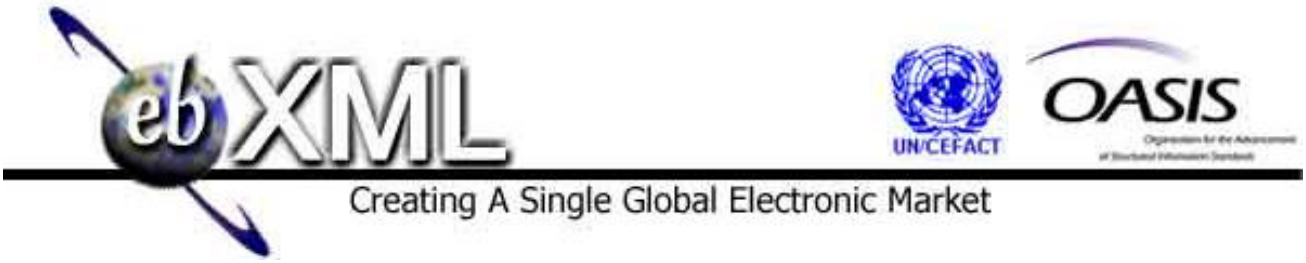

ebXML Registry Services

ebXML Registry Project Team

Working Draft 1/20/2001

This version: Version 0.84

1 Status of this Document

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

Distribution of this document is unlimited.

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

This version:

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.84.pdf

***Latest version*:**

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.84.pdf

Previous version:

http://www.ebxml.org/project_teams/registry/private/RegistryServicesSpecificationv0.83.pdf

2 ebXML participants

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

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

Table of Contents

Table of Figures

Table of Tables

3 Introduction

3.1 Summary of Contents of Document

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

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

3.2 General Conventions

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

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

3.3 Audience

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

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

3.4 Related Documents

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

a) *ebXML Registry Business Domain Model* [BDM] - defines requirements for ebXML Registry Services

b) *ebXML Registry Information Model [RIM]*- specifies the information model for the ebXML Registry

c) *ebXML Messaging Service Specification* [MS]

d) *ebXML Business Process Specification Schema* [BPM]

e) *Collaboration Protocol Specification* [CPA] (under development) - defines how profiles can be defined for a party and how two parties' profiles may be used to define a party agreement

4 Design Objectives

4.1 Goals

The goals of this version of the specification are to:

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

4.2 Caveats and Assumptions

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

It is assumed that:

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

5 System Overview

5.1 What The ebXML Registry Does

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

5.2 How The ebXML Registry Works

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

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

5.3 Schema Documents Are Submitted

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

5.4 Business Process Documents Are Submitted

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

5.5 Seller's Collaboration Protocol Profile Is Submitted

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

5.6 Buyer Discovers The Seller

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

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

5.7 CPA Is Established

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

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

5.8 Where the Registry Services May Be Implemented

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

6 Registry Architecture

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

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

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

6.1 Implicit CPA Between Clients And Registry

ebXML defines that a Collaboration Protocol Agreement [CPA] must exist between two parties in order for them to engage in B2B interactions.

Similarly, this specification defines a CPA between a Registry client and the Registry. Typical B2B interactions in ebXML require an explicit CPA to be negotiated between parties. However, the CPA between clients and the Registry is an implicit CPA that describes the interfaces that the Registry and the client expose to each other for Registry specific interactions. These interfaces are described in Figure 1 and subsequent sections.

6.2 Client To Registry Communication Bootstrapping

Because there is no previously established CPA between the Registry and the RegistryClient, the client must know at least one Transport specific communication address for the Registry. This communication address is typically a URL to Registry, although it could be some other type of address such as email address.

For example, if the communication used by the Registry is HTTP then the communication address is a URL. In this example, the client uses the Registry's public URL to create an implicit CPA with the Registry. When the client sends a request to the Registry, it provides a URL to itself. The Registry uses the client's URL to form its version of an implicit CPA with the client. At this point a session is established within the Registry.

For the duration of the client's session with the Registry, messages may be exchanged bidirectionally as required by the interaction protocols defined in this specification.

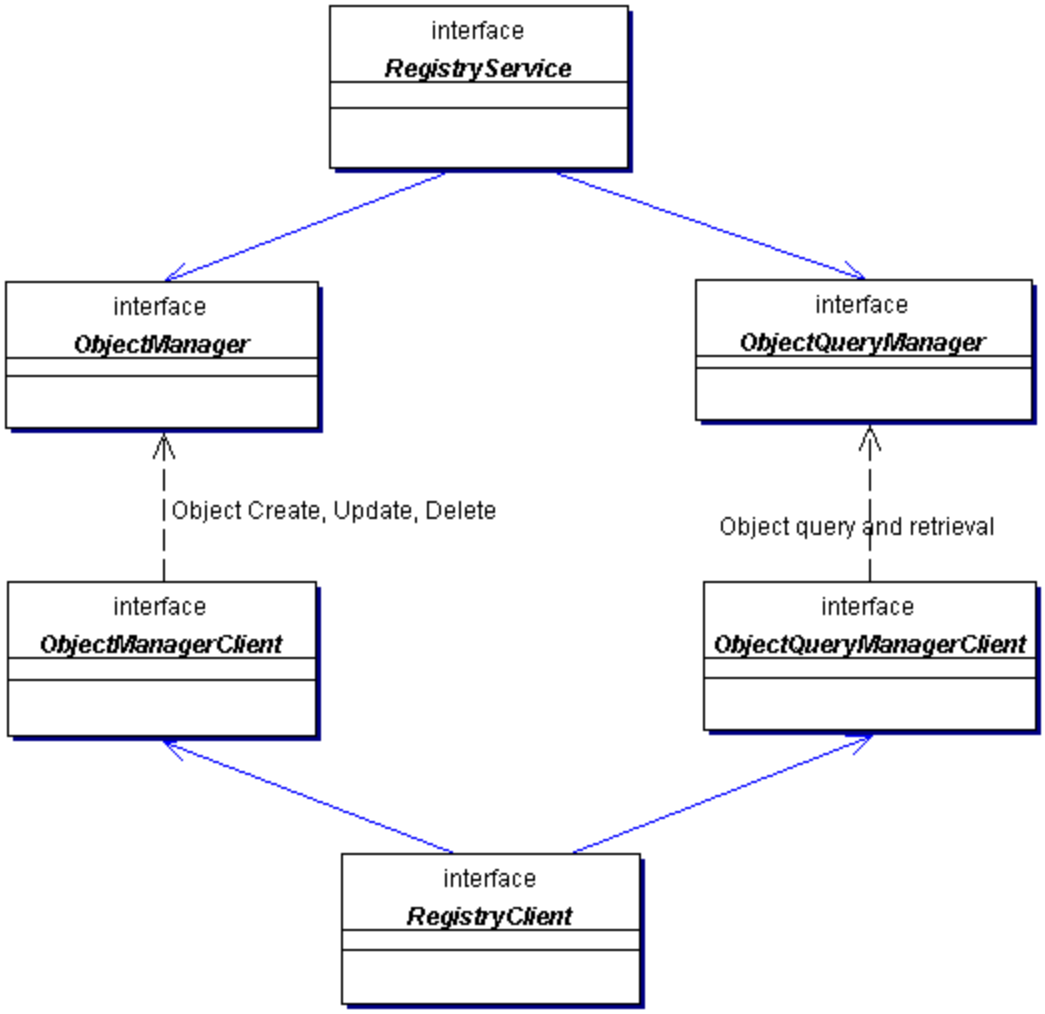

Figure 1: ebXML Registry Interfaces

6.3 Interfaces Exposed By The Registry

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

6.3.1 Interface *RegistryService*

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

Method Summary	
ObjectManager	**getObjectManager**() Returns the ObjectManager interface implemented by the Registry service.
ObjectQueryManager	**getObjectQueryManager**() Returns the ObjectQueryManager interface implemented by the Registry service.

