1

OASIS/ebXML Registry Services Specification
v2.0 -Approved Committee Specification

OASIS/ebXML Registry Technical Committee

6 December 2001

2   *This page intentionally left blank.*

3 # 1   Status of this Document

4   Distribution of this document is unlimited.

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

6   ***This version:***
7   http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf
8   ***Latest version:***
9   http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf

2 OASIS/ebXML Registry Technical Committee

10

11 The OASIS/ebXML Registry Technical Committee has approved this document in its current
12 form. At the time of this approval the following were members of the OASIS/ebXML Registry
13 Technical Committee.

14 Kathryn Breininger, Boeing
15 Lisa Carnahan, NIST
16 Joseph M. Chiusano, LMI
17 Suresh Damodaran, Sterling Commerce
18 Mike DeNicola, Fujitsu
19 Anne Fischer, Drummond Group, Inc.
20 Sally Fuger, AIAG
21 Jong Kim, InnoDigital
22 Kyu-Chul Lee, Chungnam National University
23 Joel Munter, Intel
24 Farrukh Najmi, Sun Microsystems
25 Joel Neu, Vitria Technologies
26 Sanjay Patil, IONA
27 Neal Smith, Chevron
28 Nikola Stojanovic, Encoda Systems, Inc.
29 Prasad Yendluri, webmethods
30 Yutaka Yoshida, Sun Microsystems

2.1 Contributors

31

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

34 Len Gallagher, NIST
35 Sekhar Vajjhala, Sun Microsystems

Table of Contents

203 ## Table of Figures

231

232   ## Table of Tables

244

3 Introduction

3.1 Summary of Contents of Document

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

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

3.2 General Conventions

The following conventions are used throughout this document:

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

The term "repository item" is used to refer to an object that has resides in a repository for storage and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the Registry by a RegistryObject instance.

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

Capitalized Italic words are defined in the ebXML Glossary.

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

3.3 Audience

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

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

Related Documents

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

    a) *ebXML Registry Information Model* [ebRIM]
    b) *ebXML Message Service Specification* [ebMS]
    c) *ebXML Business Process Specification Schema* [ebBPSS]
    d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

275 4   Design Objectives

276 4.1 Goals

277 The goals of this version of the specification are to:

278 • Communicate functionality of Registry services to software developers

279 • Specify the interface for Registry clients and the Registry

280 • Provide a basis for future support of more complete ebXML Registry requirements

281 • Be compatible with other ebXML specifications

282 4.2 Caveats and Assumptions

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

286 Interoperability requirements dictate that at least one of the normative interfaces as referenced in
287 this specification must be supported.

288   1. All access to the Registry content is exposed via the interfaces defined for the Registry
289   Services.
290   2. The Registry makes use of a Repository for storing and retrieving persistent information
291   required by the Registry Services. This is an implementation detail that will not be
292   discussed further in this specification.

5   System Overview

5.1   What The ebXML Registry Does

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

5.2   How The ebXML Registry Works

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

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

5.2.1   Schema Documents Are Submitted

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

5.2.2   Business Process Documents Are Submitted

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

5.2.3   Seller's Collaboration Protocol Profile Is Submitted

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

5.2.4   Buyer Discovers The Seller

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

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

328   5.2.5 CPA Is Established

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

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

335 5.3 Registry Users

336   We describe the actors who use the registry from the point of view of security and analyze the
337   security concerns of the registry below. This analysis leads up to the security requirements for
338   version 2.0. Some of the actors are defined in Section 9.7. Note that the same entity may
339   represent different actors. For example, a Registration Authority and Registry Administrator may
340   have the same identity.

341   **Table 1: Registry Users**

Actor	Function	ISO/IEC 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY have the same identity as Registration Authority
Registered User	Has a contract with the Registration Authority and MUST be authenticated by Registration Authority.		The contract could be a ebXML CPA or some other form of contract.
Registry Guest	Has no contract with Registration Authority. Does not have to be authenticated for Registry access. Cannot change contents of the Registry (MAY be permitted to read some RegistryObjects.)		Note that a Registry Guest is not a Registry Reader.
Submitting Organization	A Registered User who does lifecycle operations on permitted RegistryObjects.	Submitting Organization (SO)	
Registry Reader	A Registered User who has only *read* access		
Responsible Organization	Creates Registry Objects	Responsible Organization	RO MAY have the same identity as SO

		(RO)	
Registry Client	Registered User or Registered Guest		
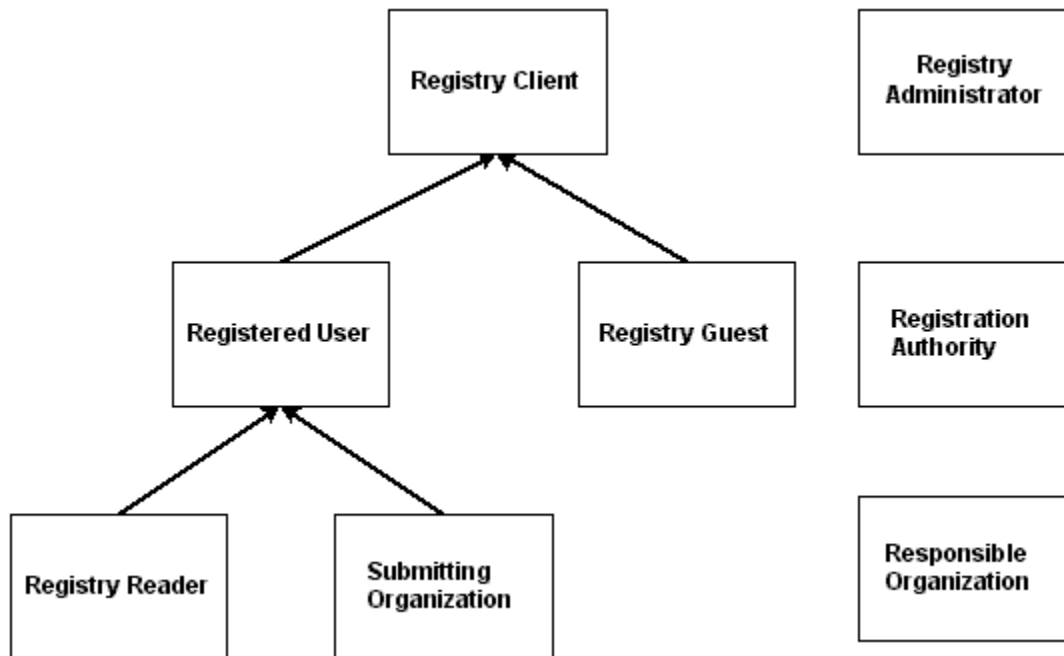
342
343   **Figure 1: Actor Relationships**

344   Note:

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

346   A Submitting Organization and a Responsible Organization are the same.

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

348   A Registry Administrator and Registration Authority are the same.

349   ## 5.4 Where the Registry Services May Be Implemented

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

352   ## 5.5 Implementation Conformance

353   An implementation is a *conforming* ebXML Registry if the implementation meets the conditions
354   in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the
355   implementation meets the conditions in Section 5.5.2. An implementation is a conforming
356   ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to
357   the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming
358   ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and
359   Registry Client.

5.5.1 Conformance as an ebXML Registry

360

361 An implementation conforms to this specification as an ebXML Registry if it meets the
362 following conditions:

363 1. Conforms to the ebXML Registry Information Model [ebRIM].

364 2. Supports the syntax and semantics of the Registry Interfaces and Security Model.

365 3. Supports the defined ebXML Registry Schema (Appendix B).

366 4. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

5.5.2 Conformance as an ebXML Registry Client

367

368 An implementation conforms to this specification, as an ebXML Registry Client if it meets the
369 following conditions:

370 1. Supports the ebXML CPA and bootstrapping process.

371 2. Supports the syntax and the semantics of the Registry Client Interfaces.

372 3. Supports the defined ebXML Error Message DTD.

373 4. Supports the defined ebXML Registry Schema (Appendix B).

374

375 # 6   ebXML Registry Architecture

376   The ebXML Registry architecture consists of an ebXML Registry Service and ebXML Registry
377   Clients.   The ebXML Registry Service provides the methods for managing a repository. An
378   ebXML Registry Client is an application used to access the Registry.
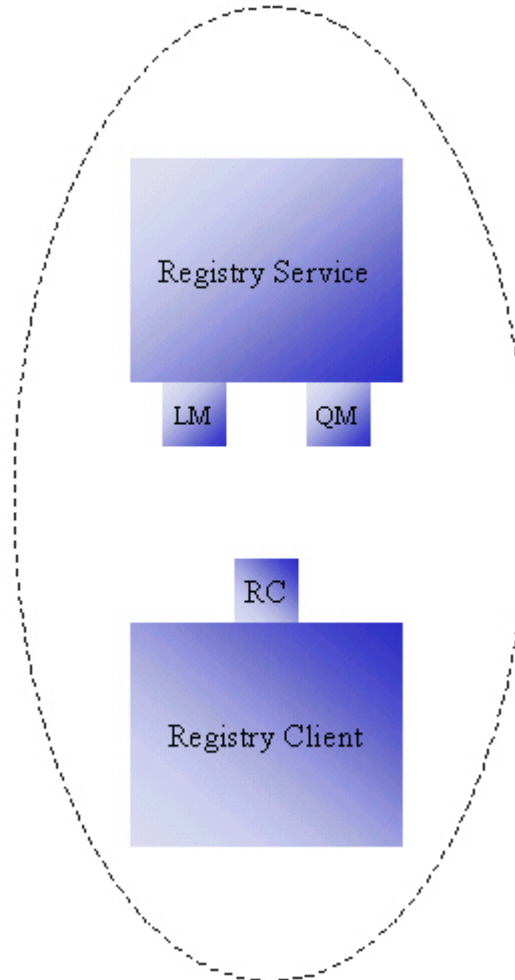
379
380   **Figure 2: ebXML Registry Service Architecture**

381 ## 6.1   Registry Service Described

382   The ebXML Registry Service is comprised of a robust set of interfaces designed to
383   fundamentally manage the objects and inquiries associated with the ebXML Registry. The two
384   primary interfaces for the Registry Service consist of:

385   • A Life Cycle Management interface that provides a collection of methods for managing
386   objects within the Registry.

387   • A Query Management Interface that controls the discovery and retrieval of information from
388   the Registry.

389   A registry client program utilizes the services of the registry by invoking methods on one of the
390   above interfaces defined by the Registry Service. This specification defines the interfaces
391   exposed by the Registry Service (Sections 6.4 and 6.5) as well as the interface for the Registry
392   Client (Section 6.6).

393 ## 6.2 Abstract Registry Service

394 The architecture defines the ebXML Registry as an abstract registry service that is defined as:

395 1. A set of interfaces that must be supported by the registry.

396 2. The set of methods that must be supported by each interface.

397 3. The parameters and responses that must be supported by each method.

398 The abstract registry service neither defines any specific implementation for the ebXML
399 Registry, nor does it specify any specific protocols used by the registry. Such implementation
400 details are described by concrete registry services that realize the abstract registry service.

401 The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide
402 two key functional interfaces called **QueryManager**[1] (QM) and **LifeCycleManager**[2]
403 (LM).

404
405 **Figure 3: The Abstract ebXML Registry Service**

406 Appendix A provides hyperlinks to the abstract service definition in the Web Service Description
407 Language (WSDL) syntax.

408 ## 6.3 Concrete Registry Services

409 The architecture allows the abstract registry service to be mapped to one or more concrete
410 registry services defined as:

411 • Implementations of the interfaces defined by the abstract registry service.

412 • Bindings of these concrete interfaces to specific communication protocols.

413 This specification describes two concrete bindings for the abstract registry service:

414 • A SOAP binding using the HTTP protocol

415 • An ebXML Messaging Service (ebMS) binding

416 A registry may implement one or both of the concrete bindings for the abstract registry service as
417 shown in Figure 4.

418

---

[1] Known as ObjectQueryManager in V1.0

[2] Known as ObjectManager in V1.0

419
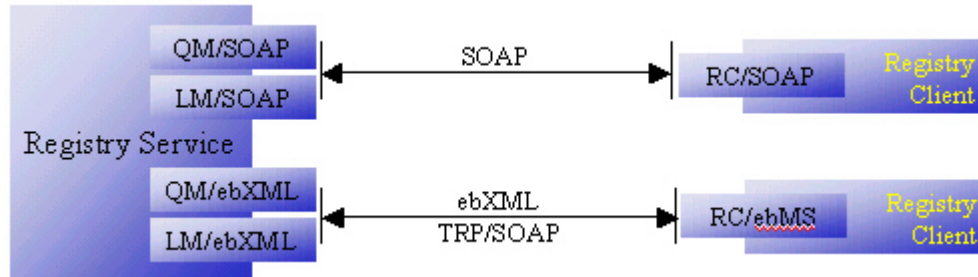
420   **Figure 4: A Concrete ebXML Registry Service**

421   Figure 4 shows a concrete implementation of the abstract ebXML Registry (RegistryService) on
422   the left side. The RegistryService provides the QueryManager and LifeCycleManager interfaces
423   available with multiple protocol bindings (SOAP and ebMS).

