated with the Subject and then using the credentials to authenticate that the IdentityProvider knows the Subject. In case of SSL/TLS based communication the credetials are acquired without any user intervention directly from the User Agent. The figure assumes that the IdentityProvider is able to authenticate the Subject.

1.2.2 The IdentityProvider sends a <sampl:Response> message containing a <saml:AuthenticationStatement> to the Registry using either HTTP POST or HTTP Artifact SAML Binding via the User Agent.

1.2.2.1 The Registry uses implementation specific means to establish a security context for the Subject authenticated by the IdentityProvider based upon the information contained about the Subject in the <samlp:Response> message. This may include creating an HTTP Session for the HTTP Requestor.

1.2.2.2 The Registry maps the information about the Subject in the <samlp:Response> message into a <rim:User> instance. This establishes the <rim:User>context for the security context.

1.2.2.3 The Registry then performs authorization decision based upon the original HTTP request and the <rim:User>. The figure assumes that authorization decision was to allow the request to be processed. The Registry processes the request and subsequently return the requested resource to the HTTP Requestor via the HTTP response.

11.6.3 SSO Operation – Authenticated HTTP Requestor

This is the case where the HTTP Requestor first authenticates with an IdentityProvider and then accesses the registry over HTTP via a User Agent such as a Web Browser.

Currently there are no standard means defined for carrying SAML Assertions resulting from the Registry Requestor authenticating with an IdentityProvider over HTTP protocol to a Service Provider such as the registry. A registry MAY support this scenario in an implementation specific manner. Typically, the Identity Provider will define any such implementation specific manner.

11.6.4 SSO Operation – Unuthenticated SOAP Requestor

This is the case where an unauthenticated Registry Requestor accesses the registry over SOAP.

Figure 29 shows the steps involved.