352

353 6.3.2 Interface *ObjectManager*

354

355 This is the interface exposed by the Registry Service that implements the Object
356 life cycle management functionality of the Registry. Its methods are invoked by
357 the Registry Client. For example, the client may use this interface to submit
358 objects, classify and associate objects and to deprecate and remove objects.

359

Method Summary	
Void	**approveObjects**(ApproveObjectsRequest req) Approves one or more previously submitted objects.
Void	**deprecateObjects**(DeprecateObjectsRequest req) Deprecates one or more previously submitted objects.
Void	**removeObjects**(RemoveObjectsRequest req) Removes one or more previously submitted objects from the Registry.
void	**submitObjects**(SubmitObjectsRequest req) Submits one or more objects and possibly metadata related to object such as Associations and Classifications.

360 6.3.3 Interface *ObjectQueryManager*

361

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

366

Method Summary	
GetClassificationTreeResponse	**getClassificationTree**(GetClassificationTreeRequest req) Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.
void	**getClassificationTreeAsync**(GetClassificationTreeRequest req) Asynchronous version of getClassificationTree.
GetClassifiedObjectsResponse	**getClassifiedObjects**(GetClassifiedObjectsRequest req) Returns a collection of references to ManagedObjects classified under specified ClassificationItem.
void	**getClassifiedObjectsAsync**(GetClassifiedObjectsRequest req) Asynchronous version of getClassifiedObjects.
GetContentResponse	**getContent**() Returns the specified content. The response includes all the content specified in the request as additional payloads within the response message.
void	**getContentAsync**() Async version of getContent.
GetRootClassificationNodesResponse	**getRootClassificationNodes**(GetRootClassificationNodesRequest req) Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.
void	**getRootClassificationNodesAsync**(GetRootClassificationNodesRequest req) Async version of getRootClassificationNodes.
AdhocQueryResponse	**submitAdhocQuery**(AdhocQueryRequest req) Submit an ad hoc query request.
void	**submitAdhocQueryAsync**(AdhocQueryRequest req) Async version of submitAdhocQuery.

6.4 Interfaces Exposed By Registry Clients

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

6.4.1 Interface *RegistryClient*

This is the principal interface implemented by a Registry client. The client provides this interface when creating a connection to the Registry. It provides the methods that are used by the Registry to discover service specific interfaces implemented by the client.

Method Summary	
ObjectManagerClient	**getObjectManagerClient**() Returns the ObjectManagerClient interface implemented by the client.
ObjectQueryManagerClient	**getObjectQueryManagerClient**() Returns the ObjectQueryManagerClient interface implemented by the client.

6.4.2 Interface *ObjectManagerClient*

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

Method Summary	
void	**approveObjectsAccepted**(RequestAcceptedResponse resp) Notifies client that a previously submitted ApproveObjectsRequest was accepted by the Registry.
void	**approveObjectsError**(ebXMLError error) Notifies client that a previously submitted ApproveObjectsRequest was not accepted by the Registry due to an error.

void	**deprecateObjectsAccepted**(RequestAcceptedResponse resp) Notifies client that a previously submitted DeprecateObjectsRequest was accepted by the Registry.
void	**deprecateObjectsError**(ebXMLError error) Notifies client that a previously submitted DeprecateObjectsRequest was not accepted by the Registry due to an error.
void	**removeObjectsAccepted**(RequestAcceptedResponse resp) Notifies client that a previously submitted RemoveObjectsRequest was accepted by the Registry.
void	**removeObjectsError**(ebXMLError error) Notifies client that a previously submitted RemoveObjectsRequest was not accepted by the Registry due to an error.
void	**submitObjectsAccepted**(RequestAcceptedResponse resp) Notifies client that a previously submitted SubmitObjectsRequest was accepted by the Registry.
void	**submitObjectsError**(ebXMLError error) Notifies client that a previously submitted SubmitObjectsRequest was not accepted by the Registry due to an error.

6.4.3 Interface *ObjectQueryManagerClient*

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

Method Summary	
void	**getClassificationTreeAsyncResponse**(GetClassificationTreeResponse resp) Async response for getClassificationTreeAsync request.
void	**getClassifiedObjectsAsyncResponse**(GetClassifiedObjectsResponse resp) Async response for getClassifiedObjectsAsync request.

void	**getContentAsyncResponse**(GetContentResponse resp) Async response for getContent request.
void	**getRootClassificationNodesAsyncResponse**(GetRootClassificationNodesResponse resp) Async response for getRootClassificationNodesAsync request.
void	**submitAdhocQueryAsyncResponse**(AdhocQueryResponse resp) Async response for submitAdhocQueryAsync request.

7 Object Management Service

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

In the current version of this specification, any client may submit content as long as the content is digitally signed by an approved Certification Authority. Submitting Organizations do not have to register prior to submitting content.

7.1 Life Cycle of a Managed Object

The main purpose of the Object Management service is to manage the life cycle of managed object contents in the Registry.

Figure 2 shows the typical life cycle of a managed object content. Note that the current version of this specification does not support Object versioning. Object versioning will be added in a future version of this specification.

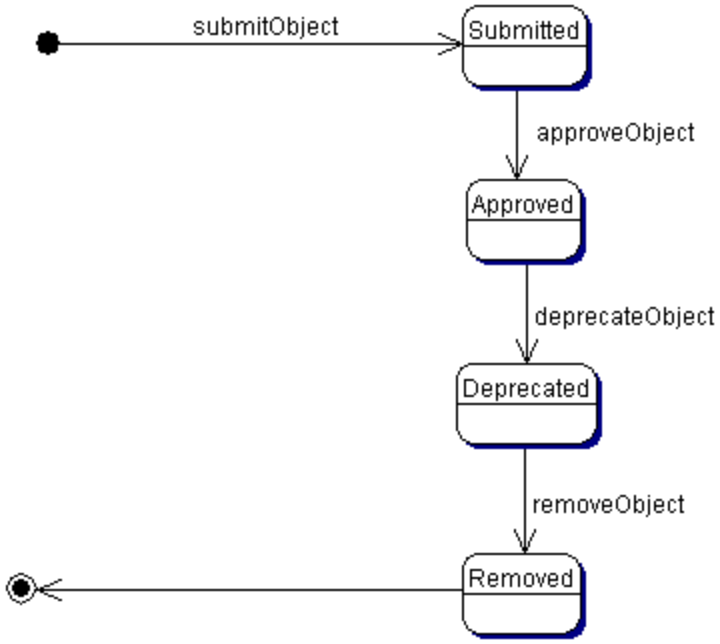

Figure 2: Life Cycle of a Managed Object

7.2 Object Attributes

A managed object content is associated with a set of standard metadata defined as attributes of the Object class and its sub-classes as described in [RIM]. These attributes reside outside of the actual managed object content and catalog descriptive information about the managed object content. XML DTD elements called ExtrinsicObject and IntrinsicObject (See Appendix A.2 for details.) are defined that encapsulates all object metadata attributes defined in [RIM] as attributes of the DTD elements.