424   Figure 4 also shows two different clients of the ebXML Registry on the right side. The top client
425   uses SOAP interface to access the registry while the lower client uses ebMS interface. Clients
426   use the appropriate concrete interface within the RegistryService service based upon their
427   protocol preference.

428   ### 6.3.1 SOAP Binding

429   #### 6.3.1.1   WSDL Terminology Primer

430   This section provides a brief introduction to Web Service Description Language (WSDL) since
431   the SOAP binding is described using WSDL syntax. WSDL provides the ability to describe a
432   web service in abstract as well as with concrete bindings to specific protocols. In WSDL, an
433   abstract service consists of one or more `port types` or end-points. Each port type consists
434   of a collection of `operations`. Each operation is defined in terms of `messages` that define
435   what data is exchanged as part of that operation. Each message is typically defined in terms of
436   elements within an XML Schema definition.

437   An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract
438   service may be used to define a concrete service by binding it to a specific protocol. This binding
439   is done by providing a `binding` definition for each abstract port type that defines additional
440   protocols specific details. Finally, a concrete `service` definition is defined as a collection of
441   `ports`, where each port simply adds address information such as a URL for each concrete port.

442   #### 6.3.1.2   Concrete Binding for SOAP

443   This section assumes that the reader is somewhat familiar with SOAP and WSDL. The SOAP
444   binding to the ebXML Registry is defined as a web service description in WSDL as follows:

445   • A single service element with name "RegistryService" defines the concrete SOAP binding
446   for the registry service.

447   • The service element includes two port definitions, where each port corresponds with one of
448   the interfaces defined for the abstract registry service. Each port includes an HTTP URL for
449   accessing that port.

450   • Each port definition also references a binding element, one for each interface defined in the
451   WSDL for the abstract registry service.
452
453   ```
<service name = "RegistryService">
```

```
454         <port name = "QueryManagerSOAPBinding" binding = "tns:QueryManagerSOAPBinding">
455             <soap:address location = "http://your_URL_to_your_QueryManager"/>
456         </port>
457
458         <port name = "LifeCycleManagerSOAPBinding" binding = "tns:LifeCycleManagerSOAPBinding">
459             <soap:address location = "http://your_URL_to_your_QueryManager"/>
460         </port>
461     </service>
462
```

463   The complete WSDL description for the SOAP binding can be obtained via a hyperlink in
464   Appendix A.

6.3.2   ebXML Message Service Binding

6.3.2.1   Service and Action Elements

467   When using the ebXML Messaging Services Specification, ebXML Registry Service elements
468   correspond to Messaging Service elements as follows:

469   • The value of the Service element in the MessageHeader is an ebXML Registry Service
470   interface name (e.g., "LifeCycleManager"). The type attribute of the Service element should
471   have a value of "ebXMLRegistry".

472   • The value of the Action element in the MessageHeader is an ebXML Registry Service
473   method name (e.g., "submitObjects").
474

```
475   <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>

476   <eb:Action>submitObjects</eb:Action>
477
```

478   Note that the above allows the Registry Client only one interface/method pair per message. This
479   implies that a Registry Client can only invoke one method on a specified interface for a given
480   request to a registry.

6.3.2.2   Synchronous and Asynchronous Responses

482   All methods on interfaces exposed by the registry return a response message.

Asynchronous response

484   When a message is sent asynchronously, the Registry will return two response messages. The
485   first message will be an immediate response to the request and does not reflect the actual
486   response for the request. This message will contain:

487   • MessageHeader;

488   • RegistryResponse element with empty content (e.g., **NO** AdHocQueryResponse);

489   – status attribute with value **Unavailable**.

490   The Registry delivers the actual Registry response element with non-empty content
491   asynchronously at a later time. The delivery is accomplished by the Registry invoking the
492   onResponse method on the RegistryClient interface as implemented by the registry client
493   application. The onResponse method includes a RegistryResponse element as shown below:

494   • MessageHeader;

495   • RegistryResponse element including;

496   – Status attribute (Success, Failure);

497   –   Optional RegistryErrorList.

498   **Synchronous response**

499   When a message is sent synchronously, the Message Service Handler will hold open the
500   communication mechanism until the Registry returns a response. This message will contain:

501   •   MessageHeader;

502   •   RegistryResponse element including;

503   –   Status attribute (Success, Failure);

504   –   Optional RegistryErrorList.

6.3.2.3   ebXML Registry Collaboration Profiles and Agreements

506   The ebXML CPP specification [ebCPP] defines a Collaboration-Protocol Profile (CPP) and a
507   Collaboration-Protocol Agreement (CPA) as mechanisms for two parties to share information
508   regarding their respective business processes. That specification assumes that a CPA has been
509   agreed to by both parties in order for them to engage in B2B interactions.

510   This specification does not mandate the use of a CPA between the Registry and the Registry
511   Client. However if the Registry does not use a CPP, the Registry shall provide an alternate
512   mechanism for the Registry Client to discover the services and other information provided by a
513   CPP. This alternate mechanism could be a simple URL.

514   The CPA between clients and the Registry should describe the interfaces that the Registry and
515   the client expose to each other for Registry-specific interactions. The definition of the Registry
516   CPP template and a Registry Client CPP template are beyond the scope of this document.

6.4 LifeCycleManager Interface

518   This is the interface exposed by the Registry Service that implements the object life cycle
519   management functionality of the Registry. Its methods are invoked by the Registry Client. For
520   example, the client may use this interface to submit objects, to classify and associate objects and
521   to deprecate and remove objects. For this specification the semantic meaning of submit, classify,
522   associate, deprecate and remove is found in [ebRIM].

523   **Table 2: LifeCycle Manager Summary**

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

	Updates one or more previously submitted objects.
RegistryResponse	**addSlots**(AddSlotsRequest req) Add slots to one or more registry entries.
RegistryResponse	**removeSlots**(RemoveSlotsRequest req) Remove specified slots from one or more registry entries.

6.5 QueryManager Interface

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

Table 3: Query Manager

Method Summary of QueryManager	
RegistryResponse	**submitAdhocQuery**(AdhocQueryRequest req) Submit an ad hoc query request.

6.6 Registry Clients

6.6.1 Registry Client Described

The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts the two possible topologies supported by the registry architecture with respect to the Registry and Registry Clients. The picture on the left side shows the scenario where the Registry provides a web based "thin client" application for accessing the Registry that is available to the user using a common web browser. In this scenario the Registry Client interfaces reside across the Internet and are local to the Registry from the user's view. The picture on the right side shows the scenario where the user is using a "fat client" Registry Browser application to access the registry. In this scenario the Registry Client interfaces reside within the Registry Browser tool and are local to the Registry from the user's view. The Registry Client interfaces communicate with the Registry over the Internet in this scenario.

A third topology made possible by the registry architecture is where the Registry Client interfaces reside in a server side business component such as a Purchasing business component. In this topology there may be no direct user interface or user intervention involved. Instead, the Purchasing business component may access the Registry in an automated manner to select possible sellers or service providers based on current business needs.

546
547   **Figure 5: Registry Architecture Supports Flexible Topologies**

548 ## 6.6.2 Registry Communication Bootstrapping

549 Before a client can access the services of a Registry, there must be some communication
550 bootstrapping between the client and the registry. The most essential aspect of this bootstrapping
551 process is for the client to discover addressing information (e.g. an HTTP URL) to each of the
552 concrete service interfaces of the Registry. The client may obtain the addressing information by
553 discovering the ebXML Registry in a public registry such as UDDI or within another ebXML
554 Registry.

555 • In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available
556   in a WSDL desription for the registry. This WSDL conforms to the template WSDL
557   description in Appendix A.1. This WSDL description may be discovered in a public registry
558   such as UDDI.

559 • In case of ebMS binding, the information exchange between the client and the registry may
560   be accomplished in a registry specific manner, which may involve establishing a CPA
561   between the client and the registry. Once the information exchange has occurred the Registry
562   and the client will have addressing information (e.g. URLs) for the other party.

563 ### 6.6.2.1 Communication Bootstrapping for SOAP Binding

564 Each ebXML Registry must provide a WSDL description for its RegistryService as defined by
565 Appendix A.1. A client uses the WSDL description to determine the address information of the
566 RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the
567 RegistryService may be accessed via a URL specified in the WSDL for the registry.

568 The use of WSDL enables the client to use automated tools such as a WSDL compiler to
569 generate stubs that provide access to the registry in a language specific manner.

570   At minimum, any client may access the registry over SOAP/HTTP using the address information
571   within the WSDL, with minimal infrastructure requirements other than the ability to make
572   synchronous SOAP call to the SOAP based ports on the RegistryService.

6.6.2.2   Communication Bootstrapping for ebXML Message Service

574   Since there is no previously established CPA between the Registry and the RegistryClient, the
575   client must know at least one Transport-specific communication address for the Registry. This
576   communication address is typically a URL to the Registry, although it could be some other type
577   of address such as an email address. For example, if the communication used by the Registry is
578   HTTP, then the communication address is a URL. In this example, the client uses the Registry's
579   public URL to create an implicit CPA with the Registry. When the client sends a request to the
580   Registry, it provides a URL to itself. The Registry uses the client's URL to form its version of an
581   implicit CPA with the client. At this point a session is established within the Registry. For the
582   duration of the client's session with the Registry, messages may be exchanged bidirectionally as
583   required by the interaction protocols defined in this specification.

6.6.3   RegistryClient Interface

585   This is the principal interface implemented by a Registry client. The client provides this interface
586   when creating a connection to the Registry. It provides the methods that are used by the Registry
587   to deliver asynchronous responses to the client. Note that a client need not provide a
588   RegistryClient interface if the [CPA] between the client and the registry does not support
589   asynchronous responses.

590   The registry sends all asynchronous responses to operations via the onResponse method.

591   **Table 4: RegistryClient Summary**

Method Summary of RegistryClient	
void	**onResponse**(RegistryResponse resp) Notifies client of the response sent by registry to previously submitted request.

6.6.4   Registry Response

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

6.7   Interoperability Requirements

6.7.1   Client Interoperability

597   The architecture requires that any ebXML compliant registry client can access any ebXML
598   compliant registry service in an interoperable manner. An ebXML Registry may implement any
599   number of protocol bindings from the set of normative bindings (currently ebXML TRP and
600   SOAP/HTTP) defined in this proposal. The support of additional protocol bindings is optional.

601 ## 6.7.2 Inter-Registry Cooperation

602 This version of the specification does not preclude ebXML Registries from cooperating with
603 each other to share information, nor does it preclude owners of ebXML Registries from
604 registering their ebXML registries with other registry systems, catalogs, or directories.

605 Examples include:

606 • An ebXML Registry that serves as a registry of ebXML Registries.

607 • A non-ebXML Registry that serves as a registry of ebXML Registries.

608 • Cooperative ebXML Registries, where multiple ebXML registries register with each other in
609 order to form a federation.

7 Life Cycle Management Service

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

The minimum-security policy for an ebXML registry is to accept content from any client if a
certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the
content.

7.1 Life Cycle of a Repository Item

The main purpose of the LifeCycleManagement service is to manage the life cycle of repository
items. Figure 6 shows the typical life cycle of a repository item. Note that the current version of
this specification does not support Object versioning. Object versioning will be added in a future
version of this specification

Figure 6: Life Cycle of a Repository Item

7.2 RegistryObject Attributes

A repository item is associated with a set of standard metadata defined as attributes of the
RegistryObject class and its sub-classes as described in [ebRIM]. These attributes reside outside
of the actual repository item and catalog descriptive information about the repository item. XML
elements called ExtrinsicObject and other elements (See Appendix B.1 for details) encapsulate
all object metadata attributes defined in [ebRIM] as XML attributes.

633 ## 7.3 The Submit Objects Protocol

634 This section describes the protocol of the Registry Service that allows a RegistryClient to submit
635 one or more repository items to the repository using the LifeCycleManager on behalf of a
636 Submitting Organization. It is expressed in UML notation as described in Appendix C.

637
638 **Figure 7: Submit Objects Sequence Diagram**

639 For details on the schema for the Business documents shown in this process refer to Appendix B.

640 The SubmitObjectRequest message includes a LeafRegistryObjectList element.

641 The LeafRegistryObjectList element specifies one or more ExtrinsicObjects or other
642 RegistryEntries such as Classifications, Associations, ExternalLinks, or Packages.

643 An ExtrinsicObject element provides required metadata about the content being submitted to the
644 Registry as defined by [ebRIM]. Note that these standard ExtrinsicObject attributes are separate
645 from the repository item itself, thus allowing the ebXML Registry to catalog objects of any
646 object type.

647 ### 7.3.1 Universally Unique ID Generation

648 As specified by [ebRIM], all objects in the registry have a unique id. The id must be a
649 Universally Unique Identifier (UUID) and must conform to the to the format of a URN that
650 specifies a DCE 128 bit UUID as specified in [UUID].

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

652 The registry usually generates this id. The client may optionally supply the id attribute for
653 submitted objects. If the client supplies the `id` and it conforms to the format of a URN that
654 specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the id
655 for the object. In this case, the registry must honour a client-supplied id and use it as the id
656 attribute of the object in the registry. If the id is found by the registry to not be globally unique,
657 the registry must raise the error condition: InvalidIdError.

658 If the client does not supply an id for a submitted object then the registry must generate a

659 universally unique id. Whether the client generates the id or whether the registry generates it, it
660 must be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

7.3.2 ID Attribute And Object References

662 The id attribute of an object may be used by other objects to reference the first object. Such
663 references are common both within the SubmitObjectsRequest as well as within the registry.
664 Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the
665 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the
666 SubmitObjectsRequest that needs to be referred to within the request document may be assigned
667 an id by the submitter so that it can be referenced within the request. The submitter may give the
668 object a proper uuid URN, in which case the id is permanently assigned to the object within the
669 registry. Alternatively, the submitter may assign an arbitrary id (not a proper uuid URN) as long
670 as the id is unique within the request document. In this case the id serves as a linkage mechanism
671 within the request document but must be ignored by the registry and replaced with a registry
672 generated id upon submission.

673 When an object in a SubmitObjectsRequest needs to reference an object that is already in the
674 registry, the request must contain an ObjectRef element whose id attribute is the id of the object
675 in the registry. This id is by definition a proper uuid URN. An ObjectRef may be viewed as a
676 proxy within the request for an object that is in the registry.

7.3.3 Audit Trail

678 The RS must create AuditableEvents object with eventType Created for each RegistryObject
679 created via a SubmitObjects request.

7.3.4 Submitting Organization

681 The RS must create an Association of type SubmitterOf between the submitting organization and
682 each RegistryObject created via a SubmitObjects request. (Submitting organization is
683 determined from the organization attribute of the User who submits a SubmitObjects request.)

7.3.5 Error Handling

685 A SubmitObjects request is atomic and either succeeds or fails in total. In the event of success,
686 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
687 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
688 the event of an immediate response for an asynchronous request, the registry sends a
689 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
690 more Error conditions are raised in the processing of the submitted objects. Warning messages
691 do not result in failure of the request. The following business rules apply:

692   **Table 5 Submit Objects Error Handling**

Business Rule	Applies To	Error/Warning
ID not unique	All Classes	Error
Not authorized	All Classes	Error

Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion are set by the RS, and ignored if supplied.	All Classes	Warning

693 7.3.6 Sample SubmitObjectsRequest

694 The following example shows several different use cases in a single SubmitObjectsRequest. It
695 does not show the complete SOAP or [ebMS] Message with the message header and additional
696 payloads in the message for the repository items.

697 A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects
698 that are being submitted. It may also contain any number of ObjectRefs to link objects being
699 submitted to objects already within the registry.

