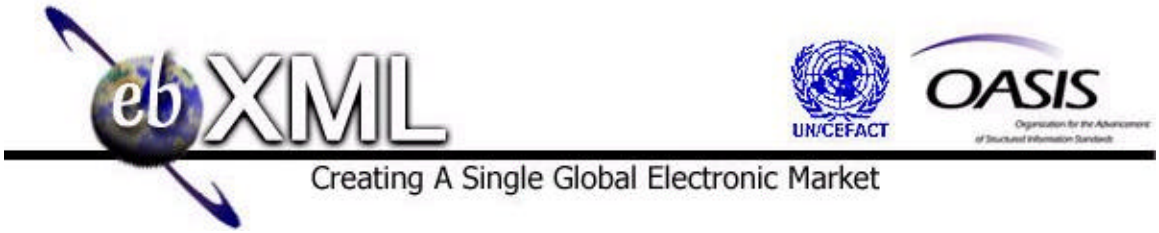


ebXML Registry And Repository Registry Services Proposal

Working Draft 9/28/2000 8:20 AM

This version:

RegistryServicesSpecification v0-7.doc

Abstract

This document is a draft proposal whose purpose is to solicit additional input and convey the current state of the ebXML Registry Service recommendations.

This document defines the various Registry Services as interaction protocols and processes between an ebXML capable party and the ebXML Registry. It is assumed that all interactions between the party and the ebXML registry will be conducted using ebXML Messaging Service.

Notational Conventions

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 Key Words for Use in RFC's to Indicate Requirement Levels (RFC 2119).

Status of this Document

This document represents work in progress upon which no reliance should be made.

Document Version History

- Version 0.1: Initial version (not released)
- Version 0.2: Corrections based on internal review (not released)
- Version 0.3: Added DocumentManager, Removed SchemaManager and ProcessManager
- Version 0.4: Added support for Querying the Registry. Added DTDs for all documents currently defined

- Version 0.5: Major changes based on feedback from team review resulted in a complete re-write from previous version. Replaced DocumentManager with ObjectManager and associated name changes (e.g. DocumentInfo replaced with ManagedObject). Factored out information model aspects in separate document [3]. Added QueryManager. Removed TPA Manager. Changed terminology where relevant to better align with [2]. Changed Registrar to Registry, Registrant to RegistryClient. Completely redid the DTD definition based on Repository Information Model spec [3].
- Version 0.6: Changes based on issue logged during review of v0.5.
- Version 0.7: Changed RequestErrorResponse with ebXMLError defined by draft TRP error handling specification. Changed registerOrganization to registerParty. Changes made to almost all DTDs with major changes in the area of classification. This was due to simplifications in classification model. DTDs now allow for multiple object submission with optional classifications defined. The new DTDs also allow for multiple object approval, deprecation, and removal. Changed GetClassifiedObjectsRequest so that multiple classifications may be specified.

41

41 Table of Contents

Table of Figures

Table of Tables

1 Introduction

This document defines the ebXML Registry Services as a set of specialized business processes. Clients of the ebXML Registry are referred to as RegistryClients while the ebXML Registry itself is referred to as Registry.

[Note] For interoperability reasons it is required that a Registry implementation *must* support its service interfaces using the ebXML Messaging Service bindings defined in this document. A Registry implementation is free to provide any other technology bindings (e.g. LDAP) as a non-interoperable implementation specific detail.

All interaction between RegistryClients and the Registry are treated as if they are B2B interactions between trading partners. Thus the processes supported by the Registry are described in terms of:

- A special TPA between the Registry and RegistryClient
- A set of business processes involving the Registry and RegistryClient
- A set of business messages that are exchanged between the Registry and RegistryClient as part of a specific Registry business process

1.1 Purpose and Scope

This document provides sufficient detail to develop:

- The specified functionality of the ebXML Registry Services
- ebXML based applications that utilize the Registry Services functionality specified in this document

Software practitioners MAY use this document in combination with other ebXML specification documents when creating ebXML compliant software.

NOTE: This version describes only some of the Registry Services that are likely to be needed by the proposed Tokyo POC scenarios. Several Registry Services identified in [2] are currently unspecified and will be added in a later version. Some of the missing functionality includes:

- **Security (assuming it will be provided by ebXML Messaging Service Specification)**
- **Transformation Service**
- **Workflow Service**
- **Quality Assurance Service.**

1.1.1 Goals

The goals of this version of the specification are to:

- Communicate functionality of registry services to software developers
- Meet the immediate requirements of the Tokyo POC demo scenarios
- Able to evolve in future to support more complete ebXML Registry and Repository requirements

- Be compatible with other ebXML specifications

1.2 Related ebXML Specifications

The following set of related specifications may be of interest to the reader:

a) Registry and Repository Part 1: Business Domain Model [2]

b) ebXML Repository Information Model [3]

c) Messaging Service Specification [4]

d) Business Process Meta Model Specification [5]

e) Trading-Partner Specification [7]

1.3 General Conventions

- *"managed object"* is used to refer to actual repository content (not meta data) instance (e.g. a DTD)
- *"ManagedObject"* is used to refer to an object that provides meta data about content instance (managed object).
- The information model *does not* contain *any* elements that are the actual content of the repository (managed object). All elements of the information model represent meta data about the content and not the content itself.
- UML diagrams are used as a way to concisely describe business processes, collaboration and concepts. They are not intended to convey any specific implementation requirements.
- The Registry Service processes are described as UML diagrams. The intent is to describe business processes in a concise manner in terms of ebXML TRP. The reader must acquaint themselves with section 3.1 (abstract) and section 3.2 (concrete example), in order to understand these processes based on their UML description.

1.4 Guiding Principals

The following principals guided the work represented by this document:

- Keep it Simple (KISS)
- All access to repository content is through interaction with Registry Services
- Interactions between Registry Services and its client are a special case of partner-to-partner business process. In fact there is a TPA between clients and Registry Services
- Interactions between Registry Services and its client are based on ebXML TRP messaging

1.5 Specification Structure

This specification is organized around the following main topics:

- **Overview** - A high level description of the Registry Service

- **Registration Service**- A description of the initial registration functionality that bootstraps the communication between a Party associated with a Submitting Organization and the Registry Services. *This section is important to understand fully as it serves as a blueprint when reading subsequent sections on how to interpret UML representation of the registry services interfaces.*

- **Object Management Service**- A description of the Object Management functionality of the ebXML Registry that simplifies the definition and subsequent sharing of business objects (e.g. documents) between partners. The types of objects that may be shared are defined in [3] and include Schema documents (e.g. DTDs), Business Process descriptions (e.g. XMI documents and software components compliant to a registered business process), Party Profiles and TPAs.

- **Object Query Management Service**- A description of the Object Query Management functionality of the ebXML Registry that enables querying the repository for managed objects.