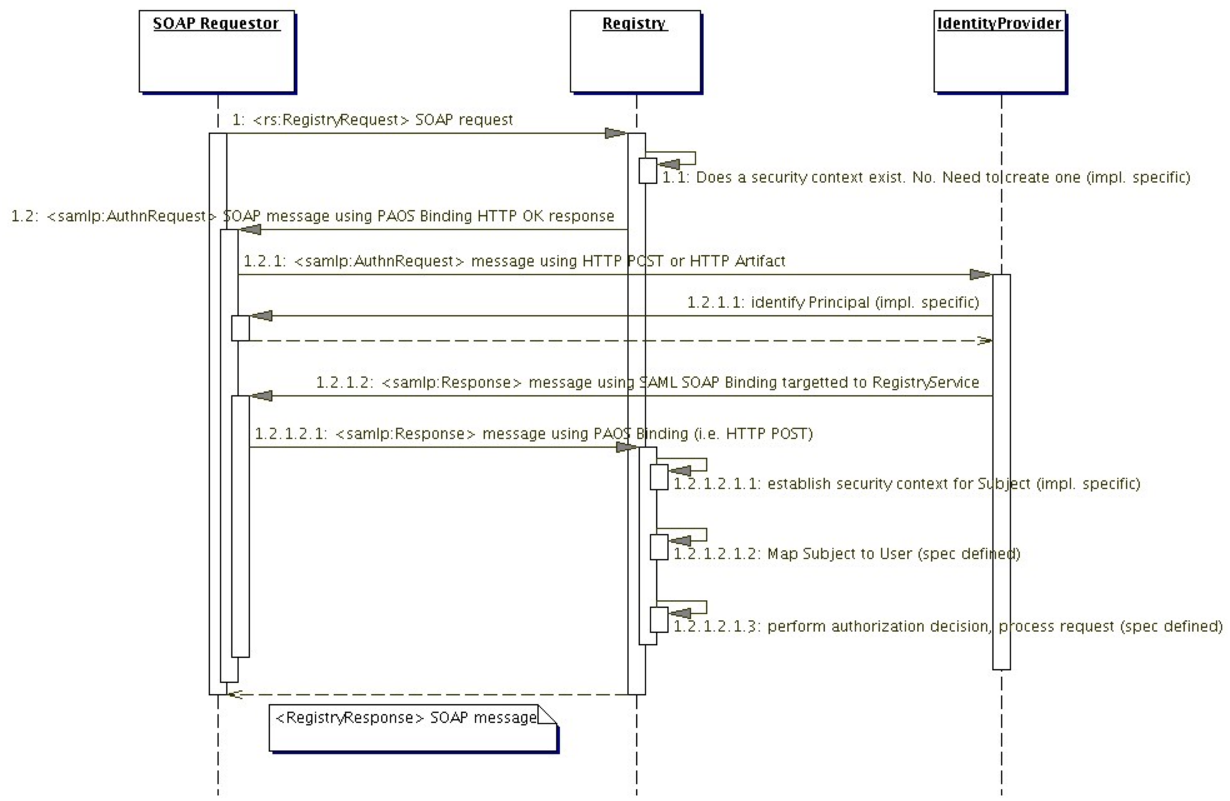

SOAP Requestor
Registry
IdentityProvider
1: <rs:RegistryRequest> SOAP request
1.1: Does a security context exist. No. Need to create one (impl. specific)
1.2: <samlp:AuthnRequest> SOAP message using PAOS Binding HTTP OK response
1.2.1: <samlp:AuthnRequest> message using HTTP POST or HTTP Artifact
1.2.1.1: identify Principal (impl. specific)
1.2.1.2: <samlp:Response> message using SAML SOAP Binding targetted to RegistryService
1.2.1.2.1: <samlp:Response> message using PAOS Binding (i.e. HTTP POST)
1.2.1.2.1.1: establish security context for Subject (impl. specific)
1.2.1.2.1.2: Map Subject to User (spec defined)
1.2.1.2.1.3: perform authorization decision, process request (spec defined)
<RegistryResponse> SOAP message

Figure 29: SSO Operation - Unauthenticated SOAP Requestor

11.6.4.1 Scenario Sequence

Figure 29 shows the following sequence of steps for the operation:

1 The SOAP Requestor sends a <rs:RegistryRequest> SOAP message such as a <lcm:SubmitObjectsRequest> to a Registry interface such as the LifeCycleManagerManager. In the request header the SOAP Requestor declares that it is an ECP requestor as defined by the ECP Profile in [SAMLProf].

1.1 The Registry checks to see if it already has a security context established for the Subject associated with the request. It determines that there is no pre-existing security context.

1.2 Because the request is from an ECP client, the registry uses the ECP Profile defined by [SAMLProf] and sends a <samlp:AuthnRequest> SOAP message as response to the <rs:RegistryRequest> SOAP message to the SOAP Requestor using the PAOS Binding as defined by [SAMLBind]. The response has an HTTP Response status of OK.

1.2.1 The SOAP Requestor then initiates the <samlp:AuthnRequest> protocol with the IdentityProvider. The <sampl:AuthnRequest> is sent using HTTP POST or Artifact Binding directly to the IdentityProvider.

1.2.1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this requires communicating with the SOAP Requestor to get the credentials associated with the Subject and then using the credentials to authenticate that the IdentityProvider knows the Subject. In case of SSL/TLS based communication the credetials are acquired without any user intervention directly from the SOAP Requestor. The figure assumes that the IdentityProvider is able to authenticate the Subject.

1.2.1.2 The IdentityProvider sends a <sampl:Response> message containing a <saml:AuthenticationStatement> to the SOAP Requestor using SAML SOAP Binding. The

HTTP header specifies the Registry as the ultimate target of the response.

1.2.1.2.1 The SOAP Requestor forwards the <sampl:Response> message containing a <saml:AuthenticationStatement> to the Registry using PAOS Binding via HTTP POST.

1.2.1.2.1.1 The Registry uses implementation specific means to establish a security context for the Subject authenticated by the IdentityProvider based upon the information contained about the Subject in the <samlp:Response> message. This may include creating an HTTP Session for the HTTP Requestor.

1.2.1.2.1.2 The Registry maps the information about the Subject in the <samlp:Response> message into a <rim:User> instance. This establishes the <rim:User>context for the security context.

1.2.1.2.1.3 The Registry then performs authorization decision based upon the original SOAP request and the <rim:User>. The figure assumes that authorization decision was to allow the request to be processed. The Registry processes the request and subsequently return a <rs:RegistryResponse> SOAP message as response to the original <rs:RegistryRequest> SOAP request.

11.6.5 SSO Operation – Authenticated SOAP Requestor

This is the case where the Registry Requestor first authenticates with an IdentityProvider directly and then makes a request to the registry using SOAP.