```
700
701  <?xml version = "1.0" encoding = "UTF-8"?>
702  <SubmitObjectsRequest
703    xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
704    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
705    xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
706  spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
707  file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
708    xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
709    xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
710    >
711
712    <rim:LeafRegistryObjectList>
713
714      <!--
715      The following 3 objects package specified ExtrinsicObject in specified
716        RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
717        being  submitted
718        -->
719
720      <rim:RegistryPackage id = "acmePackage1" >
721        <rim:Name>
722          <rim:LocalizedString value = "RegistryPackage #1"/>
723        </rim:Name>
724        <rim:Description>
725          <rim:LocalizedString value = "ACME's package #1"/>
726        </rim:Description>
727      </rim:RegistryPackage>
728
729      <rim:ExtrinsicObject id = "acmeCPP1"  >
730        <rim:Name>
731          <rim:LocalizedString value = "Widget Profile" />
732        </rim:Name>
733        <rim:Description>
734          <rim:LocalizedString value = "ACME's profile for selling widgets" />
735        </rim:Description>
736      </rim:ExtrinsicObject>
737
738      <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject
739  = "acmePackage1" targetObject = "acmeCPP1" />
740
741      <!--
742        The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
743        Where the RegistryPackage is being submitted and the ExtrinsicObject is
744        already in registry
745        -->
```

```
746
747        <rim:RegistryPackage id = "acmePackage2" >
748          <rim:Name>
749            <rim:LocalizedString value = "RegistryPackage #2"/>
750          </rim:Name>
751          <rim:Description>
752            <rim:LocalizedString value = "ACME's package #2"/>
753          </rim:Description>
754        </rim:RegistryPackage>
755
756        <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
757
758        <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
759    sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
760
761        <!--
762          The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
763          where the RegistryPackage and the ExtrinsicObject are already in registry
764          -->
765
766        <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
767        <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
768
769        <!-- id is unspecified implying that registry must create a uuid for this object -->
770
771        <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
772    123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
773
774        <!--
775          The following 3 objects externally link specified ExtrinsicObject using
776          specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
777          are being submitted
778          -->
779
780        <rim:ExternalLink id = "acmeLink1" >
781          <rim:Name>
782            <rim:LocalizedString value = "Link #1"/>
783          </rim:Name>
784          <rim:Description>
785            <rim:LocalizedString value = "ACME's Link #1"/>
786          </rim:Description>
787        </rim:ExternalLink>
788
789        <rim:ExtrinsicObject id = "acmeCPP2"   >
790          <rim:Name>
791            <rim:LocalizedString value = "Sprockets Profile" />
792          </rim:Name>
793          <rim:Description>
794            <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
795          </rim:Description>
796        </rim:ExtrinsicObject>
797
798        <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
799    sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
800
801        <!--
802          The following 2 objects externally link specified ExtrinsicObject using specified
803          ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
804          is already in registry. Note that the targetObject points to an ObjectRef in a
805          previous line
806          -->
807
808        <rim:ExternalLink id = "acmeLink2">
809          <rim:Name>
810            <rim:LocalizedString value = "Link #2"/>
811          </rim:Name>
812          <rim:Description>
813            <rim:LocalizedString value = "ACME's Link #2"/>
814          </rim:Description>
815        </rim:ExternalLink>
816
```

```
817    <rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
818  "ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
819  123456789012"/>
820
821    <!--
822     The following 3 objects externally identify specified ExtrinsicObject using specified
823     ExternalIdentifier, where the ExternalIdentifier is being submitted and the
824     ExtrinsicObject is already in registry. Note that the targetObject points to an
825     ObjectRef in a previous line
826     -->
827
828    <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
829      <rim:Name>
830        <rim:LocalizedString value = "DUNS"/>
831      </rim:Name>
832
833      <rim:Description>
834        <rim:LocalizedString value = "This is the DUNS scheme"/>
835      </rim:Description>
836    </rim:ClassificationScheme>
837
838    <rim:ExternalIdentifier id = "acmeDUNSId"  identificationScheme="DUNS-id" value =
839  "13456789012">
840      <rim:Name>
841        <rim:LocalizedString value = "DUNS" />
842      </rim:Name>
843      <rim:Description>
844        <rim:LocalizedString value = "DUNS ID for ACME"/>
845      </rim:Description>
846    </rim:ExternalIdentifier>
847
848    <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType =
849  "ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
850  123456789012"/>
851
852    <!--
853     The following show submission of a brand new classification scheme in its entirety
854     -->
855    <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
856      <rim:Name>
857        <rim:LocalizedString value = "Geography"/>
858      </rim:Name>
859
860      <rim:Description>
861        <rim:LocalizedString value = "This is a sample Geography scheme"/>
862      </rim:Description>
863
864      <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =
865  "NorthAmerica" >
866        <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =
867  "UnitedStates" />
868        <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
869      </rim:ClassificationNode>
870
871      <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
872        <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
873          <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
874        </rim:ClassificationNode>
875      </rim:ClassificationNode>
876    </rim:ClassificationScheme>
877
878
879    <!--
880     The following show submission of a Automotive sub-tree of ClassificationNodes that
881     gets added to an existing classification scheme named 'Industry'
882     that is already in the registry
883     -->
884
885    <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
886    <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
887  123456789012">
888      <rim:Name>
889        <rim:LocalizedString value = "Automotive" />
```