7.3 The Submit Objects Protocol

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

Figure 3: Submit Objects Sequence Diagram

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

The SubmitObjectRequest message includes 1 or more SubmittedObject elements.

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

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

7.4 The Approve Objects Request

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

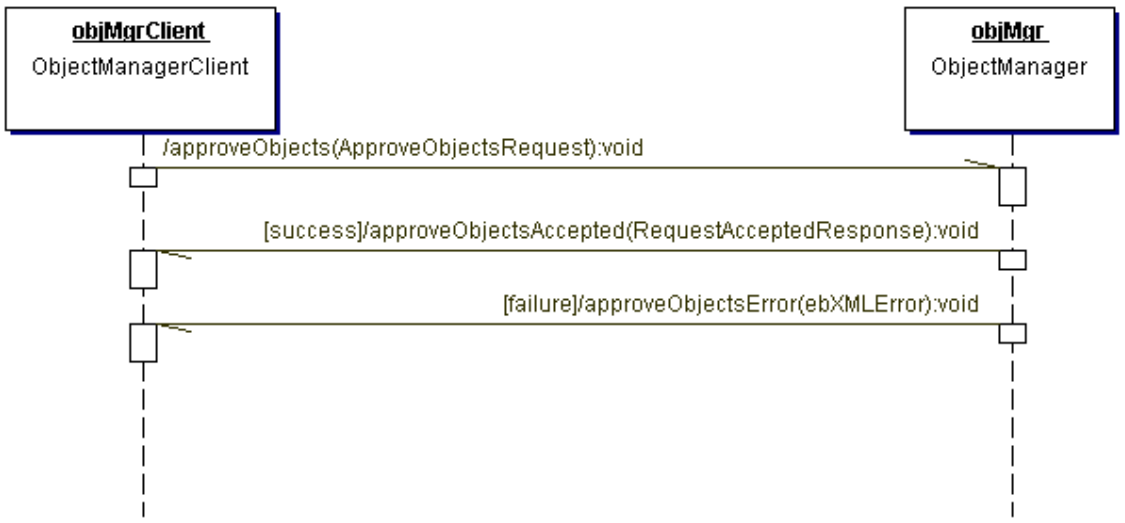

Figure 4: Approve Objects Sequence Diagram

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

7.5 The Deprecate Objects Request

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

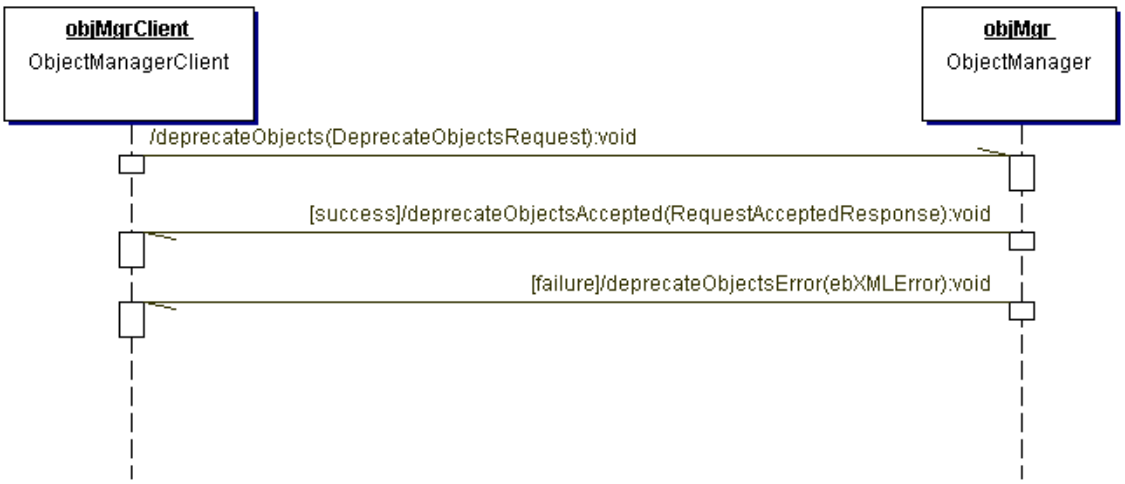

Figure 5: Deprecate Objects Sequence Diagram

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

7.6 The Remove Objects Request

This section describes the protocol of the Registry Service that allows a client to remove one or more previously deprecated managed object contents using the Object Manager.

Only if all references (e.g. Associations, Classifications, ExternalLinks) to an object have been removed, can that object then be removed using a RemoveObjectsRequest. Attempts to remove an object while it still has references results in an InvalidRequestError that is returned within an ebXMLError message sent to the ObjectManagerClient by the ObjectManager.

Once an object is removed it will be not be present at all in the Registry. The remove object protocol is expressed in UML notation as described in Appendix B.

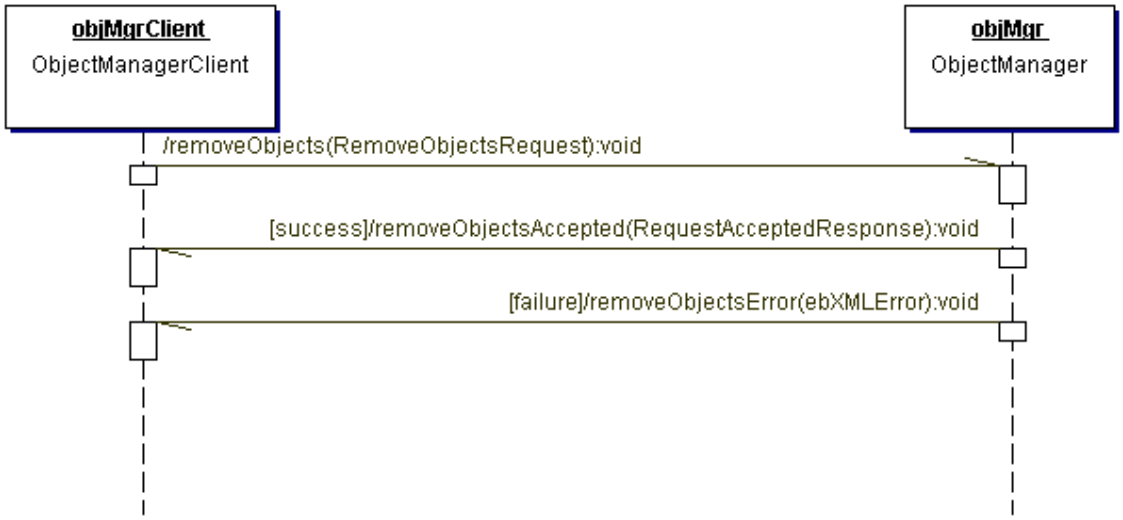

Figure 6: Remove Objects Sequence Diagram

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

8 Object Query Management Service

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

Any errors in the query request messages are indicated in the corresponding query response message. Note that for each query request/response there is both a synchronous and asynchronous version of the interaction.