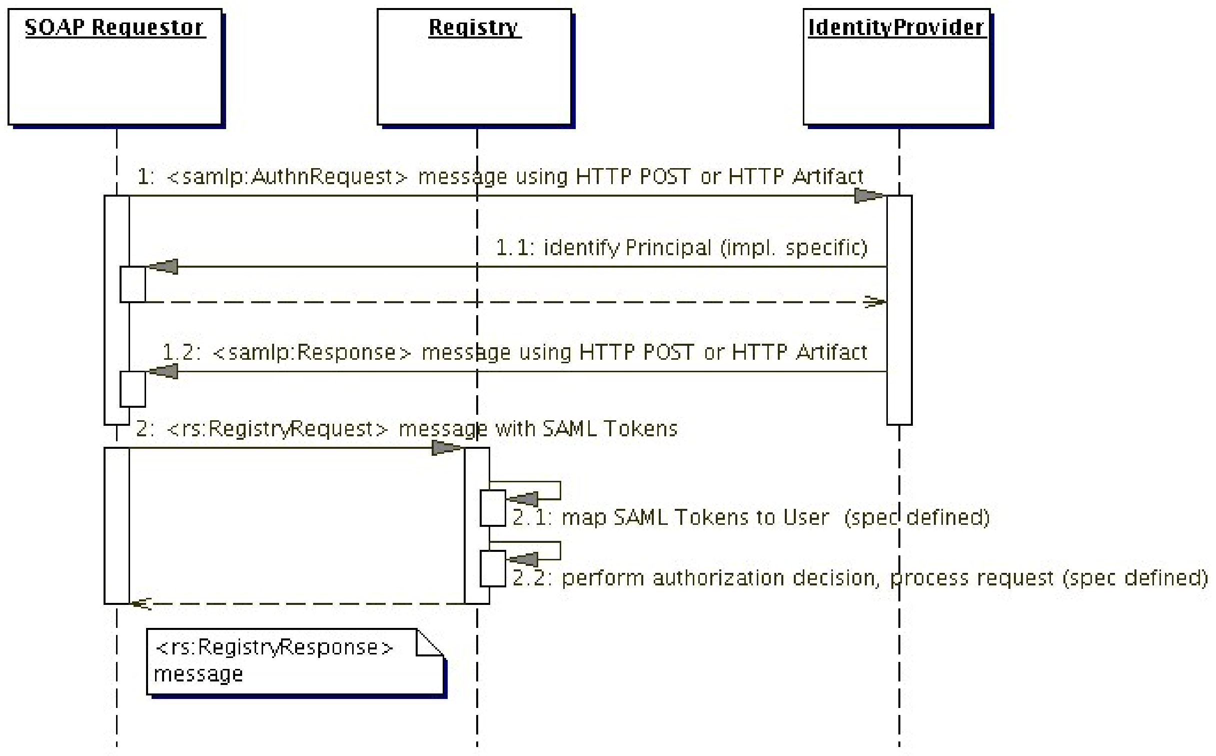

SOAP Requestor
Registry
IdentityProvider
1: <samlp:AuthnRequest> message using HTTP POST or HTTP Artifact
1.1: identify Principal (impl. specific)
1.2: <samlp:Response> message using HTTP POST or HTTP Artifact
2: <rs:RegistryRequest> message with SAML Tokens
2.1: map SAML Tokens to User (spec defined)
2.2: perform authorization decision, process request (spec defined)
<rs:RegistryResponse> message

Figure 30: SSO Operation - Authenticated SOAP Requestor

11.6.5.1 Scenario Sequence

The figure shows the following sequence of steps for the operation:

1 The SOAP Requestor then initiates the <samlp:AuthnRequest> protocol directly with the IdentityProvider. The <sampl:AuthnRequest> is sent using HTTP POST or Artifact Binding.

1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this requires communicating with the SOAP Requestor to get the credentials associated with the Subject and then using the credentials to authenticate that the IdentityProvider knows the Subject. In case of SSL/TLS based communication the credetials are acquired without any user intervention directly from the SOAP Requestor. The figure assumes that the IdentityProvider is able to authenticate the Subject.

1.2 The IdentityProvider sends a <sampl:Response> message containing a <saml:AuthenticationStatement> to the SOAP Requestor using SAML HTTP POST or HTTP Artifact Binding.

2 The SOAP Requestor sends a <rs:RegistryRequest> SOAP message such as a <lcm:SubmitObjectsRequest> to a Registry interface such as the LifeCycleManagerManager. The

<rs:RegistryRequest> SOAP message includes SAML Tokens in the <soap:Header> of the SOAP message as defined by [WSS-SAML]. The SAML Tokens are based upon the <sampl:Response> during authentication.