2 Overview

2.1 Role of ebXML Registry

The ebXML Registry provides a set of distributed 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 an ebXML Repository [3] that is managed by the ebXML Registry Services defined in this document and its future versions. All access to the repository is exposed via the interfaces defined for the Registry Services.

2.2 Use Cases for the Registry Services

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

2.2.1 Business Domain Workflow Use Cases

Figure 1: shows the Business Domain Workflow Use Cases as identified in [2].

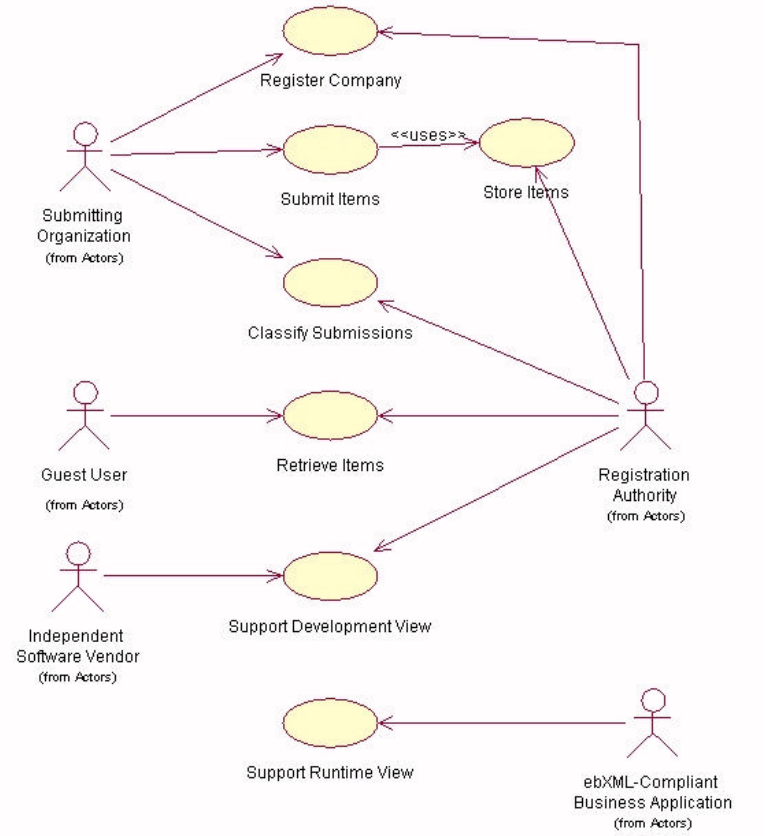


Figure 1: Business Domain Workflow Use Cases

The following scenario textually exemplifies a *subset* of above use cases in terms of interaction between registry clients and the registry. It is currently not a complete listing of use cases envisioned in [2]. It assumes for purposes of example, a buyer and a seller that wish to conduct B2B exchanges using the Rosetta Net PIP3A4 Purchase Order business protocol. It is assumed that both buyer and seller use the same registry service provided by a 3rd party. Note that the architecture supports other possibilities (e.g. each party uses their own private registry). It is assumed that the Registry Service is always operational for the use cases described below.

2.2.2 Parties Register With Registry

Both parties register with the Registry using the Registration Service described in section 3.

2.2.3 Schema Documents Are Submitted

Either the buyer or the seller or a registered 3rd party can submit the necessary schema documents required by the Rosetta Net PIP3A4 Purchase Order business protocol with the Registry using the Object Manager service of the Registry described in section 4.3.

2.2.4 Business Process Documents Are Submitted

Either the buyer or the seller or a registered 3rd party can submit the necessary business process documents required by the Rosetta Net PIP3A4 Purchase Order business protocol with the Registry using the Object Manager service of the Registry described in section 4.3.

2.2.5 TPA is Submitted

Either the buyer or the seller can now create a concrete TPA document that will be used by both parties to conduct B2B exchanges based on the Rosetta Net PIP3A4 Purchase Order business protocol. This is done using a TPA Assembler tool that is a client of the registry services and uses registry services to assemble a TPA from submitted TPA elements. The resulting TPA is submitted to the Registry using the Object Manager service of the Registry described in section 4.3.

Once the TPA is submitted the parties may now begin to conduct B2B transaction as defined by [4].

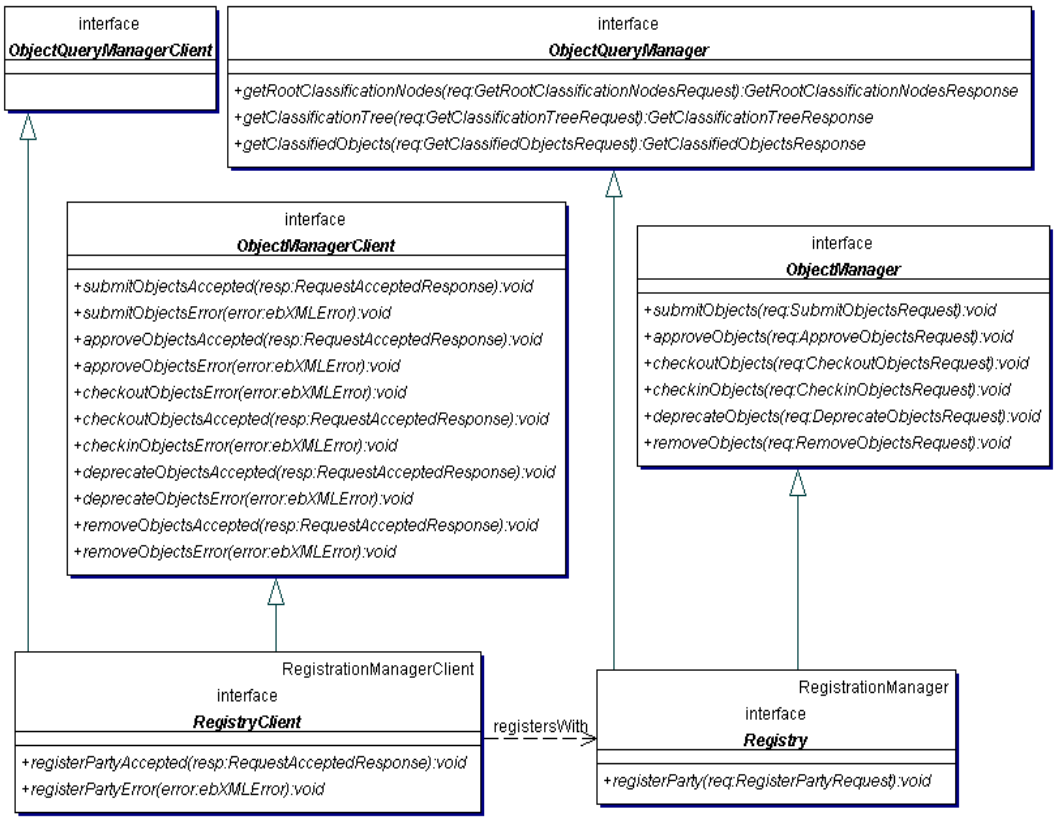


Figure 2: Registry Services Class Diagram

Figure 2: Registry Services Class Diagram shows an abstract class diagram for the Registry Service. It is intended as a high level overview and does not prescribe any particular implementation. The Registry makes use of a Repository (not shown in diagram) for storing and retrieving persistent information required by the Registry Services.