8.1 Browse and Drill Down Query Support

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

8.1.1 Get Root Classification Nodes Request

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

Figure 7: Get Root Classification Nodes Sequence Diagram

Figure 8: Get Root Classification Nodes Asynchronous Sequence Diagram

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

8.1.2 Get Classification Tree Request

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

Figure 9: Get Classification Tree Sequence Diagram

Figure 10: Get Classification Tree Asynchronous Sequence Diagram

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

8.1.3 Get Classified Objects Request

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

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

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

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

8.1.3.1 Get Classified Objects Request Example

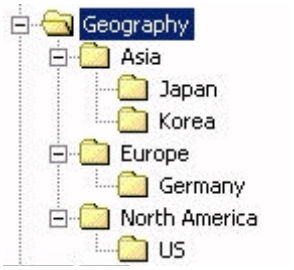

Figure 11: A Sample Geography Classification

Let us say a classification tree has the structure shown in Figure 11:

- If the Geography node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all ManagedObjects that are directly classified by Geography *or* North America *or* US *or* Asia *or* Japan *or* Korea *or* Europe *or* Germany.
- If the Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all ManagedObjects that are directly classified by Asia *or* Japan *or* Korea.
- If the Japan *and* Korea nodes are specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all ManagedObjects that are directly classified by both Japan *and* Korea.
- If the North America *and* Asia node is specified in the GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should include all ManagedObjects that are directly classified by (North America *or* US) *and* (Asia *or* Japan *or* Korea).

8.2 Ad Hoc Query Support

The Registry supports an Ad hoc query capability that is designed for Registry clients that demand more complex query capability. The ad hoc query interface allows a client to submit complex queries using a declarative query language.

8.2.1 Query Language Syntax

[Note] The query syntax may evolve in a future version of this document due to a lack of consensus within the Registry team on the choice of query syntax.

The ad hoc query language syntax of the Registry is defined by a stylized use of a proper subset of the “SELECT” statement of SQL-92 query language as defined by [SQL]. The exact syntax of the Registry query language is defined by the BNF grammar in Appendix C.

Note that the use of a subset of SQL syntax for ad hoc queries does not imply a requirement to use relational databases in a Registry implementation. Its purpose is to provide a canonical syntax for declaratively defining a query on metadata in the Registry, based on classes and attributes defined by [RIM].

In a future version of this specification, the W3C XML Query Language may be considered as an alternate query syntax when it reaches the recommendation stage.

8.2.2 Query Syntax Binding To [RIM]

Registry queries are defined based upon the query syntax in in Appendix C and a fixed logical schema defined by [RIM]. The following section define this binding.

8.2.2.1 Interface and Class Binding

Interface and class names in [RIM] map to table references in the query syntax. Interface and class names may be used in the same way as table names in SQL.

8.2.2.2 Accessor Method To Attribute Binding

Most of the [RIM] interfaces methods are simple get methods that map directly to attributes. For example the getName method on Object maps to a *name* attribute of type String.

8.2.2.3 Primitive Attributes Binding

Attributes defined by [RIM] that are of primitive types (e.g. String) may be used in the same way as column names in SQL.

8.2.2.4 Reference Attribute Binding

A few of the [RIM] interface methods return references to instances of interfaces or classes defined by [RIM]. For example, the getAccessControlPolicy method of the Object class returns a reference to an instance of an AccessControlPolicy object.

In such cases the reference maps to the ID attribute for the referenced object. This is a special case of a primitive attribute mapping.

8.2.2.5 Collection Attribute Binding

A few of the [RIM] interface methods return Collections of references to instances of interfaces or classes defined by [RIM]. For example, the getPackages method of the ManagedObject class returns a Collection of references to instances of Packages that the object is a member of. The use of Collection attributes are restricted to be only within the IN clause of the query grammar.

The SQL IN clause may be used to test for membership of an object in such collections of references.

8.2.2.6 Semantic Constraints On Query Syntax

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

1. Class names and attribute names must be processed in a case insensitive manner.
2. Collection attributes must only be specified within an IN clause.

8.2.3 Simple Metadata Based Queries

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

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

```
SELECT DISTINCT obj FROM ExtrinsicObject WHERE
        obj.name LIKE ‘%Acme%’ AND
        obj.majorVersion >= 1 AND
        (obj.majorVersion >= 2 OR  obj.minorVersion > 3);
```

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

8.2.4 Classification Queries

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

8.2.4.1 Identifying ClassificationNodes

Like all objects in [RIM], ClassificationNodes are identified by their ID. However, they may also be identified as a path attribute that specifies an absolute path from a root classification node to the specified classification node where each path element is the name attribute of a ClassificationNode and is separated by '.' as a delimiter.

8.2.4.2 Getting Root Classification Nodes

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

```
SELECT FROM ClassificationNode WHERE parent IS NULL
```

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

8.2.4.3 Getting Children of Specified ClassificationNode

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

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

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

8.2.4.4 Getting Objects Classified By a ClassificationNode

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

```
SELECT DISTINCT eo
FROM ExtrinsicObject eo, ClassificationNode auto, ClassificationNode geo
WHERE
        (geo IN (eo.classificationNodes) AND geo.path = 'Geography.Asia.Japan')
AND
        (auto IN (eo.classificationNodes) AND auto.path = 'Industry.Automotive')
```

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

8.2.4.5 Getting ClassificationNodes That Classify an Object

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

```
SELECT cn FROM ClassificationNode cn, ExtrinsicObject o WHERE
        o.ID = <id> AND
        cn IN (o.classificationNodes)
```

8.2.5 Association Queries

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

8.2.5.1 Getting All Association With Specified Object As Its Source

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

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

8.2.5.2 Getting All Association With Specified Object As Its Target

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

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

8.2.5.3 Getting Associated Objects Based On Association Attributes

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

Select Associations that have the specified name.

```
SELECT assoc FROM Association WHERE
        assoc.name = <name>
```

Select Associations that have the specified source role name.

```
SELECT assoc FROM Association WHERE
        assoc.sourceRole = <roleName>
```

Select Associations that have the specified target role name.

```
SELECT assoc FROM Association WHERE
        assoc.targetRole = <roleName>
```

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

```
SELECT DISTINCT assoc FROM Association WHERE
        assoc.associationType = <associationType>
```

8.2.5.4 Complex Association Queries

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

```
SELECT DISTINCT assoc FROM Association WHERE
        Assoc.sourceObject = <id> AND
        assoc.sourceRole = ‘buysFrom’ AND
        assoc.sourceRole = ‘sellsTo’
```

8.2.6 Package Queries

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

```
SELECT p FROM Package p, ExtrinsicObject obj WHERE
        obj.ID = <id> AND p IN (obj.packages)
```

To find all Association objects in a specified package, the following query is specified:

```
SELECT a FROM Association, Package p WHERE
        p.ID = <id> AND a IN (p.memberObjects)
```