2.1 The registry maps the SAML Tokens from the <soap:Header> of the <rs:RegistryRequest> to a <rim:User> instance. This establishes the <rim:User> context for the request.

2.2 The Registry then performs authorization decision based upon the original SOAP request and the <rim:User>. The figure assumes that authorization decision was to allow the request to be processed. The Registry processes the request and subsequently return a <rs:RegistryResponse> SOAP message as response to the original <rs:RegistryRequest> SOAP request.

11.6.6 <samlp:AuthnRequest> Generation Rules

The following rules MUST be observed when the registry or Registry Client issues a <samlp:AuthnRequest>:

- A registry MUST specify a NameIDPolicy within the <samlp:AuthRequest>
- The Format of the NameIDPolicy MUST be urn:oasis:names:tc:SAML:2.0:nameid-format:persistent as defined by section in [SAMLCore]. Note that it is the Persistent Identifier that maps to the id attribute of <rim:User>.

–

11.6.7 <samlp:Response> Processing Rules

This section describes how the registry processes the <sampl:Response> to a <sampl:AuthnRequest>:

<samlp:Response> Processing

- Response Processing: The registry MUST verify the <ds:Signature> for the <sampl:Response> if present.
- The registry MUST check the <samlp:Status> associated with <sampl:Response> for errors. If the <samlp:Status> has a top level <samlp:StatusCode> whose value is NOT `urn:oasis:names:tc:SAML:2.0:status:Success then the registry MUST throw an` AuthenticationException. The AuthenticationException message SHOULD include the information from the StatusCode, StatusMessage and StatusDetail from the <samlp:Status>.

<saml:Assertion> Processing

- The registry SHOULD check the <saml:Assertion> for Conditions and honour any standard Conditions defined by [SAMLCore] if any are specified.

<saml:AuthnStatement> Processing

- The registry MUST check the SessionNotOnOrAfter attribute of the <saml:AuthnStatement> for validity of the authenticated session.

<saml:Subject> Processing

- A registry MUST map the <saml:Subject> to a <rim:User> instance as described in 11.6.8.

11.6.8 Mapping Subject to User

As required by [SAMLCore] a <samlp:Response> to a <samlp:AuthnRequest> MUST contain a <saml:Subject> that identifies the Subject that was authenticated by the IdentityProvider. In addition it MUST contain a <sampl:AuthnStatement> which asserts that the IdentityProvider indeed authenticated the Subject.

The following table defines the mapping between a <saml:Subject> and a <rim:User>:

– Subject Attribute	– User Attribute	– Description
– NameID content	– id attribute	NameID Format MUST be “urn:oasis:names:tc:SAML:1.1:nameid-format:persistent”

Table 8: Mapping Subject to User

Note that any attribute of Subject not specified above SHOULD be ignored when mapping Subject to User. Note that any attribute of User not specified above MUST be left unspecified when mapping Subject to User.

11.7 External Users

The SAML Profile allows registry Users to be registered in an Identity Provider external to the registry. These are referred to as “External Users”. A registry dynamically creates such External Users by mapping a SAML Subject to a User instance dynamically.

The following are some restrictions on External User instances:

- External User instances are transient from the registry’s perspective and MUST not be stored within the registry as User instances
- A RegistryObject MUST not have a reference to an External User unless it is composed within that RegistryObject. Composed RegistryObjects such as Classification instances are allowed to reference their parent External User instance.
- Since External User instances are transient they MUST not match a registry Query.

12 Native Language Support (NLS)

This chapter describes the Native Languages Support (NLS) features of ebXML Registry.

12.1 Terminology

The following terms are used in NLS.

NLS Term	Description
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.
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.
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].

12.2 NLS and Registry Protcol Messages

For the accurate processing of data in both registry client and registry services, it is essential for the recipient of a protocol message to know the character set being used by it.

A Registry Client SHOULD specify charset parameter in MIME header when they specify text/xml as Content-Type. A registry MUST specify charset parameter in MIME header when they specify text/xml as Content-Type.

The following is an example of specifying the character set in the MIME header.

```
Content-Type: text/xml; charset=ISO-2022-JP
```

If a registry receives a protocol message with the charset parameter omitted then it MUST use the default charset value of "us-ascii" as defined in [RFC 3023].