```
890        </rim:Name>
891        <rim:Description>
892          <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
893        </rim:Description>
894      </rim:ClassificationNode>
895
896      <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
897        <rim:Name>
898          <rim:LocalizedString value = "Parts Supplier" />
899        </rim:Name>
900        <rim:Description>
901          <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
902        </rim:Description>
903      </rim:ClassificationNode>
904
905      <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
906        <rim:Name>
907          <rim:LocalizedString value = "Engine Supplier" />
908        </rim:Name>
909        <rim:Description>
910          <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
911        </rim:Description>
912      </rim:ClassificationNode>
913
914      <!--
915        The following show submission of 2 Classifications of an object  that is already in
916        the registry using 2 ClassificationNodes. One ClassificationNode
917        is being submitted in this request (Japan) while the other is already in the registry.
918        -->
919
920      <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-
921 1234-123456789012" classificationNode = "Japan-id">
922        <rim:Description>
923          <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
924        </rim:Description>
925      </rim:Classification>
926
927      <rim:Classification id = "classificationUsingExistingNode" classifiedObject =
928 "urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
929 123456789012">
930        <rim:Description>
931          <rim:LocalizedString value = "Classifies object using a node in the registry" />
932        </rim:Description>
933      </rim:Classification>
934
935      <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
936    </rim:LeafRegistryObjectList>
937 </SubmitObjectsRequest>
938
```

7.4 The Update Objects Protocol

This section describes the protocol of the Registry Service that allows a Registry Client to update one or more existing Registry Items in the registry on behalf of a Submitting Organization. It is expressed in UML notation as described in Appendix C.

943
944   **Figure 8: Update Objects Sequence Diagram**

945   For details on the schema for the Business documents shown in this process refer to Appendix B.
946   The UpdateObjectsRequest message includes a LeafRegistryObjectList element. The
947   LeafRegistryObjectList element specifies one or more RegistryObjects. Each object in the list
948   must be a current RegistryObject. RegistryObjects must include all attributes, even those the
949   user does not intend to change. A missing attribute is interpreted as a request to set that attribute
950   to NULL.

7.4.1 Audit Trail

951

952   The RS must create AuditableEvents object with eventType Updated for each RegistryObject
953   updated via an UpdateObjects request.

7.4.2 Submitting Organization

954

955   The RS must maintain an Association of type SubmitterOf between the submitting organization
956   and each RegistryObject updated via an UpdateObjects request. If an UpdateObjects request is
957   accepted from a different submitting organization, then the RS must delete the original
958   association object and create a new one. Of course, the AccessControlPolicy may prohibit this
959   sort of update in the first place. (Submitting organization is determined from the organization
960   attribute of the User who submits an UpdateObjects request.)

7.4.3 Error Handling

961

962   An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success,
963   the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
964   of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
965   the event of an immediate response for an asynchronous request, the registry sends a
966   RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
967   more Error conditions are raised in the processing of the updated objects. Warning messages do
968   not result in failure of the request. The following business rules apply:

969   **Table 6: Update Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	All Classes	Error
Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion cannot be changed via the UpdateObjects protocol, ignored if supplied.	All Classes	Warning
RegistryEntries with stability = "Stable" should not be updated.	All Classes	Warning

970 ## 7.5 The Add Slots Protocol

971 This section describes the protocol of the Registry Service that allows a client to add slots to a
972 previously submitted registry entry using the LifeCycleManager. Slots provide a dynamic
973 mechanism for extending registry entries as defined by [ebRIM].

974
975   **Figure 9: Add Slots Sequence Diagram**

976 In the event of success, the registry sends a RegistryResponse with a status of "success" back to
977 the client. In the event of failure, the registry sends a RegistryResponse with a status of "failure"
978 back to the client.

979 ## 7.6 The Remove Slots Protocol

980 This section describes the protocol of the Registry Service that allows a client to remove slots to
981 a previously submitted registry entry using the LifeCycleManager.

982
983   **Figure 10: Remove Slots Sequence Diagram**

7.7 The Approve Objects Protocol

985   This section describes the protocol of the Registry Service that allows a client to approve one or
986   more previously submitted repository items using the LifeCycleManager. Once a repository item
987   is approved it will become available for use by business parties (e.g. during the assembly of new
988   CPAs and Collaboration Protocol Profiles).

989
990   **Figure 11: Approve Objects Sequence Diagram**

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

7.7.1 Audit Trail

993   The RS must create AuditableEvents object with eventType Approved for each RegistryObject
994   approved via an Approve Objects request.

995 ### 7.7.2 Submitting Organization

996 The RS must maintain an Association of type SubmitterOf between the submitting organization
997 and each RegistryObject updated via an ApproveObjects request. If an ApproveObjects request
998 is accepted from a different submitting organization, then the RS must delete the original
999 association object and create a new one. Of course, the AccessControlPolicy may prohibit this
1000 sort of ApproveObjects request in the first place. (Submitting organization is determined from
1001 the organization attribute of the User who submits an ApproveObjects request.)

1002 ### 7.7.3 Error Handling

1003 An ApproveObjects request is atomic and either succeeds or fails in total. In the event of success,
1004 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
1005 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
1006 the event of an immediate response for an asynchronous request, the registry sends a
1007 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
1008 more Error conditions are raised in the processing of the object reference list. Warning messages
1009 do not result in failure of the request. The following business rules apply:

1010 **Table 7: Approve Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "approved".	All Classes other than RegistryEntry classes	Error
Object status is already "Approved".	RegistryEntry Classes	Warning

1011 ## 7.8 The Deprecate Objects Protocol

1012 This section describes the protocol of the Registry Service that allows a client to deprecate one or
1013 more previously submitted repository items using the LifeCycleManager. Once an object is
1014 deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that
1015 object can be submitted. However, existing references to a deprecated object continue to function
1016 normally.

Figure 12: Deprecate Objects Sequence Diagram

1017
1018

1019   For details on the schema for the business documents shown in this process refer to Appendix B.

7.8.1 Audit Trail

1020

1021   The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject
1022   deprecated via a Deprecate Objects request.

7.8.2 Submitting Organization

1023

1024   The RS must maintain an Association of type SubmitterOf between the submitting organization
1025   and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request
1026   is accepted from a different submitting organization, then the RS must delete the original
1027   association object and create a new one. Of course, the AccessControlPolicy may prohibit this
1028   sort of Deprecate Objects request in the first place. (Submitting organization is determined from
1029   the organization attribute of the User who submits a Deprecate Objects request.)

7.8.3 Error Handling

1030

1031   A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of
1032   success, the registry sends a RegistryResponse with a status of "Success" back to the client. In
1033   the event of failure, the registry sends a RegistryResponse with a status of "Failure" back to the
1034   client. In the event of an immediate response for an asynchronous request, the registry sends a
1035   RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
1036   more Error conditions are raised in the processing of the object reference list. Warning messages
1037   do not result in failure of the request. The following business rules apply:

1038   **Table 8: Deprecate Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryEntry	Error

	Classes	
Only RegistryEntries may be "deprecated".	All Classes other than RegistryEntry classes	Error
Object status is already "Deprecated".	RegistryEntry Classes	Warning

7.9 The Remove Objects Protocol

1039

1040 This section describes the protocol of the Registry Service that allows a client to remove one or
1041 more RegistryObject instances and/or repository items using the LifeCycleManager.

1042 The RemoveObjectsRequest message is sent by a client to remove RegistryObject instances
1043 and/or repository items. The RemoveObjectsRequest element includes an XML attribute called
1044 deletionScope which is an enumeration that can have the values as defined by the following
1045 sections.

7.9.1 Deletion Scope DeleteRepositoryItemOnly

1046

1047 This deletionScope specifies that the request should delete the repository items for the specified
1048 registry entries but not delete the specified registry entries. This is useful in keeping references to
1049 the registry entries valid.

7.9.2 Deletion Scope DeleteAll

1050

1051 This deletionScope specifies that the request should delete both the RegistryObject and the
1052 repository item for the specified registry entries. Only if all references (e.g. Associations,
1053 Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject
1054 then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to
1055 remove a RegistryObject while it still has references raises an error condition:
1056 InvalidRequestError.

1057 The remove object protocol is expressed in UML notation as described in Appendix C.

1058

1059   **Figure 13: Remove Objects Sequence Diagram**

1060   For details on the schema for the business documents shown in this process refer to Appendix B.

1061   7.9.3 Error Handling

1062   A Remove Objects request is atomic and either succeeds or fails in total. In the event of success,
1063   the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
1064   of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
1065   the event of an immediate response for an asynchronous request, the registry sends a
1066   RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
1067   more Error conditions are raised in the processing of the object reference list. Warning messages
1068   do not result in failure of the request. The following business rules apply:

1069   **Table 9: Remove Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistryObject Classes	Error

1070

8   Query Management Service

This section describes the capabilities of the Registry Service that allow a client (QueryManagerClient) to search for or query different kind of registry objects in the ebXML Registry using the QueryManager interface of the Registry. The Registry supports the following query capabilities:

- Filter Query

- SQL Query

The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry implementation. The SQL Query mechanism is an optional feature and MAY be provided by a registry implementation. However, if a vendor provides an SQL query capability to an ebXML Registry it SHALL conform to this document. As such this capability is a normative yet optional capability.

In a future version of this specification, the W3C XQuery syntax may be considered as another query syntax.

The Registry will hold a self-describing capability profile that identifies all supported AdhocQuery options. This profile is described in Appendix H.

8.1   Ad Hoc Query Request/Response

A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The AdhocQueryRequest contains a subelement that defines a query in one of the supported Registry query mechanisms.

The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously back to the client. The AdhocQueryResponse returns a collection of objects whose element type depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate classes in [ebRIM] such as RegistryObject and RegistryEntry.

Any errors in the query request messages are indicated in the corresponding query response message.

1098

1099   **Figure 14: Submit Ad Hoc Query Sequence Diagram**

1100   For details on the schema for the business documents shown in this process refer to Appendix
1101   B.2.

1102   **Definition**
1103
```
1104    <element name="AdhocQueryRequest">
1105       <complexType>
1106          <sequence>
1107             <element ref="tns:ResponseOption" minOccurs="1" maxOccurs="1" />
1108             <choice minOccurs="1" maxOccurs="1">
1109                <element ref="tns:FilterQuery" />
1110                <element ref="tns:SQLQuery" />
1111             </choice>
1112          </sequence>
1113       </complexType>
1114    </element>
1115
1116    <element name="AdhocQueryResponse">
1117       <complexType>
1118          <choice minOccurs="1" maxOccurs="1">
1119             <element ref="tns:FilterQueryResult" />
1120             <element ref="tns:SQLQueryResult" />
1121          </choice>
1122       </complexType>
1123    </element>
```
1124

1125   ### 8.1.1 Query Response Options

1126   **Purpose**

1127   A QueryManagerClient may specify what an ad hoc query must return within an
1128   AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest.
1129   ResponseOption element has an attribute "returnType" and its values are:

1130 • ObjectRef - This option specifies that the AdhocQueryResponse may contain a collection of
1131   ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return
1132   just the identifiers of the registry objects.
1133 • RegistryObject - This option specifies that the AdhocQueryResponse may contain a
1134   collection of RegistryObject XML elements as defined in [ebRIM Schema]. In this case all
1135   attributes of the registry objects are returned (objectType, name, description, …) in addition
1136   to id attribute.
1137 • RegistryEntry - This option specifies that the AdhocQueryResponse may contain a collection
1138   of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which
1139   correspond to RegistryEntry or RegistryObject attributes.
1140 • LeafClass - This option specifies that the AdhocQueryResponse may contain a collection of
1141   XML elements that correspond to leaf classes as defined in [ebRIM Schema].
1142 • LeafClassWithRepositoryItem - This option specifies that the AdhocQueryResponse may
1143   contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema]
1144   accompanied with their repository items or RegistryEntry or RegistryObject and their
1145   attributes. Linking of ExtrinsicObject and its repository item is done via contentURI as
1146   explained in Section 8.4 -Content Retrieval.

1147 ResponseOption element also has an attribute "returnComposedObjects". It specifies whether or
1148 not the whole hierarchy of composed objects are returned with the registry objects.

1149 If "returnType" is higher then the RegistryObject option, then the highest option that satisfies the
1150 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return
1151 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass
1152 option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then
1153 RegistryObject metadata will be returned.

1154 **Definition**
1155
```
1156  <complexType name="ResponseOptionType">
1157     <attribute name="returnType" default="RegistryObject">
1158        <simpleType>
1159           <restriction base="NMTOKEN">
1160              <enumeration value="ObjectRef" />
1161              <enumeration value="RegistryObject" />
1162              <enumeration value="RegistryEntry" />
1163              <enumeration value="LeafClass" />
1164              <enumeration value="LeafClassWithRepositoryItem" />
1165           </restriction>
1166        </simpleType>
1167     </attribute>
1168     <attribute name="returnComposedObjects" type="boolean" default="false" />
1169  </complexType>
1170  <element name="ResponseOption" type="tns:ResponseOptionType" />
```
1171

8.2 Filter Query Support

1173 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
1174 conforming Registry implementation. Each query alternative is directed against a single class
1175 defined by the ebXML Registry Information Model (ebRIM). There are two types of filter
1176 queries depending on which classes are queried on.

1177   • Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic
1178   queries that might return different subclasses of the class that is queried on. The result of
1179   such a query is a set of XML elements that correspond to instances of any class that satisfies
1180   the responseOption defined previously in Section 8.1.1. An example might be that
1181   RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances
1182   that satisfy the query. This implies that response might return XML elements that correspond
1183   to classes like ClassificationScheme, RegistryPackage, Organization and Service.
1184   • Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact
1185   traversals of these classes. Responses to these queries are accordingly constrained.

1186   A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an
1187   AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified
1188   herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified
1189   in Section 8.1.

1190   Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of
1191   classes derived from a single class and its associations with other classes as defined by ebRIM.
1192   Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For
1193   example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a
1194   class that is associated with Z. The ebRIM Binding for C might be as in Figure 15

1195
1196   **Figure 15: Example ebRIM Binding**

1197   Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and
1198   Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to
1199   which ebRIM association is intended. The name of the query is determined by the root class, i.e.
1200   this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances
1201   that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are
1202   limited to instances that are linked to their parent node by the identified association.

1203   Each FilterQuery alternative depends upon one or more class filters, where a class filter is a
1204   restricted predicate clause over the attributes of a single class. Class methods that are defined in
1205   ebRIM and that return simple types constitute "visible attributes" that are valid choices for
1206   predicate clauses. Names of those attributes will be same as name of the corresponding method
1207   just without the prefix 'get'. For example, in case of "getLevelNumber" method the
1208   corresponding visible attribute is "levelNumber". The supported class filters are specified in
1209   Section 8.2.13 and the supported predicate clauses are defined in Section 8.2.14. A FilterQuery

1210   will be composed of elements that traverse the tree to determine which branches satisfy the
1211   designated class filters, and the query result will be the set of instances that support such a
1212   branch.

1213   In the above example, the CQuery element will have three subelements, one a CFilter on the C
1214   class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on
1215   the Y class to eliminate branches from C to Y where the target of the association does not satisfy
1216   the YFilter, and a third to eliminate branches along a path from C through Z to V. The third
1217   element is called a branch element because it allows class filters on each class along the path
1218   from C to V. In general, a branch element will have subelements that are themselves class filters,
1219   other branch elements, or a full-blown query on the class in the path.

1220   If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one
1221   branch, filter or query element on Y is allowed. However, if the association is one-to-many, then
1222   multiple branch, filter or query elements are allowed. This allows one to specify that an instance
1223   of C must have associations with multiple instances of Y before the instance of C is said to
1224   satisfy the branch element.

1225   The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be
1226   stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then
1227   the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax
1228   follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their
1229   respective superclass queries. Structures of XML elements that match the ebRIM classes are
1230   explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in
1231   ebRIM whenever possible.

1232   The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual
1233   hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify
1234   the effect of that binding on query semantics.

1235   ## 8.2.1 FilterQuery

1236   **Purpose**
1237   To identify a set of queries that traverse specific registry class. Each alternative assumes a
1238   specific binding to ebRIM. The status is a success indication or a collection of warnings and/or
1239   exceptions.

1240   **Definition**
1241
1242
```
<element name="FilterQuery">
   <complexType>
      <choice minOccurs="1" maxOccurs="1">
         <element ref="tns:RegistryObjectQuery" />
         <element ref="tns:RegistryEntryQuery" />
         <element ref="tns:AssociationQuery" />
         <element ref="tns:AuditableEventQuery" />
         <element ref="tns:ClassificationQuery" />
         <element ref="tns:ClassificationNodeQuery" />
         <element ref="tns:ClassificationSchemeQuery" />
         <element ref="tns:RegistryPackageQuery" />
         <element ref="tns:ExtrinsicObjectQuery" />
         <element ref="tns:OrganizationQuery" />
         <element ref="tns:ServiceQuery" />
```