2.3 Interfaces Implemented By Registry Service

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

1. **Registry**: This is the principal bootstrapping interface used by clients of Registry to register themselves as RegistryClients.
2. **ObjectManager**: This is the Object Management interface of the Registry services. It provides the functionality to manage the life cycle of any managed object in the repository.
3. **ObjectQueryManager**: This is the Object Query Management interface of the Registry services. It provides the functionality to receive query requests and return managed objects that match the specified query as a response.

2.4 Interfaces Implemented By Clients of Registry Service

An ebXML application that is a client of the Registry Service must implement the following interfaces:

1. **RegistryClient**: This is the principal interface implemented by a client of the Registry Services. It is used by the Registry to communicate with the ebXML registry client application during the initial bootstrapping registration process.
2. **ObjectManagerClient**: This is the interface implemented by a client of the Object Management sub-service of the Registry Services. It is used by the ObjectManager to communicate with the ebXML application during the Object Management process.
3. **ObjectQueryManagerClient**: This is the interface implemented by a client of the Object Query sub-service of the Registry Services. It is used by the ObjectQueryManager to communicate with the RegistryClient during the Object Query process.

3 Registration Service

In order to submit or otherwise change content, a RegistryClient must register itself as Party associated with an Organization with the Registry. An Organization currently can have a role of a SubmittingOrganization, ResponsibleOrganization or a RegistrationAuthority as defined by [6].This is an essential bootstrapping process that is required prior to any other interaction between the RegistryClient and the Registry Service.

[Note] Clients that only intend to browse the repository content do not have to register with the repository.

Because there is no previously established TPA between the Registry and the RegistryClient, the Registry must somehow make public at least one well-known Transport specific communication address. It is recommended that the registry at least make public a URL to its Registry Service Interface.

This section describes the bootstrapping Registration Protocol of the Registry Service that allows a client to register itself with the Registry.

Since this is the first use of the stylized use of UML notation in this document, the diagram will be followed by its interpretation. Future processes in the document will only be described pictorially in the interest of brevity.

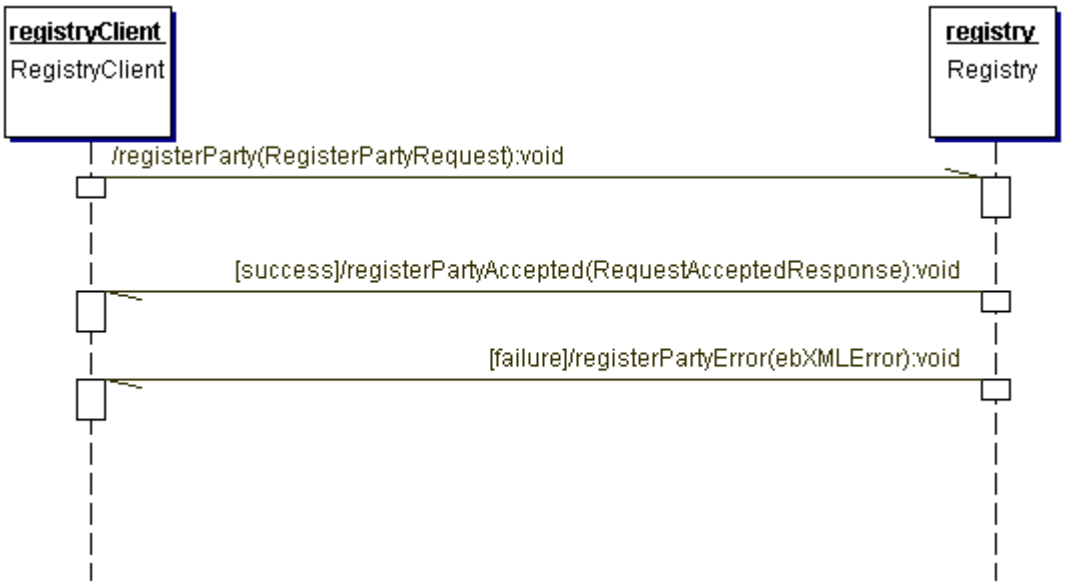


Figure 2: Bootstrap Registration Process Sequence Diagram

3.1 Interpretation of UML Diagrams Describing an ebXML Business Process

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

3.1.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces and Actions (as defined by [7]) required to implement an ebXML business process. See Figure 2: Registry Services Class Diagram for an example. The UML class diagram contains:

1. A collection of UML classes where each class represents an ebXML document. Such class definitions, their attributes and their relationships can be used to create schema elements by a schema assembler tool that is a RegistryClient.
2. A collection of UML interfaces where each interface represents a Service Interface.
3. A collection of methods on each interface where each method represents an Action (as defined by [7]) within the Service Interface representing the UML interface.
4. 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.

3.1.2 UML Sequence Diagram

A UML sequence diagram is used to specify the business protocol representing the interactions between the UML interfaces for an 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 [7].

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. 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 2 for an example.

3.2 Interpretation of Bootstrap Registration Process Sequence Diagram

This section describes in *concrete terms* the conventions used to define ebXML business process description in UML diagrams. It uses the Bootstrap Registration Process as a concrete example.

There is an implicit TPA between the Registry and the ebXML application that is a client to the Registry services. This TPA defines the Bootstrap Registration Process shown above.

3.2.1 Service Interfaces Defined

In the implicit TPA there are two Business Interfaces which are represented by the UML interfaces in UML sequence diagram above. The Registry must export a Service Interface called the Registry interface. The registry client must export a Service Interface called the RegistryClient interface.

3.2.2 The Actions Defined On Service Interfaces

The Registry interface must support an action named registerParty. The RegistryClient uses the registerParty action of the Registry interface to register its Party and associated Organization with the Registry.

The RegistryClient interface must support the following actions for receiving responses to requests made to the Registry:

1. A registerPartyAccepted action that is used by Registry to notify RegistryClient of successful registration. This serves as a business level Acknowledgement Response to the register action.
2. A registerPartyError action that is used by Registry to notify RegistryClient of a failure during registration. This serves as an Error Response to the register action.

3.2.3 Requests Defined For Action

The Registry interface defines an action with id of registerParty (corresponding to the interface method).

3.2.3.1 Requests Messages Defined For Request

The register action has a request whose RequestName and RequestMessage are both RegisterPartyRequest. This name is derived from the action name (with first letter capitalized) appended with the suffix Request.