8.2.6.1 Complex Package Queries

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

```
SELECT p FROM Package p, ExtrinsicObject obj WHERE
        obj.ID = <id> AND p IN (obj.packages) AND
        p.name LIKE ‘%RosettaNet%’ AND
        p.status != ‘DEPRECATED’
```

8.2.7 ExternalLink Queries

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

```
SELECT l FROM ExternalLink, ExtrinsicObject obj WHERE
        obj.ID = <id> AND l IN (obj.externalLinks)
```

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

```
SELECT obj FROM ExtrinsicObject, ExternalLink l WHERE
        l.ID = <id> AND obj IN (l.linkedObjects)
```

8.2.7.1 Complex ExternalLink Queries

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

```
SELECT l FROM ExternalLink, ExtrinsicObject obj WHERE
        obj.ID = <id> AND l IN (obj.externalLinks) AND
        l.description LIKE '%legal%' AND
        l.externalURI LIKE '%http://%'
```

8.2.8 Audit Trail Queries

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

```
SELECT ev FROM AuditableEvent, ExtrinsicObject obj WHERE
        obj.ID = <id> AND ev IN (obj.auditTrail)
```

8.2.9 Content Based Ad Hoc Queries

The ad hoc query interface of the Registry supports the ability to search for content based not only on metadata that catalogs the content but also the data contained within the content itself. For example it is possible for a client to submit a query that searches for all Collaboration Party Profiles that define a role named "seller" within a RoleName element in the CPP document itself.

Currently content-based query capability is restricted to XML content.

8.2.9.1 Automatic Classification of XML Content

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

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

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

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

```
        [Note] This approach is conceptually similar to the
             way databases support indexed retrieval. DBAs
```

```
            define indexes on tables in the schema. When
            data is added to the table, the data gets
            automatically indexed.
```

8.2.9.2 Index Definition

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

A SubmittedObject element for a schema or DTD may define a collection of ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList is ignored if the content being submitted is not of the SCHEMA objectType.

The ClassificationIndex element inherits the attributes of the base class Object in [RIM]. It then defines specialized attributes as follows:

1. classificationNode: This attribute references a specific ClassificationNode by its ID.
2. contentIdentifier: This attribute identifies a specific data element within the document instances of the schema using an XPATH path expression as defined by [XPT].

8.2.9.3 Example Of Index Definition

To define an index that automatically classifies a CPP based upon the roles defined within its RoleName elements, the following index must be defined on the CPP schema or DTD:

```
<ClassificationIndex
        classificationNode='id-for-role-classification-scheme'
        contentIdentifier='/Role//RoleName'
/>
```

8.2.9.4 Example of Automatic Classification

Assume that a CPP is submitted that defines two roles as “seller” and “buyer." When the CPP is submitted it will automatically be classified by two ClassificationNodes named “buyer” and “seller” that are both children of the ClassificationNode (e.g. a node named Role) specified in the classificationNode attribute of the ClassificationIndex. Note that if either of the two ClassificationNodes named “buyer” and “seller” did not previously exist, the ObjectManager would automatically create these ClassificationNodes.

8.2.10 Ad Hoc Query Request/Response

A client submits an ad hoc query to the ObjectQueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains the query string in the queryString attribute.

The ObjectQueryManager sends an AdhocQueryResponse either synchronously or asynchronously back to the client. The AdhocQueryResponse return a collection of objects whose element type is in the set of element types represented by the leaf nodes of the ManagedObject hierarchy in [RIM].

Figure 12: Submit Ad Hoc Query Sequence Diagram

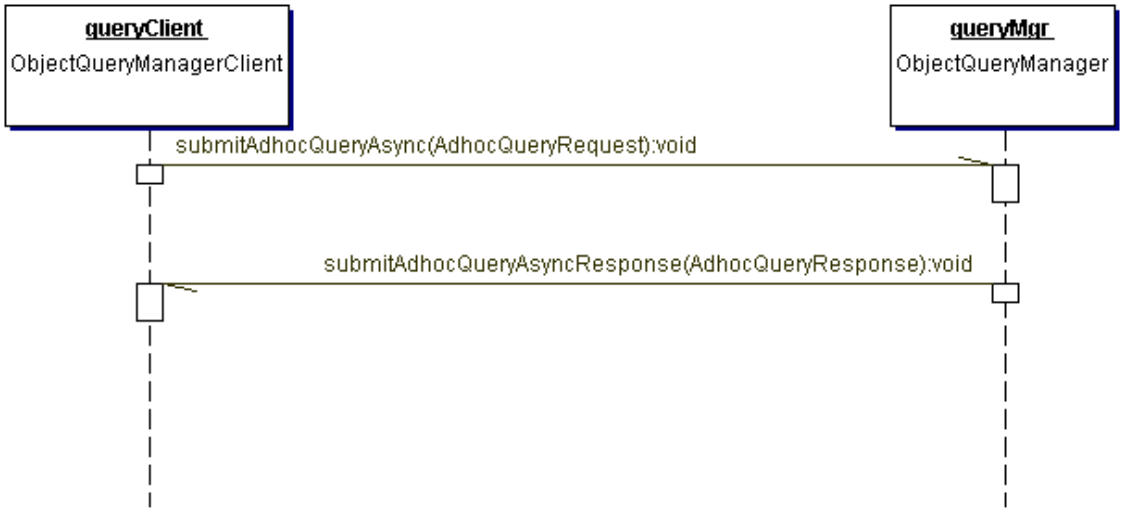

Figure 13: Submit Ad Hoc Query Asynchronous Sequence Diagram

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

8.3 Content Retrieval

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

8.3.1 Identification Of Content Payloads

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

- Use the ID for each ManagedObject instance that describes the managed object content as the DocumentLabel element in the DocumentReference for that object in the Manifest element of the ebXMLHeader.

8.3.2 GetContentResponse Message Structure

The following message fragment illustrates the structure of the GetContentResponse Message that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs for the requested objects. Note that the ID for each object retrieved in the message as additional payloads is used as its DocumentLabel in the Manifest of the ebXMLHeader.

```
…
--7250537.978150567601.JavaMail.najmi.irian
…
<ebXMLHeader MessageType="Normal" Version="1.0">
  <Manifest>
    <DocumentReference>
      <DocumentLabel>GetContentsResponse</DocumentLabel>
      <DocumentId>6835fb:e3be512ac8:-8000</DocumentId>
    </DocumentReference>
    <DocumentReference>
      <DocumentLabel> ID for CPP content #1 </DocumentLabel>
      <DocumentId>....</DocumentId>
```

```
    </DocumentReference>
    <DocumentReference>
      <DocumentLabel> ID for CPP content #2 </DocumentLabel>
      <DocumentId>… </DocumentId>
    </DocumentReference>
  </Manifest>
  <Header>
  …
  </Header>
--7250537.978150567601.JavaMail.najmi.irian
Content-Type: application/xml
Content-Description: GetContentsResponse
Content-ID: 6835fb:e3be512ac8:-7ffc
Content-Length: 97

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

--7250537.978150567601.JavaMail.najmi.irian
Content-Type: application/xml
Content-Description: ID for CPP content #1
Content-ID: ….
…
<CPP>
…
</CPP>
--7250537.978150567601.JavaMail.najmi.irian
Content-Type: application/xml
Content-Description: ID for CPP content #2
Content-ID: ….
…
<CPP>
…
</CPP>
--7250537.978150567601.JavaMail.najmi.irian--
```

8.4 Query And Retrieval: Typical Sequence

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

Figure 14: Typical Query and Retrieval Sequence

9 Registry Security

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [RIM].