```
1256            </choice>
1257          </complexType>
1258      </element>
1259
1260      <element name="FilterQueryResult">
1261          <complexType>
1262             <choice minOccurs="1" maxOccurs="1">
1263                <element ref="tns:RegistryObjectQueryResult" />
1264                <element ref="tns:RegistryEntryQueryResult" />
1265                <element ref="tns:AssociationQueryResult" />
1266                <element ref="tns:AuditableEventQueryResult" />
1267                <element ref="tns:ClassificationQueryResult" />
1268                <element ref="tns:ClassificationNodeQueryResult" />
1269                <element ref="tns:ClassificationSchemeQueryResult" />
1270                <element ref="tns:RegistryPackageQueryResult" />
1271                <element ref="tns:ExtrinsicObjectQueryResult" />
1272                <element ref="tns:OrganizationQueryResult" />
1273                <element ref="tns:ServiceQueryResult" />
1274             </choice>
1275          </complexType>
1276      </element>
1277
```

1278   **Semantic Rules**

1279   1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.

1280   2. Semantic rules specify the procedure for implementing the evaluation of Filter Queries.
1281   Implementations do not necessarily have to follow the same procedure provided that the
1282   same effect is achieved.

1283   3. Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
1284   Each XML attribute carries a value derived from the value of an attribute specified in the
1285   Registry Information Model [ebRIM Schema].

1286   4. For each FilterQuery subelement there is only one corresponding FilterQueryResult
1287   subelement that must be returned as a response. Class name of the FilterQueryResult
1288   subelement has to match the class name of the FilterQuery subelement.

1289   5. If a Filter, Branch or Query element for a class has no sub-elements then every persistent
1290   instance of that class satisfies the Filter, Branch or Query.

1291   6. If an error condition is raised during any part of the execution of a FilterQuery, then the
1292   status attribute of the XML RegistryResult is set to "failure" and no AdHocQueryResult
1293   element is returned; instead, a RegistryErrorList element must be returned with its
1294   highestSeverity element set to "error". At least one of the RegistryError elements in the
1295   RegistryErrorList will have its severity attribute set to "error".

1296   7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of
1297   the XML RegistryResult is set to "success" and an appropriate FilterQueryResult element
1298   must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of
1299   the RegistryErrorList is set to "warning" and the serverity attribute of each RegistryError is
1300   set to "warning".

1301   ## 8.2.2 RegistryObjectQuery

1302   **Purpose**

1303   To identify a set of registry object instances as the result of a query over selected registry
1304   metadata.

1305   **ebRIM Binding**

1306   **Figure 16: ebRIM Binding for RegistryObjectQuery**

1307   **Definition**
1308   ```
        <complexType name="RegistryObjectQueryType">
1309   <sequence>
1310   <element ref="tns:RegistryObjectFilter" minOccurs="0" maxOccurs="1" />
1311   <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="unbounded" />
1312   <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="unbounded" />
1313   <element ref="tns:NameBranch" minOccurs="0" maxOccurs="1" />
1314   <element ref="tns:DescriptionBranch" minOccurs="0" maxOccurs="1" />
1315   <element ref="tns:ClassifiedByBranch" minOccurs="0" maxOccurs="unbounded" />
1316   <element ref="tns:SlotBranch" minOccurs="0" maxOccurs="unbounded" />
1317   <element ref="tns:SourceAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1318   <element ref="tns:TargetAssociationBranch" minOccurs="0" maxOccurs="unbounded" />
1319   </sequence>
1320   </complexType>
1321   <element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType" />
1322
1323   <complexType name="LeafRegistryObjectListType">
1324   <choice minOccurs="0" maxOccurs="unbounded">
1325   <element ref="tns:ObjectRef" />
1326   <element ref="tns:Association" />
1327   <element ref="tns:AuditableEvent" />
1328   <element ref="tns:Classification" />
1329   <element ref="tns:ClassificationNode" />
1330   <element ref="tns:ClassificationScheme" />
1331   <element ref="tns:ExternalIdentifier" />
1332   <element ref="tns:ExternalLink" />
1333   <element ref="tns:ExtrinsicObject" />
        ```
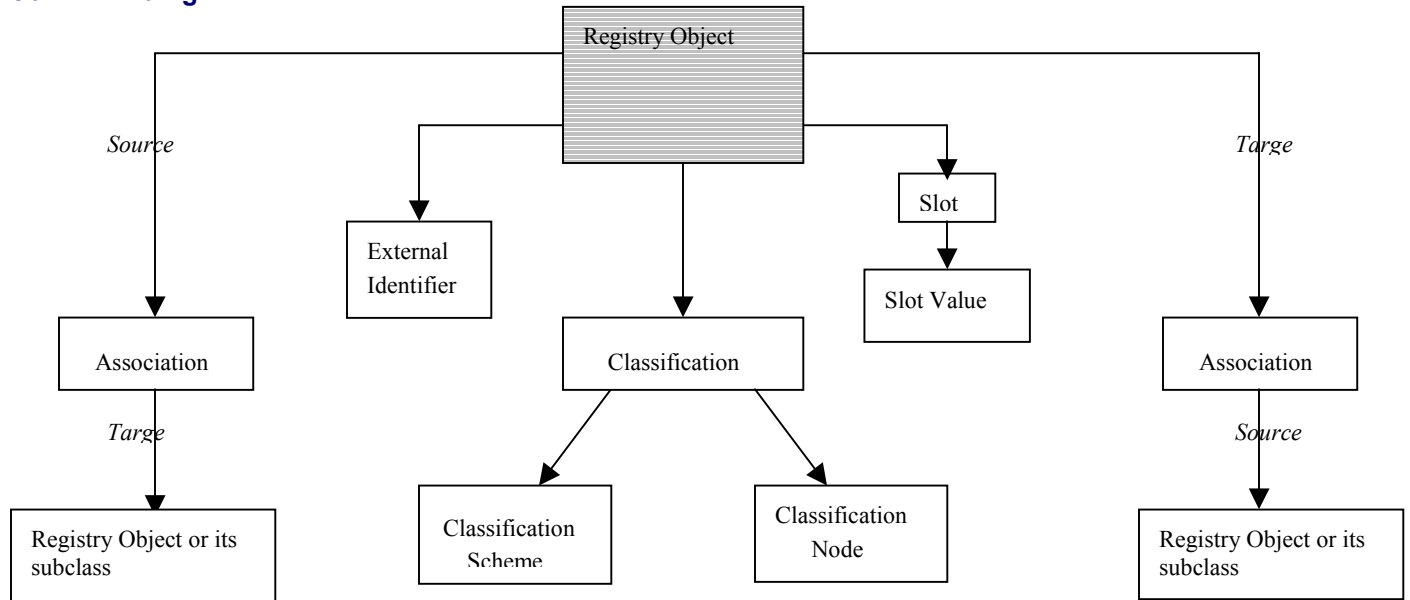
```
1334          <element ref="tns:Organization" />
1335          <element ref="tns:RegistryPackage" />
1336          <element ref="tns:Service" />
1337          <element ref="tns:ServiceBinding" />
1338          <element ref="tns:SpecificationLink" />
1339          <element ref="tns:User" />
1340       </choice>
1341    </complexType>
1342
1343    <complexType name="RegistryObjectListType">
1344       <complexContent>
1345          <extension base="tns:LeafRegistryObjectListType">
1346             <choice minOccurs="0" maxOccurs="unbounded">
1347                <element ref="tns:RegistryEntry" />
1348                <element ref="tns:RegistryObject" />
1349             </choice>
1350          </extension>
1351       </complexContent>
1352    </complexType>
1353    <element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType" />
1354
1355    <complexType name="InternationalStringBranchType">
1356       <sequence>
1357          <element ref="tns:LocalizedStringFilter" minOccurs="0" maxOccurs="unbounded" />
1358       </sequence>
1359    </complexType>
1360
1361    <complexType name="AssociationBranchType">
1362       <sequence>
1363          <element ref="tns:AssociationFilter" minOccurs="0" maxOccurs="1" />
1364          <choice minOccurs="0" maxOccurs="1">
1365             <element ref="tns:ExternalLinkFilter" minOccurs="0" maxOccurs="1" />
1366             <element ref="tns:ExternalIdentifierFilter" minOccurs="0" maxOccurs="1" />
1367             <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1368             <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1369             <element ref="tns:AssociationQuery" minOccurs="0" maxOccurs="1" />
1370             <element ref="tns:ClassificationQuery" minOccurs="0" maxOccurs="1" />
1371             <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1372             <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1373             <element ref="tns:OrganizationQuery" minOccurs="0" maxOccurs="1" />
1374             <element ref="tns:AuditableEventQuery" minOccurs="0" maxOccurs="1" />
1375             <element ref="tns:RegistryPackageQuery" minOccurs="0" maxOccurs="1" />
1376             <element ref="tns:ExtrinsicObjectQuery" minOccurs="0" maxOccurs="1" />
1377             <element ref="tns:ServiceQuery" minOccurs="0" maxOccurs="1" />
1378             <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
1379             <element ref="tns:ServiceBindingBranch" minOccurs="0" maxOccurs="1" />
1380             <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="1" />
1381          </choice>
1382       </sequence>
1383    </complexType>
1384    <element name="SourceAssociationBranch" type="tns:AssociationBranchType" />
1385    <element name="TargetAssociationBranch" type="tns:AssociationBranchType" />
1386
1387    <element name="ClassifiedByBranch">
1388       <complexType>
1389          <sequence>
1390             <element ref="tns:ClassificationFilter" minOccurs="0" maxOccurs="1" />
```