3.2.3.2 Responses Defined For Request

Each Request message may have a Response message associated with it. The ResponseName is inferred from the sequence diagram from the name of the method called upon success of the Request method invocation.

Most Registry Requests in this document consistently use a standard business level acknowledgement response message of type RequestAcceptedResponse.

3.2.3.3 Exception Responses Defined For Request

Each Request message may have one or more Exception Response messages associated with it. The ExceptionResponseName is inferred from the sequence diagram from the name of the method called upon failure of the Request method invocation.

Most Registry Requests in this document consistently use the generic ebXMLError message defined by [8] as a business level error response message.

3.2.4 Messages Defined For Requests and Responses

The type of ebXML message exchanged during each interaction (method invocation in diagram) can be inferred from the type of argument for each method invocation.

The RegisterPartyRequest message must contain either an Organization element that provides information on the Organization being registered, or a ManagedObjectRef element that is a reference to a previously defined Organization. Note that is is possible to have multiple Parties be associated with the same Organization.

The RegisterPartyRequest message must also contain a Party Profile element that describes the RegistryClient's half of the special TPA between the Registry and the RegistryClient. See Appendix AA.4 for details.

3.2.5 Registry Service Interface in TPA SPECIFICATION

The above specification of the Registry service interface is expressed in TPA SPECIFICATION [7] as follows[1]:

```
<BusinessInterface>
  <ServiceInterface InterfaceId = "Registry">
  <OrgName Partyname = "Registry">Registry</OrgName>
  <TaskName>RegisterPartyRequest</TaskName>
  <ActionMenu>
    <Action id = "registerParty" Type = "basic" Invocation = "asyncOnly">
      <Request>
        <RequestName>RegisterPartyRequest</RequestName>
        <RequestMessage>RegisterPartyRequest</RequestMessage>
      </Request>
      <Response Required = "yes">
        <ResponseName>RequestAcceptedResponse</ResponseName>
      </Response>
      <ExceptionResponse>
        <ExceptionResponseName>ebXMLError
      </ExceptionResponseName>
      </Response>
    </Action>
  </ActionMenu>
  </ServiceInterface>
</BusinessInterface>
```

3.2.6 RegistryClient Service Interface in TPA SPECIFICATION

The above specification of the RegistryClient service interface is expressed in TPA SPECIFICATION [7] as follows[2]:

```
<BusinessInterface>
  <ServiceInterface InterfaceId = "RegistryClient">
  <OrgName Partyname = "RegistryClient">RegistryClient</OrgName>
  <TaskName>RegisterPartyResponse</TaskName>
  <ActionMenu>
    <Action id = "registerPartyAccepted" Type = "basic" Invocation = "asyncOnly">
      <Request>
        <RequestName>RequestAcceptedResponse</RequestName>
        <RequestMessage> RequestAcceptedResponse </RequestMessage>
      </Request>
      <Response Required = "no"/>
    </Action>
    <Action id = "registerPartyError" Type = "basic" Invocation = "asyncOnly">
      <Request>
        <RequestName>ebXMLError</RequestName>
        <RequestMessage> ebXMLError</RequestMessage>
      </Request>
      <Response Required = "no"/>
    </Action>
  </ActionMenu>
```

[1] It should be noted that this XML fragment is tentative since the exact grammar of the ebXML TPA has not yet been defined.

[2] It should be noted that this XML fragment is tentative since the exact grammar of the ebXML TPA has not yet been defined.

```
  </ServiceInterface>
</BusinessInterface>
```

4 Object Management Service

[Note] The workflow envisioned in [2] is not addressed completely in this version. This chapter is a simplified sub-set of that workflow.

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 RegistryClient's to manage the life cycle of managed objects (e.g. documents) required for ebXML business processes. The Object Management Service can be used with all types of managed objects including the built-in managed objects specified in [3] such as Classification and Association.

Once an ebXML client of the Registry Services has successfully been through the Bootstrapping Registration Process, it is now capable of using the Object Management services of the Registry.

4.1 Life Cycle of a Managed Object

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

Figure 3 shows the typical life cycle of a managed object.

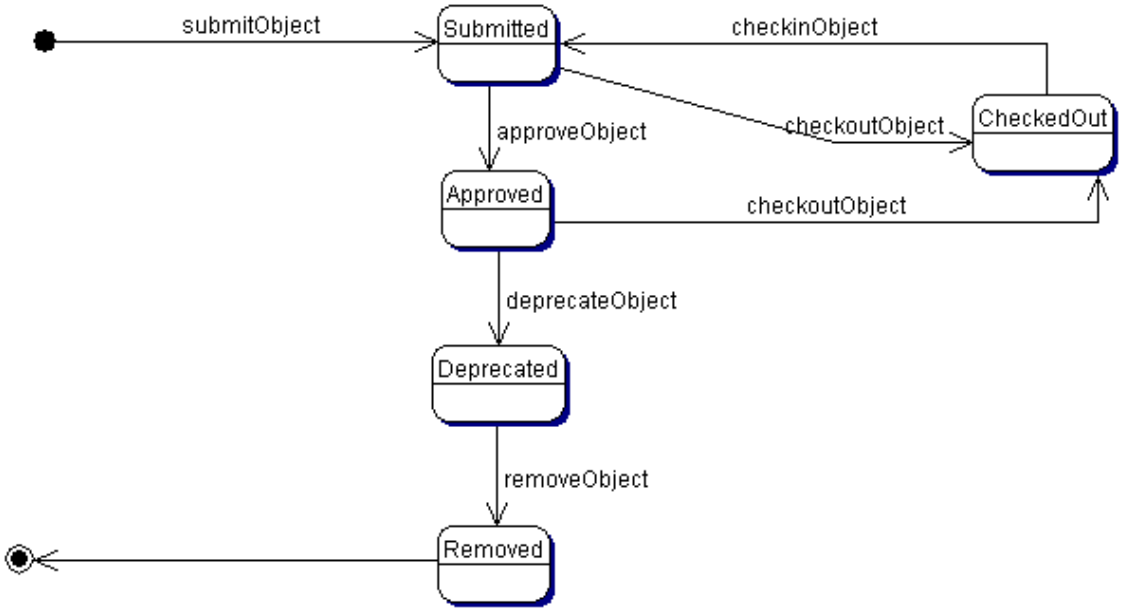


Figure 3: Life Cycle of a Managed Object

4.2 Object Attributes

A managed object is associated with a set of standard meta-data defined as attributes of the ManagedObject class described in [3]. These attributes reside outside of the actual object content and provide valuable meta-data about the managed object. An XML schema element called ManagedObject (See Appendix AA.3 for details.) is defined that encapsulates all object meta-data attributes defined in [3] as attributes of the schema element.