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 used, then usage of charset MUST follow [RFC 3023].

12.3 NLS Support in RegistryObjects

The information model XML Schema [RR-RIM-XSD] defines the <rim:InternationalStringType> for defining elements that contains a locale senstive string value.

```
  <complexType name="InternationalStringType">
    <sequence maxOccurs="unbounded" minOccurs="0">
```

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

An InternationalStringType may contain zero or more LocalizedStrings within it where each LocalizedString contain a string value is a specified local language and character set.

```
<complexType name="LocalizedStringType">
  <attribute ref="xml:lang" default="en-US"/>
  <attribute default="UTF-8" name="charset"/>
  <attribute name="value" type="tns:FreeFormText" use="required"/>
</complexType>
```

Examples of such attributes are the “name” and “description” attributes of the RegistryObject class defined by [ebRIM] as shown below.

```
  <complexType name="InternationalStringType">
    <sequence maxOccurs="unbounded" minOccurs="0">
      <element ref="tns:LocalizedString"/>
    </sequence>
  </complexType>
  <element name="InternationalString"
type="tns:InternationalStringType"/>
  <element name="Name" type="tns:InternationalStringType"/>
  <element name="Description" type="tns:InternationalStringType"/>

  <complexType name="LocalizedStringType">
    <attribute ref="xml:lang" default="en-US"/>
    <!--attribute name = "lang" default = "en-US" form = "qualified"
type = "language"/-->
    <attribute default="UTF-8" name="charset"/>
    <attribute name="value" type="tns:FreeFormText" use="required"/>
  </complexType>
  <element name="LocalizedString" type="tns:LocalizedStringType"/>
```

An element InternationalString is capable of supporting multiple locales within its collection of LocalizedStrings.

The above schema allows a single RegistryObject instance to include values for any NLS sensitive element in multiple locales.

The following example illustrates how a single RegistryObject can contain NLS sesnitive <rim:Name> and “<rim:Description> elements with their value specified in multiple locales. Note that the <rim:Name> and <rim:Description> use the <rim:InternationalStringType> as their type.

```
    <rim:ExtrinsicObject id="${ID}"  mimeType="text/xml">
      <rim:Name>
        <rim:LocalizedString xml:lang="en-US" value="customACP1.xml"/>
        <rim:LocalizedString xml:lang="fi-FI" value="customACP1.xml"/>
        <rim:LocalizedString xml:lang="pt-BR" value="customACP1.xml"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString xml:lang="en-US" value="A sample custom
ACP"/>
        <rim:LocalizedString xml:lang="fi-FI" value="Esimerkki custom
ACP"/>
        <rim:LocalizedString xml:lang="pt-BR" value="Exemplo de ACP
customizado
"/>
      </rim:Description>
```

```
        </rim:ExtrinsicObject>
```

Since locale information is specified at the sub-element level there is no language or character set associated with a specific RegistryObject instance.

12.3.1 Character Set of *LocalizedString*

The character set used by a locale specific String (LocalizedString) is defined by the charset attribute. Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of LocalizedStrings for maximum interoperability.

12.3.2 Language of *LocalizedString*

The language MAY be specified in xml:lang attribute (Section 2.12 [REC-XML]).

12.4 NLS and Repository Items

While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is always associated with a single repository item. The repository item MAY be in a single locale or MAY be in multiple locales. This specification does not specify any NLS requirements for repository items.

12.4.1 Character Set of Repository Items

When a submitter submits a repository item, they MAY specify the character set used by the respository item using the MIME *Content-Type* mime header for the mime multipart containing the repository item as shown below:

```

        Content-Type: text/xml; charset="UTF-8"

```

Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of LocalizedStrings for maximum interoperability. A registry MUST preserve the charset of a repository item as it is originally specified when it is submitted to the registry.

12.4.2 Language of Repository Items

The Content-language mime header for the mime bodypart containing the repository item MAY specify the language for a locale specific repository item. The value of the Content-language mime header property MUST conform to [RFC 1766].

This document currently specifies only the method of sending the information of character set and language, and how it is stored in a registry. However, the language information MAY be used as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation procedure, like registry client is asking a favorite language for messages from registry services, could be another functionality for the future revision of this document.

13 Conformance

This chapter defines the technical conformance requirements for ebXML Registry. Note that it does not define specific conformance tests to verify compliance with various conformance profiles.