```
1391              <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
1392              <element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" />
1393          </sequence>
1394        </complexType>
1395     </element>
1396
1397     <element name="SlotBranch">
1398        <complexType>
1399          <sequence>
1400            <element ref="tns:SlotFilter" minOccurs="0" maxOccurs="1" />
1401            <element ref="tns:SlotValueFilter" minOccurs="0" maxOccurs="unbounded" />
1402          </sequence>
1403        </complexType>
1404     </element>
1405
1406        <element name = "UserBranch">
1407            <complexType>
1408                <sequence>
1409                    <element ref = "tns:UserFilter" minOccurs = "0" maxOccurs="1"/>
1410                    <element ref = "tns:PostalAddressFilter" minOccurs = "0" maxOccurs="1"/>
1411                    <element ref = "tns:TelephoneNumberFilter" minOccurs = "0" maxOccurs="unbounded"/>
1412                    <element ref = "tns:EmailAddressFilter" minOccurs = "0" maxOccurs="unbounded"/>
1413                    <element ref = "tns:OrganizationQuery" minOccurs = "0" maxOccurs="1"/>
1414                </sequence>
1415            </complexType>
1416        </element>
1417
1418     <complexType name="ServiceBindingBranchType">
1419        <sequence>
1420          <element ref="tns:ServiceBindingFilter" minOccurs="0" maxOccurs="1" />
1421          <element ref="tns:SpecificationLinkBranch" minOccurs="0" maxOccurs="unbounded" />
1422          <element ref="tns:ServiceBindingTargetBranch" minOccurs="0" maxOccurs="1" />
1423        </sequence>
1424     </complexType>
1425     <element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType" />
1426     <element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType" />
1427
1428     <element name="SpecificationLinkBranch">
1429        <complexType>
1430          <sequence>
1431            <element ref="tns:SpecificationLinkFilter" minOccurs="0" maxOccurs="1" />
1432            <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
1433            <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
1434          </sequence>
1435        </complexType>
1436     </element>
1437
```

1438   **Semantic Rules**

1439   1. Let RO denote the set of all persistent RegistryObject instances in the Registry. The
1440   following steps will eliminate instances in RO that do not satisfy the conditions of the
1441   specified filters.

1442   a) If RO is empty then go to number 2 below.

1443   b) If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a
1444   registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from
1445   RO. If RO is empty then continue to the next numbered rule.

1446   c) If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise,
1447   let x be a remaining registry object in RO. If x is not linked to at least one
1448   ExternalIdentifier instance, then remove x from RO; otherwise, treat each
1449   ExternalIdentifierFilter element separately as follows: Let EI be the set of
1450   ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If
1451   EI is empty, then remove x from RO. If RO is empty then continue to the next numbered
1452   rule.

1453   d) If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a
1454   remaining registry object in RO. If x doesn't have an auditable event that satisfy
1455   AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is
1456   empty then continue to the next numbered rule.

1457   e) If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining
1458   registry object in RO. If x does not have a name then remove x from RO. If RO is empty
1459   then continue to the next numbered rule; otherwise treat NameBranch as follows: If any
1460   LocalizedStringFilter that is specified is not satisfied by at least one of the
1461   LocalizedStrings that constitute the name of the registry object then remove x from RO.
1462   If RO is empty then continue to the next numbered rule.

1463   f) If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a
1464   remaining registry object in RO. If x does not have a name then remove x from RO. If
1465   RO is empty then continue to the next numbered rule; otherwise treat DescriptionBranch
1466   as follows: If any LocalizedStringFilter that is specified is not satisfied by some of the
1467   LocalizedStrings that constitute the description of the registry object then remove x from
1468   RO. If RO is empty then continue to the next numbered rule.

1469   g) If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x
1470   be a remaining registry object in RO. If x is not the classifiedObject of at least one
1471   Classification instance, then remove x from RO; otherwise, treat each
1472   ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified
1473   within the ClassifiedByBranch, then let CL be the set of all Classification instances that
1474   have x as the classifiedObject; otherwise, let CL be the set of Classification instances that
1475   satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then
1476   remove x from RO and continue to the next numbered rule. Otherwise, if CL is not
1477   empty, and if a ClassificationSchemeQuery is specified, then replace CL by the set of
1478   remaining Classification instances in CL whose defining classification scheme satisfies
1479   the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and
1480   continue to the next numbered rule. Otherwise, if CL remains not empty, and if a
1481   ClassificationNodeQuery is specified, then replace CL by the set of remaining
1482   Classification instances in CL for which a classification node exists and for which that
1483   classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then
1484   remove x from RO. If RO is empty then continue to the next numbered rule.

1485   h) If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a
1486   remaining registry object in RO. If x is not linked to at least one Slot instance, then
1487   remove x from RO. If RO is empty then continue to the next numbered rule; otherwise,
1488   treat each SlotBranch element separately as follows: If a SlotFilter is not specified within
1489   the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the
1490   set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty,
1491   then remove x from RO and continue to the next numbered rule. Otherwise, if SL
1492   remains not empty, and if a SlotValueFilter is specified, replace SL by the set of
1493   remaining Slot instances in SL for which every specified SlotValueFilter is valid. If SL is
1494   empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1495   i) If a SourceAssociationBranch element is not specified then go to the next step; otherwise,
1496   let x be a remaining registry object in RO. If x is not the source object of at least one
1497   Association instance, then remove x from RO. If RO is empty then continue to the next
1498   numbered rule; otherwise, treat each SourceAssociationBranch element separately as
1499   follows:
1500   If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be
1501   the set of all Association instances that have x as a source object; otherwise, let AF be the
1502   set of Association instances that satisfy the AssociationFilter and have x as the source
1503   object. If AF is empty, then remove x from RO.
1504
1505   If RO is empty then continue to the next numbered rule.
1506
1507   If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT
1508   be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target
1509   object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1510   then continue to the next numbered rule.
1511
1512   If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let
1513   ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1514   are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1515   RO is empty then continue to the next numbered rule.
1516
1517   If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT
1518   be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1519   target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1520   empty then continue to the next numbered rule.
1521
1522   If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT
1523   be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the
1524   target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1525   empty then continue to the next numbered rule.
1526
1527   If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let
1528   ROT be the set of ClassificationScheme instances that satisfy the
1529   ClassificationSchemeQuery and are the target object of some element of AF. If ROT is
1530   empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1531
1532   If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let
1533   ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1534   and are the target object of some element of AF. If ROT is empty, then remove x from
1535   RO. If RO is empty then continue to the next numbered rule.

1536

1537   If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT
1538   be the set of Organization instances that satisfy the OrganizationQuery and are the target
1539   object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1540   then continue to the next numbered rule.

1541

1542   If an AuditableEventQuery is specified within the SourceAssociationBranch, then let
1543   ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1544   the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1545   is empty then continue to the next numbered rule.

1546

1547   If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let
1548   ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1549   are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1550   RO is empty then continue to the next numbered rule.

1551

1552   If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let
1553   ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1554   the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1555   is empty then continue to the next numbered rule.

1556

1557   If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the
1558   set of Service instances that satisfy the ServiceQuery and are the target object of some
1559   element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue
1560   to the next numbered rule.

1561

1562   If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set
1563   of User instances that are the target object of some element of AF. If ROT is empty, then
1564   remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the
1565   member of ROT. If a UserFilter element is specified within the UserBranch, and if u does
1566   not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from
1567   RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter
1568   element is specified within the UserBranch, and if the postal address of u does not satisfy
1569   that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is
1570   empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are
1571   specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied
1572   by at least one of the telephone numbers of u then remove u from ROT. If ROT is empty,
1573   then remove x from RO. If RO is empty then continue to the next numbered rule. If an
1574   OrganizationQuery element is specified within the UserBranch, then let o be the
1575   Organization instance that is identified by the organization that u is affiliated with. If o
1576   doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROT.
1577   If ROT is empty, then remove x from RO. If RO is empty then continue to the next
1578   numbered rule.
1579
1580   If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT
1581   be the set of Classification instances that satisfy the ClassificationQuery and are the
1582   target object of some element of AF. If ROT is empty, then remove x from RO. If RO is
1583   empty then continue to the next numbered rule (Rule 2).
1584
1585   If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let
1586   ROT be the set of ServiceBinding instances that are the target object of some element of
1587   AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next
1588   numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is
1589   specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then
1590   remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then
1591   continue to the next numbered rule. If a SpecificationLinkBranch is specified within the
1592   ServiceBindingBranch then consider each SpecificationLinkBranch element separately as
1593   follows:

1594	Let sb be a remaining service binding in ROT. Let SL be the set of all specification link
1595	instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1596	specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1597	remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then
1598	remove x from RO. If RO is empty then continue to the next numbered rule. If a
1599	RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1600	be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:
1601	Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
1602	not a specification link for at least one registry object in RO, then remove sl from SL. If
1603	SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO
1604	is empty then continue to the next numbered rule. If a RegistryEntryQuery element is
1605	specified within the SpecificationLinkBranch then let sl be a remaining specification link
1606	in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the
1607	RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least
1608	one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from
1609	ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next
1610	numbered rule. If a ServiceBindingTargetBranch is specified within the
1611	ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy
1612	the ServiceBindingTargetBranch and are the target service binding of some element of
1613	ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from
1614	RO. If RO is empty then continue to the next numbered rule.
1615	
1616	If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let
1617	ROT be the set of SpecificationLink instances that are the target object of some element
1618	of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the
1619	next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is
1620	specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1621	remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then
1622	continue to the next numbered rule. If a RegistryObjectQuery element is specified within
1623	the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat
1624	RegistryObjectQuery element as follows: Let RO be the result set of the
1625	RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
1626	registry object in RO, then remove sl from ROT. If ROT is empty then remove x from
1627	RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery
1628	element is specified within the SpecificationLinkBranch then let sl be a remaining
1629	specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the
1630	result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification
1631	link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then
1632	remove x from RO. If RO is empty then continue to the next numbered rule.
1633	
1634	If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be
1635	the set of Association instances that satisfy the AssociationQuery and are the target object
1636	of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then
1637	continue to the next numbered rule (Rule 2).
1638	

1639   j)   If a TargetAssociationBranch element is not specified then go to the next step; otherwise,
1640   let x be a remaining registry object in RO. If x is not the target object of some
1641   Association instance, then remove x from RO. If RO is empty then continue to the next
1642   numbered rule; otherwise, treat each TargetAssociationBranch element separately as
1643   follows:

1644

1645   If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be
1646   the set of all Association instances that have x as a target object; otherwise, let AF be the
1647   set of Association instances that satisfy the AssociationFilter and have x as the target
1648   object. If AF is empty, then remove x from RO. If RO is empty then continue to the next
1649   numbered rule.

1650

1651   If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be
1652   the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source
1653   object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1654   then continue to the next numbered rule.

1655

1656   If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let
1657   ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1658   are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1659   RO is empty then continue to the next numbered rule.

1660

1661   If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS
1662   be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1663   source object of some element of AF. If ROS is empty, then remove x from RO. If RO is
1664   empty then continue to the next numbered rule.

1665

1666   If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS
1667   be the set of
1668   RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of
1669   some element of AF. If ROS is empty, then remove x from RO. If RO is empty then
1670   continue to the next numbered rule.

1671

1672   If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let
1673   ROS be the set of ClassificationScheme instances that satisfy the
1674   ClassificationSchemeQuery and are the source object of some element of AF. If ROS is
1675   empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1676

1677   If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let
1678   ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1679   and are the source object of some element of AF. If ROS is empty, then remove x from
1680   RO. If RO is empty then continue to the next numbered rule.

1681

1682   If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS
1683   be the set of Organization instances that satisfy the OrganizationQuery and are the source
1684   object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1685   then continue to the next numbered rule.

1686

1687   If an AuditableEventQuery is specified within the TargetAssociationBranch, then let
1688   ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1689   the source object of some element of AF. If ROS is empty, then remove x from RO. If
1690   RO is empty then continue to the next numbered rule.

1691

1692   If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let
1693   ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1694   are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1695   RO is empty then continue to the next numbered rule.

1696

1697   If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let
1698   ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1699   the source object of some element of AF. If ROS is empty, then remove x from RO. If
1700   RO is empty then continue to the next numbered rule.

1701

1702   If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the
1703   set of Service instances that satisfy the ServiceQuery and are the source object of some
1704   element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue
1705   to the next numbered rule.

1706

1707   If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set
1708   of User instances that are the source object of some element of AF. If ROS is empty, then
1709   remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the
1710   member of ROS. If a UserFilter element is specified within the UserBranch, and if u does
1711   not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from
1712   RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter
1713   element is specified within the UserBranch, and if the postal address of u does not satisfy
1714   that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is
1715   empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are
1716   specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied
1717   by some of the telephone numbers of u then remove u from ROS. If ROS is empty, then
1718   remove x from RO. If RO is empty then continue to the next numbered rule. If an
1719   OrganizationQuery element is specified within the UserBranch, then let o be the
1720   Organization instance that is identified by the organization that u is affiliated with. If o
1721   doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROS.
1722   If ROS is empty, then remove x from RO. If RO is empty then continue to the next
1723   numbered rule.

1724

1725   If a ClassificationQuery is specified within the TargetAssociationBranch, then let ROS be
1726   the set of Classification instances that satisfy the ClassificationQuery and are the source
1727   object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1728   then continue to the next numbered rule (Rule 2).
1729

1730   If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS
1731   be the set of ServiceBinding instances that are the source object of some element of AF.
1732   If ROS is empty, then remove x from RO. If RO is empty then continue to the next
1733   numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is
1734   specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then
1735   remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then
1736   continue to the next numbered rule. If a SpecificationLinkBranch is specified within the
1737   ServiceBindingBranch then consider each SpecificationLinkBranch element separately as
1738   follows:
1739   Let sb be a remaining service binding in ROS. Let SL be the set of all specification link
1740   instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
1741   specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1742   remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then
1743   remove x from RO. If RO is empty then continue to the next numbered rule. If a
1744   RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl
1745   be a remaining specification link in SL. Treat RegistryObjectQuery element as follows:
1746   Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is
1747   not a specification link for some registry object in RO, then remove sl from SL. If SL is
1748   empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is
1749   empty then continue to the next numbered rule. If a RegistryEntryQuery element is
1750   specified within the SpecificationLinkBranch then let sl be a remaining specification link
1751   in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the
1752   RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
1753   registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.
1754   If ROS is empty then remove x from RO. If RO is empty then continue to the next
1755   numbered rule.
1756

1757   If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let
1758   ROS be the set of SpecificationLink instances that are the source object of some element
1759   of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the
1760   next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is
1761   specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
1762   remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then
1763   continue to the next numbered rule. If a RegistryObjectQuery element is specified within
1764   the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat
1765   RegistryObjectQuery element as follows: Let RO be the result set of the
1766   RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some
1767   registry object in RO, then remove sl from ROS. If ROS is empty then remove x from
1768   RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery
1769   element is specified within the SpecificationLinkBranch then let sl be a remaining
1770   specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the
1771   result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification
1772   link for some registry entry in RE, then remove sl from ROS. If ROS is empty then
1773   remove x from RO. If RO is empty then continue to the next numbered rule. If a
1774   ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT
1775   be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and
1776   are the target service binding of some element of ROT. If SBT is empty then remove sb
1777   from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the
1778   next numbered rule.
1779
1780   If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be
1781   the set of Association instances that satisfy the AssociationQuery and are the source
1782   object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1783   then continue to the next numbered rule (Rule 2).

1784 2. If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set
1785   RO to be the result of the RegistryObjectQuery.

1786 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
1787   within the RegistryResponse.

1788 **Examples**

1789 A client application needs all items that are classified by two different classification schemes,
1790 one based on "Industry" and another based on "Geography". Both schemes have been defined by
1791 ebXML and are registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography",
1792 respectively. The following query identifies registry entries for all registered items that are
1793 classified by Industry as any subnode of "Automotive" and by Geography as any subnode of
1794 "Asia/Japan".
1795