4.3 The Submit Objects Protocol

This section describes the protocol of the Registry Service that allows a RegistryClient to submit one or more managed objects in the repository using the *Object* Manager on behalf of a Submitting Organization. It is expressed in UML notation as described in section 3.1.

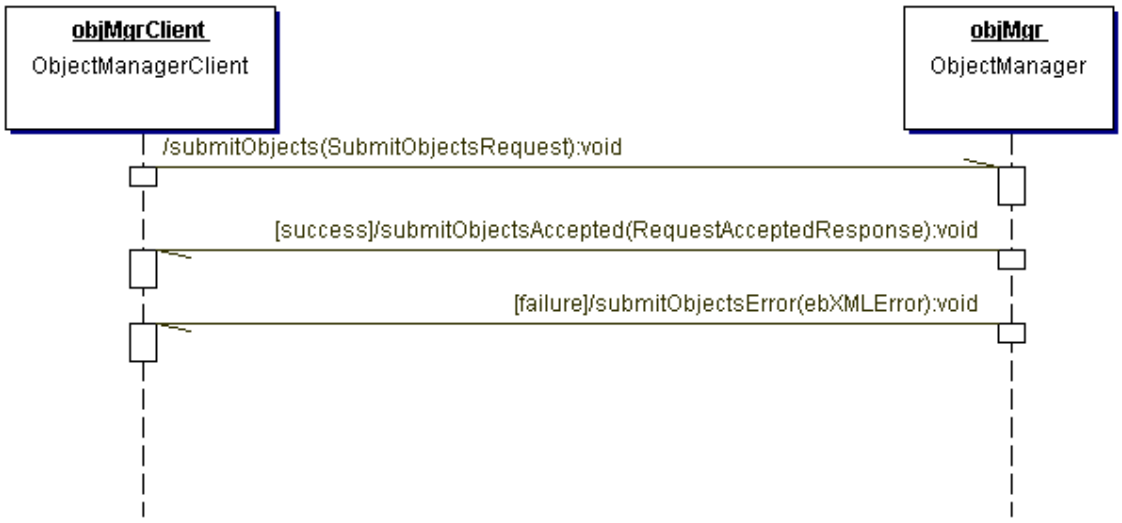


Figure 4: Submit Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix AA.5.

4.3.1 ManagedObject

The SubmitObjectRequest message includes a ManagedObject element which identifies the Submitting Party. It also includes 1 or more SubmittedObject elements.

Each SubmittedObject element specifies a ManagedObject element which provides standard meta-data about the object being submitted to the repository as defined by [3]. Note that these standard ManagedObject attributes are separate from the managed object itself, thus allowing the ebXML Repository to catalog arbitrary objects. In addition each SubmittedObject in the request may optionally specify any number of Classifications or Associations for the SubmittedObject.

In summary each managed object in the Repository is associated with a standard set of meta-data attributes collectively described by a ManagedObject element in XML. A simple URI reference to a ManagedObject is represented in XML as a ManagedObjectRef element.

4.4 The Approve Objects Request

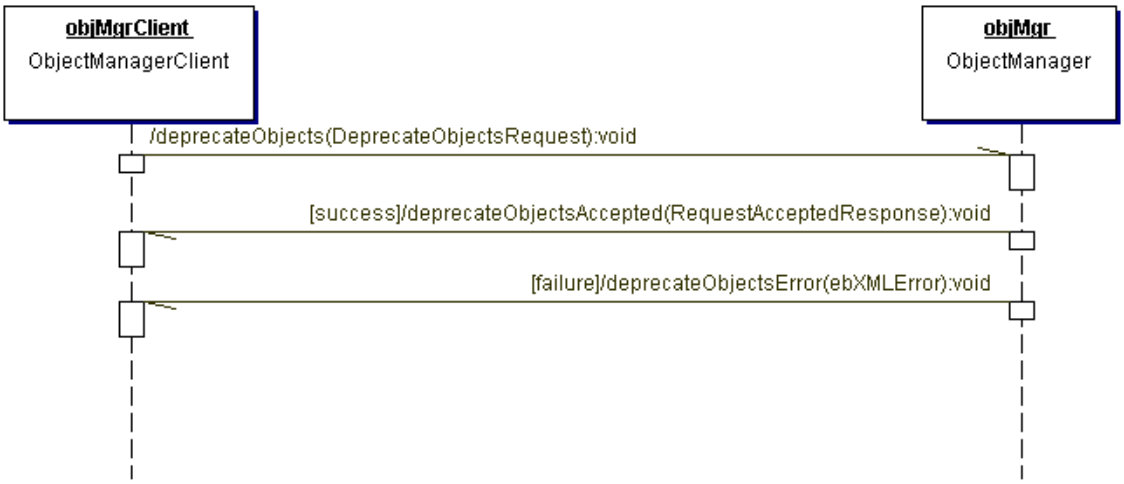
This section describes the protocol of the Registry Service that allows a client to approve one or more previously submitted managed objects using the Object Manager. Once a managed object is approved it will become available for use by business parties (e.g. during the assembly of new TPAs and Party Profiles). It is expressed in UML notation as described in section 3.1.

Figure 5: Approve Objects Sequence Diagram

For details on the schema for the business documents shown in this process refer to Appendix AA.6.

4.5 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 using the Object Manager. Once an object is deprecated, no *new* associations to that object can be submitted. In effect the managed object is marked as removed or deleted. However, existing references to a deprecated managed object continue to function normally. The deprecate object protocol is expressed in UML notation as described in section 3.1.