In the current version of this specification, a minimalist approach has been specified for Registry security. The philosophy is that “Any *known* entity can publish content and *anyone* can view published content.” The Registry information model has been designed to allow more sophisticated security policies in future versions of this specification.

9.1 Integrity of Registry Content

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

9.1.1 Message Payload Signature

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

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

9.2 Authentication

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

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

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

9.2.1 Message Header Signature

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

9.3 Confidentiality

9.3.1 On-the-wire Message Confidentiality

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

9.3.2 Confidentiality of Registry Content

In the current version of this specification, there are no provisions for confidentiality of Registry content. All content submitted to the Registry may be discovered and read by *any* client. The Registry must decrypt any submitted content after it has been received and prior to storing it in its repository.

9.4 Authorization

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

9.4.1 Pre-defined Roles For Registry Users

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

Role	Description
ContentOwner	The submitter or owner of a Registry content. Submitting Organization (SO) in ISO 11179
RegistryAdministrator	A “super” user that is an administrator of the Registry. Registration Authority (RA) in ISO 11179
RegistryGuest	Any unauthenticated user of the Registry. Clients that browse the Registry do not need to be authenticated.

9.4.2 Default Access Control Policies

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

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

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

947

948 The following list summarizes the default role-based AccessControlPolicy:

949 ?? The Registry must implement the default AccessControlPolicy and
950 associate it with all Objects in the Registry

951 ?? Anyone can publish content, but needs to be authenticated

952 ?? Anyone can access the content without requiring authentication

953 ?? The ContentOwner has access to all methods for Registry Objects owned
954 by them

955 ?? The RegistryAdministrator has access to all methods on all Registry
956 Objects

957 ?? Unauthenticated clients can access all read-only (getXXX) methods

958 ?? At the time of content submission, the Registry must assign the default
959 ContentOwner role to the Submitting Organization (SO) as authenticated
960 by the credentials in the submission message. In the current version of
961 this specification, it will be the DN as identified by the certificate

962 ?? Clients that browse the Registry need not use certificates. The Registry
963 must assign the default RegistryGuest role to such clients.