```
1796    <AdhocQueryRequest>
1797      <ResponseOption returnType = "RegistryEntry"/>
1798      <FilterQuery>
1799        <RegistryObjectQuery>
1800          <ClassifiedByBranch>
1801            <ClassificationFilter>
1802              <Clause>
```

```
1803            <SimpleClause leftArgument = "path">
1804              <StringClause stringPredicate = "Equal">//Automotive</StringClause>
1805            </SimpleClause>
1806          </Clause>
1807        </ClassificationFilter>
1808        <ClassificationSchemeQuery>
1809          <NameBranch>
1810            <LocalizedStringFilter>
1811              <Clause>
1812                <SimpleClause leftArgument = "value">
1813                  <StringClause stringPredicate = "Equal">urn:ebxml:cs:industry</StringClause>
1814                </SimpleClause>
1815              </Clause>
1816            </LocalizedStringFilter>
1817          </NameBranch>
1818        </ClassificationSchemeQuery>
1819      </ClassifiedByBranch>
1820      <ClassifiedByBranch>
1821        <ClassificationFilter>
1822          <Clause>
1823            <SimpleClause leftArgument = "path">
1824              <StringClause stringPredicate = "StartsWith">/Geography-id/Asia/Japan</StringClause>
1825            </SimpleClause>
1826          </Clause>
1827        </ClassificationFilter>
1828        <ClassificationSchemeQuery>
1829          <NameBranch>
1830            <LocalizedStringFilter>
1831              <Clause>
1832                <SimpleClause leftArgument = "value">
1833                  <StringClause stringPredicate = "Equal">urn:ebxml:cs:geography</StringClause>
1834                </SimpleClause>
1835              </Clause>
1836            </LocalizedStringFilter>
1837          </NameBranch>
1838        </ClassificationSchemeQuery>
1839      </ClassifiedByBranch>
1840    </RegistryObjectQuery>
1841  </FilterQuery>
1842 </AdhocQueryRequest>
1843
```

1844 A client application wishes to identify all RegistryObject instances that are classified by some
1845 internal classification scheme and have some given keyword as part of the description of one of
1846 the classification nodes of that classification scheme. The following query identifies all such
1847 RegistryObject instances. The query takes advantage of the knowledge that the classification
1848 scheme is internal, and thus that all of its nodes are fully described as ClassificationNode
1849 instances.

```
1850
1851   <AdhocQueryRequest>
1852     <ResponseOption returnType = "RegistryObject"/>
1853     <FilterQuery>
1854       <RegistryObjectQuery>
1855         <ClassifiedByBranch>
1856           <ClassificationNodeQuery>
1857             <DescriptionBranch>
1858               <LocalizedStringFilter>
1859                 <Clause>
1860                   <SimpleClause leftArgument = "value">
1861                     <StringClause stringPredicate = "Equal">transistor</StringClause>
1862                   </SimpleClause>
1863                 </Clause>
1864               </LocalizedStringFilter>
1865             </DescriptionBranch>
1866           </ClassificationNodeQuery>
1867         </ClassifiedByBranch>
1868       </RegistryObjectQuery>
1869     </FilterQuery>
1870   </AdhocQueryRequest>
1871
```

1872 8.2.3 RegistryEntryQuery

1873 Purpose

1874 To identify a set of registry entry instances as the result of a query over selected registry
1875 metadata.

1876
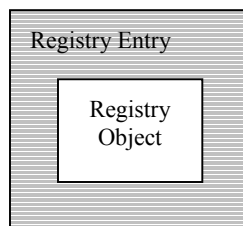
1877 ebRIM Binding

1878 **Figure 17: ebRIM Binding for RegistryEntryQuery**

1879 Definition

```
1880
1881   <complexType name="RegistryEntryQueryType">
1882       <complexContent>
1883           <extension base="tns:RegistryObjectQueryType">
1884               <sequence>
1885                   <element ref="tns:RegistryEntryFilter" minOccurs="0" maxOccurs="1" />
1886               </sequence>
```

```
1887              </extension>
1888          </complexContent>
1889      </complexType>
1890      <element name="RegistryEntryQuery" type="tns:RegistryEntryQueryType" />
1891
1892      <element name="RegistryEntryQueryResult">
1893          <complexType>
1894              <choice minOccurs="0" maxOccurs="unbounded">
1895                  <element ref="rim:ObjectRef" />
1896                  <element ref="rim:ClassificationScheme" />
1897                  <element ref="rim:ExtrinsicObject" />
1898                  <element ref="rim:RegistryEntry" />
1899                  <element ref="rim:RegistryObject" />
1900                  <element ref="rim:RegistryPackage" />
1901              </choice>
1902          </complexType>
1903      </element>
1904
```

Semantic Rules

1906 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
1907   steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.

1908   a) If RE is empty then continue to the next numbered rule.

1909   b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
1910   registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
1911   If RE is empty then continue to the next numbered rule.

1912   c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
1913   RegistryObjectQuery over RE as explained in Section 8.2.2.

1914 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, set RE
1915   to be the result of the RegistryEntryQuery.

1916 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
1917   within the RegistryResponse.

Examples

1919 A client wishes to establish a trading relationship with XYZ Corporation and wants to know if
1920 they have registered any of their business documents in the Registry. The following query
1921 returns a set of registry entry identifiers for currently registered items submitted by any
1922 organization whose name includes the string "XYZ". It does not return any registry entry
1923 identifiers for superseded, replaced, deprecated, or withdrawn items.

```
1925  <AdhocQueryRequest>
1926    <ResponseOption returnType = "ObjectRef"/>
1927    <FilterQuery>
1928      <RegistryEntryQuery>
1929        <TargetAssociationBranch>
1930          <AssociationFilter>
1931            <Clause>
1932              <SimpleClause leftArgument = "associationType">
1933                <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
```

```
1934                </SimpleClause>
1935              </Clause>
1936            </AssociationFilter>
1937            <OrganizationQuery>
1938              <NameBranch>
1939                <LocalizedStringFilter>
1940                  <Clause>
1941                    <SimpleClause leftArgument = "value">
1942                      <StringClause stringPredicate = "Contains">XYZ</StringClause>
1943                    </SimpleClause>
1944                  </Clause>
1945                </LocalizedStringFilter>
1946              </NameBranch>
1947            </OrganizationQuery>
1948          </TargetAssociationBranch>
1949          <RegistryEntryFilter>
1950            <Clause>
1951              <SimpleClause leftArgument = "status">
1952                <StringClause stringPredicate = "Equal">Approved</StringClause>
1953              </SimpleClause>
1954            </Clause>
1955          </RegistryEntryFilter>
1956        </RegistryEntryQuery>
1957      </FilterQuery>
1958    </AdhocQueryRequest>
1959
```

1960 A client is using the United Nations Standard Product and Services Classification (UNSPSC)
1961 scheme and wants to identify all companies that deal with products classified as "Integrated
1962 circuit components", i.e. UNSPSC code "321118". The client knows that companies have
1963 registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
1964 such profile has been classified by UNSPSC according to the products the company deals with.
1965 However, the client does not know if the UNSPSC classification scheme is internal or external to
1966 this registry. The following query returns a set of approved registry entry instances for CPP's of
1967 companies that deal with integrated circuit components.
1968

```
1969    <AdhocQueryRequest>
1970      <ResponseOption returnType = "RegistryEntry"/>
1971      <FilterQuery>
1972        <RegistryEntryQuery>
1973          <ClassifiedByBranch>
1974            <ClassificationFilter>
1975              <Clause>
1976                <SimpleClause leftArgument = "code">
1977                  <StringClause stringPredicate = "Equal">321118</StringClause>
1978                </SimpleClause>
1979              </Clause>
1980            </ClassificationFilter>
1981            <ClassificationSchemeQuery>
1982              <NameBranch>
1983                <LocalizedStringFilter>
1984                  <Clause>
1985                    <SimpleClause leftArgument = "value">
1986                      <StringClause stringPredicate = "Equal">urn:org:un:spsc:cs2001</StringClause>
1987                    </SimpleClause>
1988                  </Clause>
```

```
1989              </LocalizedStringFilter>
1990            </NameBranch>
1991          </ClassificationSchemeQuery>
1992        </ClassifiedByBranch>
1993        <RegistryEntryFilter>
1994          <Clause>
1995            <CompoundClause connectivePredicate = "And">
1996              <Clause>
1997                <SimpleClause leftArgument = "objectType">
1998                  <StringClause stringPredicate = "Equal">CPP</StringClause>
1999                </SimpleClause>
2000              </Clause>
2001              <Clause>
2002                <SimpleClause leftArgument = "status">
2003                  <StringClause stringPredicate = "Equal">Approved</StringClause>
2004                </SimpleClause>
2005              </Clause>
2006            </CompoundClause>
2007          </Clause>
2008        </RegistryEntryFilter>
2009      </RegistryEntryQuery>
2010    </FilterQuery>
2011  </AdhocQueryRequest>
2012
```

2013   8.2.4 AssociationQuery

2014 **Purpose**

2015 To identify a set of association instances as the result of a query over selected registry metadata.

2016
2017 **ebRIM Binding**
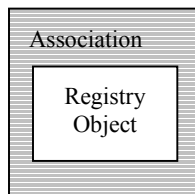
2018   **Figure 18: ebRIM Binding for AssociationQuery**

2019 **Definition**
2020
```
2021  <complexType name = "AssociationQueryType">
2022    <complexContent>
2023      <extension base = "tns:RegistryObjectQueryType">
2024        <sequence>
2025          <element ref = "tns:AssociationFilter" minOccurs = "0" maxOccurs = "1"/>
2026        </sequence>
2027      </extension>
2028    </complexContent>
2029  </complexType>
2030  <element name = "AssociationQuery" type = "tns:AssociationQueryType"/>
2031
2032  <element name="AssociationQueryResult">
2033    <complexType>
```

```
2034        <choice minOccurs="0" maxOccurs="unbounded">
2035          <element ref="rim:ObjectRef" />
2036          <element ref="rim:RegistryObject" />
2037          <element ref="rim:Association" />
2038        </choice>
2039      </complexType>
2040    </element>
2041
```

2042   **Semantic Rules**

2043   1. Let A denote the set of all persistent Association instances in the Registry. The following
2044   steps will eliminate instances in A that do not satisfy the conditions of the specified filters.

2045   a) If A is empty then continue to the next numbered rule.

2046   b) If an AssociationFilter element is not directly contained in the AssociationQuery element,
2047   then go to the next step; otherwise let x be an association instance in A. If x does not
2048   satisfy the AssociationFilter then remove x from A. If A is empty then continue to the
2049   next numbered rule.

2050   c) Let A be the set of remaining Association instances. Evaluate inherited
2051   RegistryObjectQuery over A as explained in Section 8.2.2.

2052   2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to
2053   be the result of the AssociationQuery.

2054   3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2055   within the RegistryResponse.

2056   **Examples**

2057   A client application wishes to identify a set of associations that are 'equivalentTo' a set of other
2058   associations.
2059

```
2060    <AdhocQueryRequest">
2061      <ResponseOption returnType="LeafClass" />
2062      <FilterQuery>
2063        <AssociationQuery>
2064          <SourceAssociationBranch>
2065            <AssociationFilter>
2066              <Clause>
2067                <SimpleClause leftArgument="associationType">
2068                  <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
2069                </SimpleClause>
2070              </Clause>
2071            </AssociationFilter>
2072            <AssociationQuery>
2073              <AssociationFilter>
2074                <Clause>
2075                  <SimpleClause leftArgument="associationType">
2076                    <StringClause stringPredicate="StartsWith">Sin</StringClause>
2077                  </SimpleClause>
2078                </Clause>
2079              </AssociationFilter>
2080            </AssociationQuery>
2081          </SourceAssociationBranch>
2082          <AssociationFilter>
```