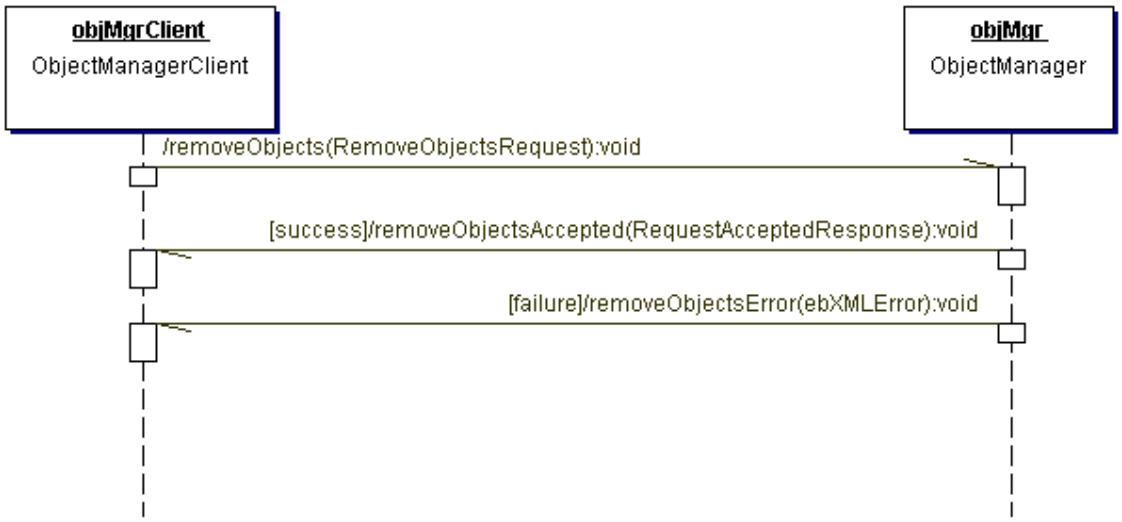


491 **Figure 6: Deprecate Objects Sequence Diagram**

492 For details on the schema for the business documents shown in this process refer to Appendix
493 AA.7.

494 4.6 The Remove Objects Request

495 This section describes the protocol of the Registry Service that allows a client to remove one or
496 more previously deprecated managed objects using the Object Manager. An object cannot be
497 removed as long as there exists 1 or more objects with associations to that object. Once an
498 object is removed it will be not be present at all in the Registry. The remove object protocol is
499 expressed in UML notation as described in section 3.1.



500

501 **Figure 7: Remove Objects Sequence Diagram**

502 For details on the schema for the business documents shown in this process refer to Appendix
503 AA.8.

504 5 Object Query Management Service

505 This section describes the capabilities of the Registry Service that allows a client
506 (ObjectQueryManagerClient) to search for (query) managed objects in the ebXML Repository
507 using the ObjectQueryManager *interface of the Registry*.

508 Due to the synchronous nature of queries all interactions between the
509 ObjectQueryManagerClient and the ObjectQueryManager are synchronous in nature as reflected
510 in the UML sequence diagrams that follow. Any errors in the query request messages are
511 indicated in the corresponding query response message.

512 5.1 Design Goals For Object Query Management Support

513 The following goal motivated the design of Object Query Management service:

514 1. Make it simple for Registry clients to query the registry for managed objects

2. Do not invent a new query language (a very large wheel to reinvent)
3. Make it simple for Registry providers to implement the Repository using a relational database
4. Support all query mechanisms described in [3]
5. Leverage security mechanism that will be specified in ebXML TRP rather than inventing a different model for the Registry

5.2 Browse and Drill Down Query Support

The browse and drill drown query style is completely supported by the following primitive interactions between the ObjectQueryManagerClient and the ObjectQueryManager.

5.2.1 Get Root Classification Nodes Request

An ObjectQueryManagerClient send this request to get a list of root ClassificationNodes (nodes with no parent) defined in the repository. 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. It is expressed in UML notation as described in section 3.1.

Figure 8: Get Root Classification Nodes Sequence Diagram

For details on the schema for the business documents shown in this process refer to A.10 and A.11.

5.2.2 Get Classification Tree Request

An ObjectQueryManagerClient send this request to get the ClassificationNode sub-tree defined in the repository under the ClassificationNode specified in the request. Note that a GetClassificationTreeRequest can specify an integer attribute called *depth* to get the sub-tree upto the specified depth. If depth is 1 (default) then only the immediate children of the specified ClassificationItemRef are returned. If depth is 0 then the entire sub-tree is retrieved.

It is expressed in UML notation as described in section 3.1.

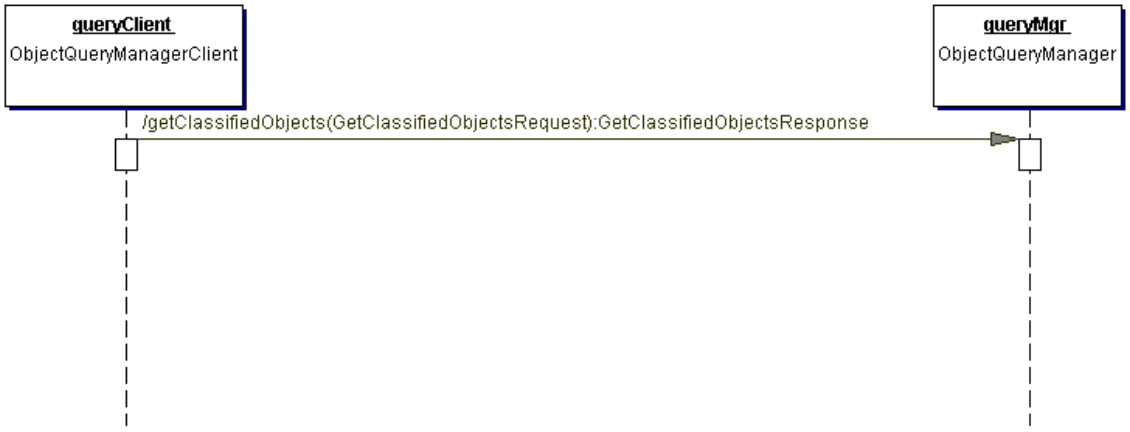
542

543 **Figure 9: Get Classification Tree Sequence Diagram**

544 For details on the schema for the business documents shown in this process refer to A.12 and
545 A.13.

546 5.2.3 Get Classified Objects Request

547 An ObjectQueryManagerClient send this request to get a list of references to managed objects
548 defined in the repository that are classified by the specified ClassificationNodes in the
549 ManagedObjectRefList in the request. Note that it is possible to get managed objects based on
550 matches with multiple classifications. It is expressed in UML notation as described in section 3.1.



551

552 **Figure 10: Get Classified Objects Sequence Diagram**

553 For details on the schema for the business documents shown in this process refer to A.14 and
554 A.15.

555 5.3 Ad Hoc Query Support

556 Details will be specified post Tokyo.

5.4 Keyword Search Based Query Support

The Registry provides a search engine functionality to search for managed objects that contain specified keywords in their content or attributes. Details will be specified post Tokyo.

5.5 Object Retrieval

The response messages that is returned by ObjectQueryManager contain zero or more ManagedObjectRef elements, one for each object that matched the query. Each ManagedObjectRef element contains a URI for the object matching the query in the repository. The client can use the URI to retrieve the object. Note that security issues will be addressed by ebXML TRP security mechanisms that are outside the scope of this document.

6 References

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

[2] Registry and Repository Part 1: Business Domain Model

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

[3] ebXML Repository Information Model

[4] ebXML Messaging Service Specification, Version 0.21,

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

[5] ebXML Business Process Meta-model

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

[6] ebXML Core Components Meta-model (unable to find reference despite best attempt)

[7] Trading-Partner Specification

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

[8] ebXML TRP Error Handling Specification

2 Acknowledgments

The members of the Registry and Repository team wish to acknowledge the support of the members of the various ebXML working groups who have contributed ideas and suggestions for this proposal.

Appendix A Schemas and DTD Definitions

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

[Note] The DTDs for messages should be considered as works-in-progress. They are open to change based on collaboration with and input from the team and various other working groups.