964

965 Appendix A Schemas and DTD Definitions

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

968 A.1 ebXMLError Message DTD

969 See [ERR] for ebXMLError Message DTD.

A.2 ebXML Registry DTD

```
<?xml version='1.0' encoding='UTF-8' ?>

<!--Generated by XML Authority-->
<!-- $Header: /jse/jaxr/schema/Registry.dtd,v 1.7 2001/01/10 17:56:28 najmi Exp
$ -->
<!ENTITY % errorSchema SYSTEM "ebXMLError.dtd">

%errorSchema;

<!ENTITY % VersionAttribute " version CDATA  #REQUIRED">

<!ENTITY % ObjectAttributes " description CDATA  #IMPLIED
                ID        CDATA  #REQUIRED
                name      CDATA  #REQUIRED">

<!ENTITY % ManagedObjectAttributes " %ObjectAttributes;
                status          (SUBMITTED | APPROVED | DEPRECATED )
'SUBMITTED'
                majorVersion    CDATA  '1'
                minorVersion    CDATA  '0'">

<!ELEMENT ManagedObject EMPTY>
<!ATTLIST ManagedObject  %ManagedObjectAttributes; >
<!ELEMENT ExtrinsicObject EMPTY>
<!ATTLIST ExtrinsicObject  %ManagedObjectAttributes;
                    contentURN          CDATA    #IMPLIED
                    mimeType            CDATA    #IMPLIED
                    objectType          (PARTY_AGREEMENT |
                                        PARTY_PROFILE |
                                        PROCESS |
                                        ROLE |
                                        SERVICE_INTERFACE |
                                        SOFTWARE_COMPONENT |
                                        TRANSPORT |
                                        UML_MODEL |
                                        UNKNOWN |
                                        XML_SCHEMA )  #REQUIRED
                    opaque              CDATA    'false'
                    a-dtype             NMTOKENS  'opaque boolean' >
<!--
A ClassificationIndex is specified on SCHEMA ExtrinsicObjects to define
an automatic Classification of instance objects of the schema using
the specified classificationNode as parent and a ClassificationNode
```

```
created or selected by the object content as selected by the contentIdentifier
-->
<!ELEMENT ClassificationIndex EMPTY>
<!ATTLIST ClassificationIndex  %ObjectAttributes;
                        classificationNode CDATA  #REQUIRED
                        contentIdentifier  CDATA  #REQUIRED >
<!-- ClassificationIndexList contains new ClassificationIndexs  -->
<!ELEMENT ClassificationIndexList  (ClassificationIndex )*>

<!ENTITY % IntrinsicObjectAttributes " %ManagedObjectAttributes;">

<!ELEMENT IntrinsicObject EMPTY>
<!ATTLIST IntrinsicObject  %ManagedObjectAttributes; >
<!-- Leaf classes that reflect the concrete classes in RIM -->
<!ELEMENT ManagedObjectList  (Association | Classification |
ClassificationNode | ExternalLink | Organization | ExtrinsicObject )*>

<!-- Reference to an Object via its URN specified by it ID attribute -->
<!ELEMENT ObjectRef EMPTY>
<!ATTLIST ObjectRef  uuid CDATA  #REQUIRED >
<!ELEMENT ObjectRefList  (ObjectRef )*>

<!--
An ExternalLink specifies a link from a ManagedObject and an external URI

The sourceObjectRef is ref to the ManagedObject

The sourceObjectRef is optional when Association is defined as part of
a SubmittedObject.
-->
<!ELEMENT ExternalLink EMPTY>
<!ATTLIST ExternalLink  %IntrinsicObjectAttributes;
                sourceObjectRef         CDATA  #IMPLIED
                uri                     CDATA  #IMPLIED >
<!-- ExternalLinkList contains new ExternalLinks or refs to pre-existing
ExternalLinks -->
<!ELEMENT ExternalLinkList  (ExternalLink | ObjectRef )*>

<!--
An Association specifies references to two previously submitted
managed objects.

The sourceObjectRef is ref to the sourceObject in association
The targetObjectRef is ref to the targetObject in association
```

```
The sourceObjectRef is optional when Association is defined part of
a SubmittedObject.
-->
<!ELEMENT Association EMPTY>
<!ATTLIST Association  %IntrinsicObjectAttributes;
                fromLabel             CDATA    #IMPLIED
                toLabel               CDATA    #IMPLIED
                associationType         (CLASSIFIED_BY |
                                CONTAINED_BY |
                                CONTAINS |
                                EXTENDS |
                                IMPLEMENTS |
                                INSTANCE_OF |
                                RELATED_TO |
                                SUPERSEDED_BY |
                                SUPERSEDES |
                                USED_BY |
                                USES )  #FIXED 'RELATED_TO'
                bidirection           CDATA    'false'
                sourceObjectRef         CDATA    #REQUIRED
                targetObjectRef         CDATA    #REQUIRED
                a-dtype               NMTOKENS  'bidirection boolean' >
<!ELEMENT AssociationList  (Association )*>

<!--
A Classification specifies references to two previously submitted
managed objects.

The sourceObjectRef is ref to the sourceObject in Classification
The targetObjectRef is ref to the targetObject in Classification

The sourceObjectRef is optional when Classification is defined as part of
a SubmittedObject.
-->
<!ELEMENT Classification EMPTY>
<!ATTLIST Classification  %IntrinsicObjectAttributes;
                  sourceObjectRef         CDATA  #REQUIRED
                  targetObjectRef         CDATA  #REQUIRED >
<!ELEMENT ClassificationList  (Classification )*>

<!ELEMENT Package EMPTY>
<!ATTLIST Package  %IntrinsicObjectAttributes; >
<!-- PackageList contains new Packages or refs to pre-existing Packages -->
<!ELEMENT PackageList  (Package | ObjectRef )*>
```

```
<!ENTITY % TelephoneNumberAttributes " areaCode   CDATA  #REQUIRED
                contryCode CDATA  #REQUIRED
                extension  CDATA  #IMPLIED
                number     CDATA  #REQUIRED
                url        CDATA  #IMPLIED">

<!ELEMENT TelephoneNumber EMPTY>
<!ATTLIST TelephoneNumber  %TelephoneNumberAttributes; >
<!ELEMENT FaxNumber EMPTY>
<!ATTLIST FaxNumber  %TelephoneNumberAttributes; >
<!ELEMENT MobileTelephoneNumber EMPTY>
<!ATTLIST MobileTelephoneNumber  %TelephoneNumberAttributes; >
<!-- PostalAddress -->
<!ELEMENT PostalAddress EMPTY>
<!ATTLIST PostalAddress  city      CDATA  #REQUIRED
                  country    CDATA  #REQUIRED
                  postalCode CDATA  #REQUIRED
                  state      CDATA  #REQUIRED
                  street     CDATA  #REQUIRED >
<!-- PersonName -->
<!ELEMENT PersonName EMPTY>
<!ATTLIST PersonName  firstName  CDATA  #REQUIRED
                middleName CDATA  #REQUIRED
                lastName   CDATA  #REQUIRED >
<!-- Contact -->
<!ELEMENT Contact  (PostalAddress , PersonName , FaxNumber? ,
TelephoneNumber , MobileTelephoneNumber? )>
<!ATTLIST Contact  email CDATA  #REQUIRED >
<!-- Organization -->
<!ELEMENT Organization  (PostalAddress , Contact , FaxNumber? ,
TelephoneNumber )>
<!ATTLIST Organization  %IntrinsicObjectAttributes;
                  parent                CDATA  #IMPLIED >
<!--
ClassificationNode is used to submit a Classification tree to the Registry.
Note that this is a recursive schema definition.

The parent attribute of a node in tree is implied by the enclosing
ClassificationNode
The children nodes of a node are implied by enclosing immediate child elements
of type ClassificationNode.
-->
<!ELEMENT ClassificationNode EMPTY>
<!ATTLIST ClassificationNode %IntrinsicObjectAttributes;>
```

```
<!--
parent is the URN to the parent node. parent is optional if ClassificationNode is
enclosed
in a parent ClassificationNode or if it is a root ClassificationNode
-->
<!ATTLIST ClassificationNode parent              CDATA  #IMPLIED>

<!ELEMENT ClassificationNodeList  (ClassificationNode )*>

<!--
End information model mapping.

Begin Registry Services Interface
-->
<!ELEMENT RequestAcceptedResponse EMPTY>
<!ATTLIST RequestAcceptedResponse  %VersionAttribute;
                        xml:lang       NMTOKEN  #REQUIRED
                        interfaceId    CDATA    #REQUIRED
                        requestMessage CDATA    #REQUIRED
                        actionId       CDATA    #REQUIRED >
<!--
The SubmittedObject provides meta data for submitted object
Note object being submitted is in a separate document that is not
in this DTD.
-->
<!ELEMENT SubmitObjectsRequest  (SubmittedObject+ )>
<!ATTLIST SubmitObjectsRequest  %VersionAttribute; >
<!--
The ExtrinsicObject provides meta data about the object being submitted
ClassificationList can be optionally specified to define Classifications
for the SubmittedObject

AssociationList can be optionally specified to define Associations
for the SubmittedObject

The ExternalLinkList provides zero or more external objects related to
the object being submitted.
-->
<!ELEMENT SubmittedObject  (ExtrinsicObject? , ClassificationIndexList? ,
ClassificationList? , AssociationList? , ExternalLinkList? , PackageList? )>

<!--
The ObjectRefList is the list of
refs to the managed objects being approved.
-->
```

```
<!ELEMENT ApproveObjectsRequest  (ObjectRefList )>
<!ATTLIST ApproveObjectsRequest  %VersionAttribute; >
<!--
The ObjectRefList is the list of
refs to the managed objects being deprecated.
-->
<!ELEMENT DeprecateObjectsRequest  (ObjectRefList )>
<!ATTLIST DeprecateObjectsRequest  %VersionAttribute; >
<!--
The ObjectRefList is the list of
refs to the managed objects being removed
-->
<!ELEMENT RemoveObjectsRequest  (ObjectRefList )>
<!ATTLIST RemoveObjectsRequest  %VersionAttribute; >
<!ELEMENT GetRootClassificationNodesRequest EMPTY>
<!ATTLIST GetRootClassificationNodesRequest %VersionAttribute;>

<!--
The namePattern follows SQL-92 syntax for the pattern specified in
LIKE clause. It allows for selecting only those root nodes that match
the namePattern. The default value of '*' matches all root nodes.
-->
<!ATTLIST GetRootClassificationNodesRequest namePattern     CDATA  "*">

<!--
The response includes a ClassificationNodeList which has zero or more
ClassificationNodes
-->
<!ELEMENT GetRootClassificationNodesResponse  (ClassificationNodeList |
ebXMLError )>
<!ATTLIST GetRootClassificationNodesResponse  %VersionAttribute; >
<!--
Get the classification tree under the ClassificationNode specified parentRef.

If depth is 1 just fetch immediate child
nodes, otherwise fetch the descendant tree upto the specified depth level.
If depth is 0 that implies fetch entire sub-tree
-->
<!ELEMENT GetClassificationTreeRequest EMPTY>
<!ATTLIST GetClassificationTreeRequest  %VersionAttribute;
                        parent        CDATA  #REQUIRED
                        depth         CDATA  '1' >
<!--
The response includes a ClassificationNodeList which includes only
immediate ClassificationNode children nodes if depth attribute in
```

```
GetClassificationTreeRequest was 1, otherwise the decendent nodes
upto specified depth level are returned.
-->
<!ELEMENT GetClassificationTreeResponse  (ClassificationNodeList |
ebXMLError )>
<!ATTLIST GetClassificationTreeResponse  %VersionAttribute; >
<!--
Get refs to all managed objects that are classified by all the
ClassificationNodes specified by ObjectRefList.
Note this is an implicit logical AND operation
-->
<!ELEMENT GetClassifiedObjectsRequest  (ObjectRefList )>

<!--
objectType attribute can specify the type of objects that the registry
client is interested in, that is classified by this ClassificationNode.
It is a String that matches a choice in the type attribute of ExtrinsicObject.
The default value of '*' implies that client is interested in all types
of managed objects that are classified by the specified ClassificationNode.
-->
<!--
The response includes a ManagedObjectList which has zero or more
ManagedObjects that are classified by the ClassificationNodes
specified in the ObjectRefList in GetClassifiedObjectsRequest.
-->
<!ELEMENT GetClassifiedObjectsResponse  (ManagedObjectList | ebXMLError
)>
<!ATTLIST GetClassifiedObjectsResponse  %VersionAttribute; >
<!--
An Ad hoc query request specifies a query string as defined by [RS] in the
queryString attribute
-->
<!ELEMENT AdhocQueryRequest EMPTY>
<!ATTLIST AdhocQueryRequest  %VersionAttribute;
                        queryString     CDATA  #REQUIRED >
<!--
The response includes a ManagedObjectList which has zero or more
ManagedObjects that match the query specified in AdhocQueryRequest.
-->
<!ELEMENT AdhocQueryResponse  (ManagedObjectList | ebXMLError )>
<!ATTLIST AdhocQueryResponse  %VersionAttribute; >
<!--
Gets the actual content (not metadata) specified by the ObjectRefList
-->
<!ELEMENT GetContentRequest  (ObjectRefList )>
```

```
<!ATTLIST GetContentRequest  %VersionAttribute; >
<!--
The GetObjectsResponse will have no sub-elements if there were no errors.
The actual contents will be in the other payloads of the message.
If any errors were encountered the message will contain the ebXMLError and
the content payloads will be empty.
-->
<!ELEMENT GetContentResponse  (ebXMLError? )>
<!ATTLIST GetContentResponse  %VersionAttribute; >
<!--
The contrived root node
-->
<!ELEMENT RootElement  (RequestAcceptedResponse | ebXMLError |
SubmitObjectsRequest | ApproveObjectsRequest | DeprecateObjectsRequest |
RemoveObjectsRequest | GetRootClassificationNodesRequest |
GetRootClassificationNodesResponse | GetClassificationTreeRequest |
GetClassificationTreeResponse | GetClassifiedObjectsRequest |
GetClassifiedObjectsResponse | AdhocQueryRequest | AdhocQueryResponse |
GetContentRequest | GetContentResponse )>
```