13.1 Conformance Profiles

An ebXML Registry MUST comply with one of the following conformance profiles:

- Registry Lite – This conformance profile requires the regsitry to implement a minimal set of core features defined by this specification.
- Registry Full – This conformance profile requires the registry to implement additional set of features in addition to those required by the Registry Lite conformance profile.

13.2 Feature Matrix

The following table identifies the implementation requirements for each feature defined by this specification for each conformance profile defined above.

Table 9: Feature Conformance Matrix

Feature	Registry Lite	Registry Full
SOAP Binding		
QueryManager binding	MUST	MUST
LifeCycleManager binding	MUST	MUST
HTTP Binding		
RPC Encoded URL	MUST	MUST
User Defined URL	MAY	MUST
File Path URL	MAY	MUST
LifeCycleManager		
SubmitObjects Protocol	MUST	MUST
UpdateObjects Protocol	MUST	MUST
ApproveObjects Protocol	MUST	MUST
DeprecateObjects Protocol	MUST	MUST
UnderprecateObjects Protocol	MUST	MUST
RemoveObjects Protocol	MUST	MUST
Registry Managed Version Control	MAY	MUST
QueryManager		
SQL Query	MAY	MUST
Filter Query	MUST	MUST
Stored Parameterized Query	MAY	MUST
Iterative Query	MAY	MUST
Event Notification	MAY	MUST
Content Management Services		
Validate Content Protocol	MAY	MUST
Catalog Content Protocol	MAY	MUST
Canonical XML Cataloging Service	MAY	MUST
Cooperating Registries		
Remote object references	MAY	MUST
Federated queries	MAY	MUST
Object Replication	MAY	MUST
Object Relocation	MAY	MUST
Registry Security		
Identity Management	MUST	MUST
Message Security		
Transport layer security	MAY	MUST
SOAP Message Security	MUST	MUST
Repository Item Security	MUST	MUST
Authorization and Access Control		
Default Access Control Policy	MUST	MUST
Custom Access Control Policies	MAY	MUST
Audit Trail	MUST	MUST

Feature	Registry Lite	Registry Full
Registry SAML Profile	MAY	MUST
NLS	MUST	MUST

4339

14 References

14.1 Normative References

[RFC2119]	S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.
[ebRIM]	ebXML Registry Information Model Version 3.0 http://www.oasis-open.org/committees/regrep/documents/3.0/specs/regrep-rim-3.0-cs-01.pdf
[REC-XML]	W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition) http://www.w3.org/TR/REC-xml
[RFC 1766]	IETF (Internet Engineering Task Force). RFC 1766: Tags for the Identification of Languages, ed. H. Alvestrand. 1995. http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html
[RFC 2130]	IETF (Internet Engineering Task Force). RFC 2130 The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996 http://www.faqs.org/rfcs/rfc2130.html
[RFC 2277]	IETF (Internet Engineering Task Force). RFC 2277: IETF policy on character sets and languages, ed. H. Alvestrand. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html
[RFC 2278]	IETF (Internet Engineering Task Force). RFC 2278: IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html
[RFC2616]	IETF (Internet Engineering Task Force). RFC 2616: Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1* . 1999. http://www.w3.org/Protocols/rfc2616/rfc2616.html
[RFC2965]	IETF (Internet Engineering Task Force). RFC 2965: D. Kristol et al. *HTTP State Management Mechanism*. 2000. http://www.w3.org/Protocols/rfc2616/rfc2616.html
[RR-CMS-XSD]	ebXML Registry Content Management Services XML Schema http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd
[RR-LCM-XSD]	ebXML Registry LifeCycleManager XML Schema http://www.oasis-open.org/committees/regrep/documents/3.0/schema/lcm.xsd
[RR-RIM-XSD]	ebXML Registry Information Model XML Schema http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd
[RR-RS-XSD]	ebXML Registry Service Protocol XML Schema http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rs.xsd
[RR-QM-XSD]	ebXML Registry QueryManager XML Schema http://www.oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd
[SAMLBind]	S. Cantor et al., *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-bindings-2.0-cd-03. http://www.oasis-open.org/committees/security/. Note: when this document is finalized, this URL will be updated.
[SAMLConform]	P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0.* OASIS SSTC, September 2004. Document ID sstc-saml-conformance-2.0-cd-03. http://www.oasis-open.org/committees/security/.

Note: when this document is finalized, this URL will be updated.