[Note] Several DTDs use a common boiler plate DTD ManagedObject.dtd that must be read in order to understand the main DTD. This common DTD is defined in Appendix AA.3

A.1 RequestAcceptedResponse Message DTD

```
<!ELEMENT RequestAcceptedResponse EMPTY>
<!ATTLIST RequestAcceptedResponse  xml:lang      NMTOKEN  #REQUIRED
                          interfaceId    CDATA    #REQUIRED
                          requestMessage CDATA    #REQUIRED
                          actionId       CDATA    #REQUIRED >
<!ENTITY % interfaceId "">
```

A.2 ebXMLError Message DTD

```
<!ELEMENT ebXMLError  (ErrorHeader , ErrorLocation* )>
<!ATTLIST ebXMLError  xml:lang NMTOKEN  #REQUIRED >
<!ELEMENT ErrorHeader  (Severity , ErrorCode , ErrorDesc? , MinRetrySecs? )>
<!ATTLIST ErrorHeader  ID NMTOKEN  #REQUIRED >
<!ELEMENT Severity  (#PCDATA )>

<!-- Either Warning, TransientError or HardError -->
<!ELEMENT ErrorCode  (#PCDATA )>

<!-- string max 14 char -->
<!ELEMENT ErrorDesc  (#PCDATA )>

<!-- string max 256 (?) char -->
<!ELEMENT MinRetrySecs  (#PCDATA )>

<!-- An integer -->
<!ELEMENT SwVendorErrorRef  (#PCDATA )>

<!-- string max 256 (?) chars -->
<!ELEMENT ErrorLocation  (RefToMessageId? ,  (Href | XMLDocumentErrorLocn ) )>
<!ATTLIST ErrorLocation  ID NMTOKEN  #REQUIRED >
<!ELEMENT RefToMessageId  (#PCDATA )>

<!ELEMENT Href  (#PCDATA )>

<!ELEMENT XMLDocumentErrorLocn  (DocumentId , Xpath )>

<!ELEMENT DocumentId  (#PCDATA )>

<!ELEMENT Xpath  (#PCDATA )>
```

A.3 ManagedObject DTD

```
<!ENTITY % ebXMLError SYSTEM "ebXMLError.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%ebXMLError;

<!ELEMENT ManagedObject EMPTY>

<!--
The following are standard document attributes that provide meta-data
about the document. These are based on the ebXML Repository Information Model
specification:
-->
<!ATTLIST ManagedObject guid          CDATA  #REQUIRED>

<!ATTLIST ManagedObject uri           CDATA  #REQUIRED>

<!ATTLIST ManagedObject type          (UserDefined |
                                Schema |
                                Process |
                                PartyProfile |
                                ServiceInterface |
                                BusinessService |
                                Role |
                                Transport |
                                Association |
                                ClassificationNode |
                                Classification )  #REQUIRED>

<!ATTLIST ManagedObject name          CDATA  #REQUIRED>

<!ATTLIST ManagedObject description    CDATA  #IMPLIED>

<!ATTLIST ManagedObject mimeType      CDATA  #IMPLIED>

<!ATTLIST ManagedObject majorVersion   CDATA  "0">

<!ATTLIST ManagedObject minorVersion   CDATA  "1">

<!ATTLIST ManagedObject registryStatus  (Submitted | Approved | Deployed | Deprecated )
"Submitted">

<!ELEMENT ManagedObjectRef EMPTY>
<!ATTLIST ManagedObjectRef  guid CDATA  #REQUIRED
                      uri  CDATA  #IMPLIED
                      name CDATA  #IMPLIED >
<!ELEMENT ManagedObjectRefList  (ManagedObjectRef )*>

<!ELEMENT ExternalObject EMPTY>
<!ATTLIST ExternalObject  guid       CDATA  #REQUIRED
                    uri        CDATA  #IMPLIED
                    description CDATA  #IMPLIED >
<!ELEMENT ExternalObjectList  (ExternalObject )*>
```

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

The first ManagedObjectRef is ref to a ManagedObject being classified
The second ManagedObjectRef is a ref to ClassificationNode

The first ManagedObjectRef is optional when Classification is defined part of
a SubmittedObject.
-->
<!ELEMENT Classification  (ManagedObjectRef? , ManagedObjectRef )>

<!ELEMENT ClassificationList  (Classification )*>

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

The first ManagedObjectRef is ref to the "from" ManagedObject in association
The first ManagedObjectRef is ref to the "to" ManagedObject in association

The first ManagedObjectRef is optional when Classification is defined part of
a SubmittedObject.
-->
<!ELEMENT Association  (ManagedObjectRef? , ManagedObjectRef )>
<!ATTLIST Association  fromLabel   CDATA  #IMPLIED
                      toLabel     CDATA  #IMPLIED
                      type        CDATA  #IMPLIED
                      bidirection CDATA  #IMPLIED >
<!ELEMENT AssociationList  (Association )*>
```

A.4 RegisterPartyRequest Message DTD

The RegisterPartyRequest Message includes a Party Profile specified by a TPA element, which conforms to the DTD specified for TPA SPECIFICATION in [7].

```
<!-- Pulls in the module at this spot in my DTD: -->
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

%managedObject;

<!ENTITY % tpa SYSTEM "tpa_1_0_6.dtd">

%tpa;

<!--
The Organization element needs to be defined by CC team.
For now this is a place holder.
-->
<!ELEMENT Organization EMPTY>

<!--
The party must be associated with an Organization. The Organization
may be defined in the request or may refer to a previously defined
Organization referred to by ManagedObjectRef.

The party must provide a PartyProfile which currently is represented
by a TPA. TP team needs to define PartyProfile post Tokyo.
-->
<!ELEMENT RegisterPartyRequest  ( (Organization | ManagedObjectRef ) , TPA )>
```

A.5 SubmitObjectsRequest Message DTD

```
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%managedObject;

<!--
The ManagedObjectRef must be a ref to a previously registered Party
which is the Submitting party

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  (ManagedObjectRef , SubmittedObject+ )>

<!--
The ManagedObject provides meta data about the object being submitted

ClassificationList can be optionaly be specified to define Classifications
for the SubmittedObject

AssociationList can be optionaly be specified to define Associations
for the SubmittedObject

The ExternalObjectList provides zero or more external objects related to
```

```
the object being submitted.

-->
<!ELEMENT SubmittedObject  (ManagedObject , ClassificationList?,
      AssociationList?, ExternalObjectList? )>
```

A.6 ApproveObjectsRequest Message DTD

```
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%managedObject;

<!--
The ManagedObjectRef is to a previously registered Party
which is the approving party.

The ManagedObjectRefList is the list of
refs to the managed objects being approved.
-->

<!ELEMENT ApproveObjectsRequest  (ManagedObjectRef , ManagedObjectRefList )>
```

A.7 DeprecateObjectsRequest Message DTD

```
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%managedObject;

<!--
The ManagedObjectRef is to a previously registered Party
which is the deprecating party.