Appendix B Interpretation of UML Diagrams

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

B.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces (as defined by [CPA]) required to implement an ebXML Registry Services and clients. See Figure 1 on page 12 for an example. The UML class diagram contains:

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

B.2 UML Sequence Diagram

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

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

Each method invocation may be followed by a response method invocation from the responder to the requestor to indicate the ResponseName for the previous Request. Possible error response is indicated by a conditional response method invocation from the responder to the requestor. See Figure 3 on page 19 for an example.

Appendix C BNF for Query Syntax Grammar

The following BNF defines the grammar for the registry query syntax. This grammer is a proper sub-set of SQL-92 as defined by [SQL].

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

RegistryQuery = SQLSelect [";"]

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

SQLSelectCols = ( "ALL" | "DISTINCT" )* [ ID ]

SQLTableList = SQLTableRef ( "," SQLTableRef )*

SQLTableRef = ID  [ ID  ]

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*

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

```
SQLNotExpr = [ "NOT" ] SQLCompareExpr

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


SQLCompareExprRight =
   SQLLikeClause
  | SQLInClause
  | SQLCompareOp SQLSumExpr

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

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

SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*

SQLLValueElement = "NULL" | SQLSumExpr

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

SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern

SQLPattern = STRING_LITERAL

SQLLiteral =
   STRING_LITERAL
  | INTEGER_LITERAL
  | FLOATING_POINT_LITERAL

SQLColRef = SQLLvalue

SQLLvalue =  SQLLvalueTerm

SQLLvalueTerm = ID ( "."  ID )*

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

```

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

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

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

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

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

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

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

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

Appendix D Security Implementation Guideline

This section provides a suggested blueprint for how security processing may be implemented in the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have different implementations as long as they support the default security roles and authorization rules described in this document.

D.1 Authentication

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

4. Then the message is processed for the command and the objects it will act on

D.2 Authorization

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

D.3 Registry Bootstrap

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

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

D.4 Content Submission – Client Responsibility

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

D.5 Content Submission – Registry Responsibility

1. Like any other request, the client will be first authenticated. In this case, the Principal object will get the DN from the certificate.
2. As per the request in the message, the ManagedObject will be created.
3. The ManagedObject is assigned the singleton default AccessControlPolicy.
4. If a principal with the identity of the SO is not available, an identity object with the SO's DN is created
5. A principal with this identity is created

D.6 Content Delete/Deprecate – Client Responsibility

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

D.7 Content Delete/Deprecate – Registry Responsibility

1. Like any other request, the client will be first authenticated. In this case, the Principal object will get the DN from the certificate. As there will be a principal with this identity in the Registry, the Principal object will get all the roles from that object
2. As per the request in the message (delete or deprecate), the appropriate method in the Object will be accessed.
3. The access controller performs the authorization by iterating through the Permission objects associated with this object via the singleton default AccessControlPolicy.
4. If authorization succeeds then the action will be permitted. Otherwise an error response is sent back with a suitable AuthorizationException error message.

Appendix E Terminology Mapping

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

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

This Document	OASIS	ISO 11179
“managed object content”	Registered Object	
ManagedObject	Registry Item	Administered Component
ExternalObject	Related Data	N/A
Object.ID	RaItemId	
ExtrinsicObject.uri	ObjectLocation	
ExtrinsicObject.objectType	DefnSource, PrimaryClass, SubClass	
ManagedObject.name	CommonName	
Object.description	Description	
ExtrinsicObject.mimeType	MimeType	
Versionable.majorVersion	partially to Version	
Versionable.minorVersion	partially to Version	
ManagedObject.status	RegStatus	

Table 1: Terminology Mapping Table

10 References

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

[TA] ebXML Technical Architecture

http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf

[OAS] OASIS Information Model

http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html

[ISO] ISO 11179 Information Model

http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

[BDM] Registry and Repository: Business Domain Model

http://www.ebxml.org/specdrafts/RegRepv1-0.pdf

[RIM] ebXML Registry Information Model

http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf

[BPM] ebXML Business Process Metamodel Specification Schema

http://www.ebxml.org/specdrafts/Busv2-0.pdf

[CPA] Trading-Partner Specification

http://www.ebxml.org/project_teams/trade_partner/private/

[CTB] Context table informal document from Core Components

[MS] ebXML Messaging Service Specification, Version 0.21

http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf

[ERR] ebXML TRP Error Handling Specification

http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf

[SEC] ebXML Security Specification

http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html

[XPT] XML Path Language (XPath) Version 1.0

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

[SQL] Structured Query Language (FIPS PUB 127-2)

http://www.itl.nist.gov/fipspubs/fip127-2.htm

11 Disclaimer

The views and specification expressed in this document are those of the authors and are not necessarily those of their employers. The authors and their employers specifically disclaim responsibility for any problems arising from correct or incorrect implementation or use of this design.

12 Contact Information

Team Leader

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

Vice Team Lead

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

Editor

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

Copyright Statement