[SAMLCore] *S. Cantor et al., Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. OASIS SSTC, December 2004. Document ID sstc-saml-core-2.0-cd-03.* *http://www.oasis-open.org/committees/security/.* Note: when this document is finalized, this URL will be updated.

[SAMLProf] S. Cantor et al., *Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-profiles-2.0-cd-03.
http://www.oasis-open.org/committees/security/.
Note: when this document is finalized, this URL will be updated.

[SAMLP-XSD] S. Cantor et al., SAML protocols schema. OASIS SSTC, September 2004. Document ID sstc-saml-schema-protocol-2.0.
http://www.oasis-open.org/committees/security/.
Note: when this document is finalized, this URL will be updated.

[SAML-XSD] S. Cantor et al., SAML assertions schema. OASIS SSTC, September 2004. Document ID sstc-saml-schema-assertion-2.0.
http://www.oasis-open.org/committees/security/.
Note: when this document is finalized, this URL will be updated.

[SOAP11] W3C Note. Simple Object Access Protocol, May 2000 http://www.w3.org/TR/SOAP

[SwA] W3C Note: SOAP with Attachments, Dec 2000
http://www.w3.org/TR/SOAP-attachments

[SQL] Structured Query Language (FIPS PUB 127-2)
http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
(SQL/PSM) [ISO/IEC 9075-4:1996]

[UUID] DCE 128 bit Universal Unique Identifier
http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20

[WSDL] W3C Note. Web Services Description Language (WSDL) 1.1
http://www.w3.org/TR/wsdl

[XML] T. Bray, et al. Extensible Markup Language (XML) 1.0 (Second Edition). World Wide Web Consortium, October 2000.
http://www.w3.org/TR/REC-xml

[XMLDSIG] XML-Signature Syntax and Processing
http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

[WSI-BSP] WS-I: Basic Security Profile 1.0
http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html
Note: when this document is finalized, this URL will be updated.

[WSS-SMS] Web Services Security: SOAP Message Security 1.0
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[WSS-SWA] Web Services Security: SOAP Message with Attachments (SwA) Profile 1.0
http://www.oasis-open.org/apps/org/workgroup/wss/download.php/10902/wss-swa-profile-1.0-cd-01.pdf
Note: when this document is finalized, this URL will be updated.

14.2 Informative

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

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

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

[DeltaV] Versioning Extension to WebDAV, IETF RFC 3253
http://www.webdav.org/deltav/protocol/rfc3253.html

[XPT] XML Path Language (XPath) Version 1.0
http://www.w3.org/TR/xpath

[IANA] IANA (Internet Assigned Numbers Authority).
Official Names for Character Sets, ed. Keld Simonsen et al.
http://www.iana.org/

[RFC2392] **E. Levinson, Content-ID and Message-ID Uniform Resource Locators, IETF RFC 2392,**
http://www.ietf.org/rfc/rfc2392.txt

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

[RFC 3023] IETF (Internet Engineering Task Force). RFC 3023:
XML Media Types, ed. M. Murata. 2001.
ftp://ftp.isi.edu/in-notes/rfc3023.txt

[SAMLMeta] S. Cantor et al., *Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-metadata-2.0-cd-02.
http://www.oasis-open.org/committees/security/.

[SAMLGloss] J. Hodges et al., *Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-glossary-2.0-cd-02.
http://www.oasis-open.org/committees/security/.

[**SAMLSecure]** F. Hirsch et al., *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-sec-consider-2.0-cd-02.
http://www.oasis-open.org/committees/security/.

[SAMLTech] J.Hughes et al.,Technical Overview of the OASIS Security
Assertion Markup Language (SAML)V2.0.
http://www.oasis-open.org/committees/download.php/7874/sstc-saml-tech-overview-2.0-draft-01.pdf

[UML] Unified Modeling Language
http://www.uml.org
http://www.omg.org/cgi-bin/doc?formal/03-03-01

A. Acknowledgments

The editors would like to acknowledge the contributions of the OASIS ebXML Registry Technical Committee, whose voting members at the time of publication are listed as contributors on the title page of this document.

- Finally, the editors wish to acknowledge the following people for their contributions of material used as input to the OASIS ebXML Registry specifications:

Name	Affiliation
Aziz Abouelfoutouh	Government of Canada
Ed Buchinski	Government of Canada
Asuman Dogac	Middle East Technical University, Ankara Turkey
Michael Kass	NIST
Richard Lessard	Government of Canada
Evan Wallace	NIST
David Webber	Individual

-

B. Notices