The ManagedObjectRefList is the list of
refs to the managed objects being deprecated.
-->
<!ELEMENT DeprecateObjectsRequest  (ManagedObjectRef , ManagedObjectRefList )>
```

A.8 RemoveObjectsRequest Message DTD

```
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%managedObject;

<!--
The ManagedObjectRef is to a previously registered Party
which is the removing party.

The ManagedObjectRefList is the list of
refs to the managed objects being removed
```

```
-->
<!ELEMENT RemoveObjectsRequest  (ManagedObjectRef , ManagedObjectRefList )>
```

A.9 ClassificationNode DTD

This DTD is used to submit ClassificationNodes. It is capable of submitting a single node or an entire sub-tree.

```
<!ENTITY % managedObject SYSTEM "ManagedObject.dtd">

<!-- Pulls in the module at this spot in my DTD: -->
%managedObject;

<!--
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  (ClassificationNode* )>

<!--
The name of the ClassificationNode. Maps to the name attribute
of the ManagedObject meta data class.
-->
<!ATTLIST ClassificationNode name CDATA  #REQUIRED>

<!--
ClassificationNodeRef is used by the ObjectQueryManager
for various query responses. It represents a tree of
ClassificationNodeRef.
-->
<!ELEMENT ClassificationNodeRef  (ManagedObjectRef , ClassificationNodeRef* )>

<!--
ClassificationNodeRefList is used to send a list of
ClassificationNodeRef when returning immediate children nodes
of a node.
-->
<!ELEMENT ClassificationNodeRefList  (ClassificationNodeRef )*>
```

A.10 GetRootClassificationNodesRequest Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
The query request that gets the specified root ClassificationNodes
-->
<!ELEMENT GetRootClassificationNodesRequest EMPTY>

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

A.11 GetRootClassificationNodesResponse Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
The response includes a ManagedObjectRefList which has zero or more
references to ManagedObjects that represent ClassificationSchemeRefs
-->
<!ELEMENT GetRootClassificationNodesResponse  (ManagedObjectRefList | ebXMLError )>
```

A.12 GetClassificationTreeRequest Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
Get the ClassificationItemTreeRef under the ClassificationItemRef specified
by ManagedObjectRef.
-->
<!ELEMENT GetClassificationTreeRequest  (ManagedObjectRef )>

<!--
If depth is 1 just fetch immediate child
nodes, otherwise fetch the descendant tree upto specified depth level.
If depth is 0 that implies fetch entire sub-tree
-->
<!ATTLIST GetClassificationTreeRequest  depth CDATA '1' >
```

A.13 GetClassificationTreeResponse Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
The response includes a ClassificationNodeRefList which includes only
immediate ClassificationNodeRef children nodes if depth attribute in
GetClassificationTreeRequest was 1, otherwise the decendent nodes
upto specifed depth level are returned.
-->
<!ELEMENT GetClassificationTreeResponse  (ClassificationNodeRefList | ebXMLError )>
```

A.14 GetClassifiedObjectsRequest Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
Get refs to all managed objects that are classified by all the
ClassificationNodeRef specified by ManagedObjectRefList.
```

```
Note this is an implicit logical AND operation
-->
<!ELEMENT GetClassifiedObjectsRequest  (ManagedObjectRefList )>

<!--
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 ManagedObject.
The default value of '*' implies that client is interested in all types
of managed objects that are classified by the specified ClassificationNode.
-->
<!ATTLIST GetClassifiedObjectsRequest objectType CDATA  "*">
```

A.15 GetClassifiedObjectsResponse Message DTD

```
<!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">

%classificationNode;

<!--
The response includes a ManagedObjectRefList which has zero or more
references to ManagedObjects that are classified by the ClassificationNodeRef
specified in the GetClassifiedObjectsRequest.
-->
<!ELEMENT GetClassifiedObjectsResponse  (ManagedObjectRefList | ebXMLError )>
```

Appendix B TPA Between Registry Client And Registry

Appendix C Example XML for Submitting a Classification Tree

The following XML message submits the classification tree for the classification tree example in [3].

```
<?xml version = "1.0"?>
<!DOCTYPE ClassificationNode SYSTEM "ClassificationNode.dtd">

<!--
This is a sample XML that 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.
-->
<ClassificationNode name = "Industry">
    <ClassificationNode name = "Automotive">
        <ClassificationNode name = "Geography">
            <ClassificationNode name = "US"/>
            <ClassificationNode name = "Europe"/>
```

```
            </ClassificationNode>
        </ClassificationNode>
        <ClassificationNode name = "Health Care"/>
        <ClassificationNode name = "Retail"/>
/ClassificationNode>
```

Appendix D Terminology Mapping

While every attempt has been made to use the same terminology used in previous 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”	Registered Object	
ManagedObject	Registry Item	Administered Component
ExternalObject	Related Data	N/A
Object.guid	RaitemId	
Object.uri	ObjectLocation	
ManagedObject.type	DefnSource, PrimaryClass, SubClass	
Object.name	CommonName	
ManagedObject.description	Description	
ManagedObject.mimeType	MimeType	
ManagedObject.majorVersion	partially to Version	
ManagedObject.minorVersion	partially to Version	
ManagedObject.registryStatus	RegStatus	

Table 1: Terminology Mapping Table

Appendix E Open Issues

The following are some open issues that will be evaluated based on further investigation and within the context of broader team discussions:

- Do we need the ObjectQueryManagerClient interface? It is currently an empty place-holder because all query manager interactions are synchronous. Reason to keep it is to allow for future support of asynchronous responses. Should we close this issue?
- Should Library Control Service be broken out into a separate service from the Object Manager (SH). . This is a post Tokyo issue.

- Add more comprehensive and common error and warning messages throughout. Consider adding error responses such as "Not Authenticated," and "Registry Not Available," etc. Align these messages with Technical Architecture WG. This is a post Tokyo issue.
- Why are we requiring that an "Authorization Mechanism" be accomplished as a separate (internet-based) external call? (SH) Why can't this approval process be modeled as an Internal Implementation detail?
- When someone tries to request a deprecated object, they should be notified that the object has been deprecated. (LG) "SO" can deprecate objects but where is the Notification processing related to others who will attempt to associate with those deprecate objects? This is a post Tokyo issue.
- When an object is deprecated and/or then removed, what do we maintain within the Registry/Repository? (LG) When an object is removed, the Meta Data for the object should be retained and marked as "Removed." Farrukh's opinion is that this is exactly what Deprecation was designed to do. It is essentially a way to mark an object as logically deleted so future use (references) are prevented while existing uses would still work. Should we close this issue?
- Remove the requirement that RS must be accessible over ebXML Messaging Service (SH)
- Inconsistency noted by DW that objects retrieval is not done over ebXML Messaging Service while all other interactions are. This is a post Tokyo issue.