```
2083            <Clause>
2084              <SimpleClause leftArgument="associationType">
2085                <StringClause stringPredicate="StartsWith">Son</StringClause>
2086              </SimpleClause>
2087            </Clause>
2088          </AssociationFilter>
2089        </AssociationQuery>
2090      </FilterQuery>
2091  </AdhocQueryRequest>
2092
```

2093 8.2.5 AuditableEventQuery

2094 **Purpose**

2095 To identify a set of auditable event instances as the result of a query over selected registry
2096 metadata.

2097 **ebRIM Binding**
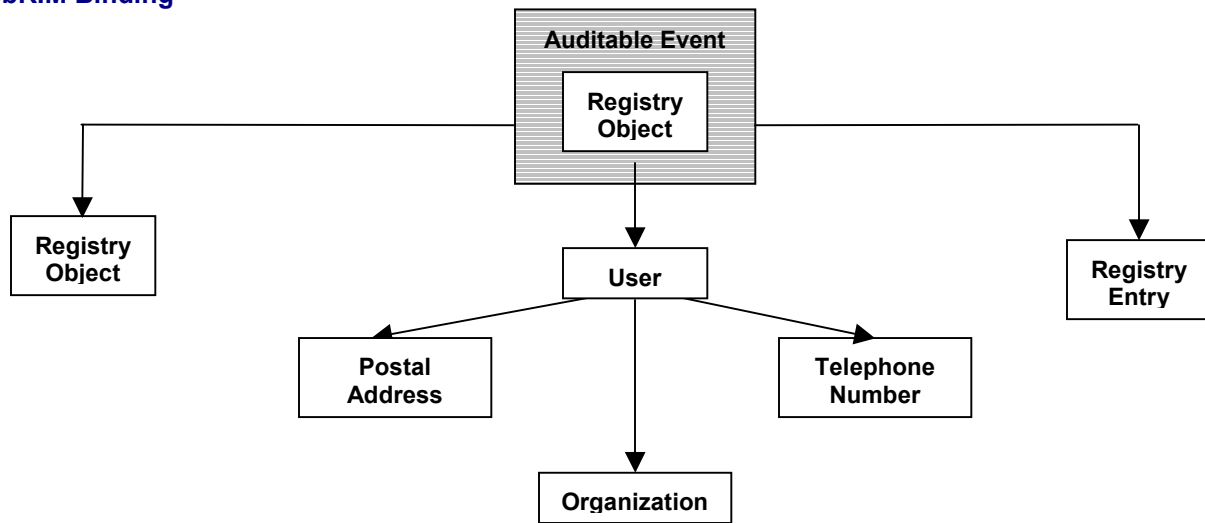
2098   **Figure 19: ebRIM Binding for AuditableEventQuery**

2099 **Definition**

```
2100
2101  <complexType name="AuditableEventQueryType">
2102    <complexContent>
2103      <extension base="tns:RegistryObjectQueryType">
2104        <sequence>
2105          <element ref="tns:AuditableEventFilter" minOccurs="0" />
2106          <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
2107          <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
2108          <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
2109        </sequence>
2110      </extension>
2111    </complexContent>
2112  </complexType>
2113  <element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />
2114
2115  <element name="AuditableEventQueryResult">
2116    <complexType>
```

```
2117        <choice minOccurs="0" maxOccurs="unbounded">
2118          <element ref="rim:ObjectRef" />
2119          <element ref="rim:RegistryObject" />
2120          <element ref="rim:AuditableEvent" />
2121        </choice>
2122      </complexType>
2123  </element>
2124
```

Semantic Rules

2125

2126 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
2127   following steps will eliminate instances in AE that do not satisfy the conditions of the
2128   specified filters.

2129   a) If AE is empty then continue to the next numbered rule.

2130   b) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an
2131   auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from
2132   AE. If AE is empty then continue to the next numbered rule.

2133   c) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let
2134   x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows:
2135   Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is
2136   not an auditable event for some registry object in RO, then remove x from AE. If AE is
2137   empty then continue to the next numbered rule.

2138   d) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x
2139   be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let
2140   RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an
2141   auditable event for some registry entry in RE, then remove x from AE. If AE is empty
2142   then continue to the next numbered rule.

2143   e) If a UserBranch element is not specified then go to the next step; otherwise, let x be a
2144   remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilter
2145   element is specified within the UserBranch, and if u does not satisfy that filter, then
2146   remove x from AE. If a PostalAddressFilter element is specified within the UserBranch,
2147   and if the postal address of u does not satisfy that filter, then remove x from AE. If
2148   TelephoneNumberFilter(s) are specified within the UserBranch and if any of the
2149   TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then
2150   remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if
2151   any of the EmailAddressFilters isn't satisfied by some of the email addresses of u then
2152   remove x from AE. If an OrganizationQuery element is specified within the UserBranch,
2153   then let o be the Organization instance that is identified by the organization that u is
2154   affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then
2155   remove x from AE. If AE is empty then continue to the next numbered rule.

2156   f) Let AE be the set of remaining AuditableEvent instances. Evaluate inherited
2157   RegistryObjectQuery over AE as explained in Section 8.2.2.

2158 2. If AE is empty, then raise the warning: ***auditable event query result is empty***; otherwise set
2159   AE to be the result of the AuditableEventQuery.

2160 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2161   within the RegistryResponse.

2162   **Examples**

2163   A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The
2164   client is now interested in all events since the beginning of the year that have impacted that item.
2165   The following query will return a set of AuditableEvent instances for all such events.
2166
```
2167   <AdhocQueryRequest>
2168     <ResponseOption returnType = "LeafClass"/>
2169     <FilterQuery>
2170       <AuditableEventQuery>
2171         <AuditableEventFilter>
2172           <Clause>
2173             <SimpleClause leftArgument = "timestamp">
2174               <RationalClause logicalPredicate = "GE">
2175                 DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2176               </RationalClause>
2177             </SimpleClause>
2178           </Clause>
2179         </AuditableEventFilter>
2180         <RegistryEntryQuery>
2181           <NameBranch>
2182             <LocalizedStringFilter>
2183               <Clause>
2184                 <SimpleClause leftArgument = "value">
2185                   <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
2186                 </SimpleClause>
2187               </Clause>
2188             </LocalizedStringFilter>
2189           </NameBranch>
2190         </RegistryEntryQuery>
2191       </AuditableEventQuery>
2192     </FilterQuery>
2193   </AdhocQueryRequest>
```
2194

2195   A client company has many registered objects in the Registry. The Registry allows events
2196   submitted by other organizations to have an impact on your registered items, e.g. new
2197   classifications and new associations. The following query will return a set of identifiers for all
2198   auditable events, invoked by some other party, that had an impact on an item submitted by
2199   "myorg".
2200
```
2201   <AdhocQueryRequest>
2202     <ResponseOption returnType = "LeafClass"/>
2203     <FilterQuery>
2204       <AuditableEventQuery>
2205         <RegistryEntryQuery>
2206           <TargetAssociationBranch>
2207             <AssociationFilter>
2208               <Clause>
2209                 <SimpleClause leftArgument = "associationType">
2210                   <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2211                 </SimpleClause>
2212               </Clause>
2213             </AssociationFilter>
2214             <OrganizationQuery>
2215               <NameBranch>
2216                 <LocalizedStringFilter>
```

```
2217                    <Clause>
2218                     <SimpleClause leftArgument = "value">
2219                     <StringClause stringPredicate = "Equal">myorg</StringClause>
2220                     </SimpleClause>
2221                    </Clause>
2222                   </LocalizedStringFilter>
2223                  </NameBranch>
2224                 </OrganizationQuery>
2225               </TargetAssociationBranch>
2226           </RegistryEntryQuery>
2227           <UserBranch>
2228             <OrganizationQuery>
2229               <NameBranch>
2230                 <LocalizedStringFilter>
2231                   <Clause>
2232                     <SimpleClause leftArgument = "value">
2233                       <StringClause stringPredicate = "-Equal">myorg</StringClause>
2234                     </SimpleClause>
2235                   </Clause>
2236                 </LocalizedStringFilter>
2237               </NameBranch>
2238             </OrganizationQuery>
2239           </UserBranch>
2240         </AuditableEventQuery>
2241      </FilterQuery>
2242  </AdhocQueryRequest>
2243
```

2244 8.2.6 ClassificationQuery

2245 **Purpose**

2246 To identify a set of classification instances as the result of a query over selected registry
2247 metadata.

2248 **ebRIM Binding**
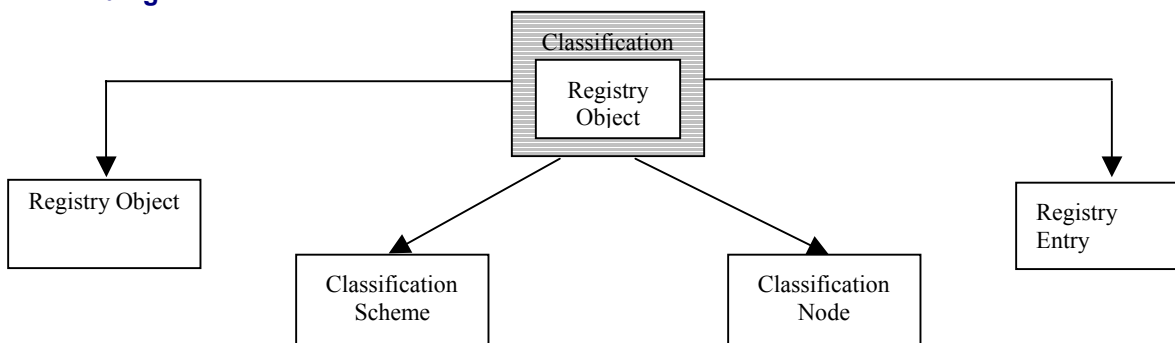
2249   **Figure 20: ebRIM Binding for ClassificationQuery**

2250 **Definition**
2251
```
2252  <complexType name = "ClassificationQueryType">
2253    <complexContent>
2254      <extension base = "tns:RegistryObjectQueryType">
2255        <sequence>
2256          <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
```

```
2257          <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
2258          <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
2259          <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
2260          <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
2261        </sequence>
2262      </extension>
2263    </complexContent>
2264  </complexType>
2265  <element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>
2266
2267  <element name="ClassificationQueryResult">
2268    <complexType>
2269      <choice minOccurs="0" maxOccurs="unbounded">
2270        <element ref="rim:ObjectRef" />
2271        <element ref="rim:RegistryObject" />
2272        <element ref="rim:Classification" />
2273      </choice>
2274    </complexType>
2275  </element>
2276
```

Semantic Rules

2277

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

   a) If C is empty then continue to the next numbered rule.

   b) If a ClassificationFilter element is not directly contained in the ClassificationQuery element, then go to the next step; otherwise let x be an classification instance in C. If x does not satisfy the ClassificationFilter then remove x from C. If C is empty then continue to the next numbered rule.

   c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C. If C is empty then continue to the next numbered rule.

   d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the classification node of x does not satisfy the ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is empty then continue to the next numbered rule.

   e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a classification of at least one registry object in RO, then remove x from C. If C is empty then continue to the next numbered rule.

   f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a classification of at least one registry entry in RE, then remove x from C. If C is empty then continue to the next numbered rule.

2303   2. If C is empty, then raise the warning: *classification query result is empty*; otherwise
2304   otherwise, set C to be the result of the ClassificationQuery.

2305   3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2306   within the RegistryResponse.

8.2.7 ClassificationNodeQuery

Purpose

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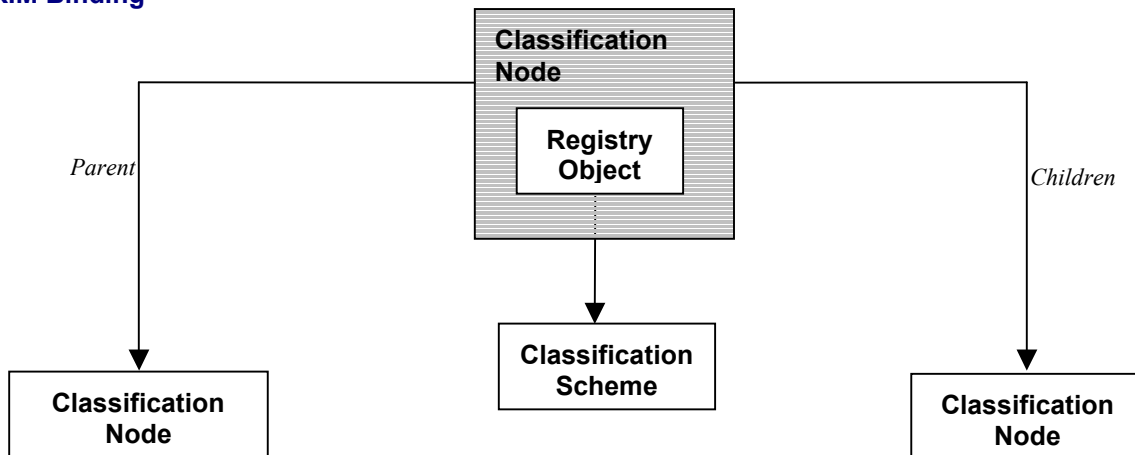
ebRIM Binding

Figure 21: ebRIM Binding for ClassificationNodeQuery

Definition

```
<complexType name="ClassificationNodeQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:ClassificationNodeFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:ClassificationSchemeQuery" minOccurs="0" maxOccurs="1" />
        <element name="ClassificationNodeParentBranch" type="ClassificationNodeQueryType" minOccurs="0"
          maxOccurs="1" />
        <element name="ClassificationNodeChildrenBranch" type="ClassificationNodeQueryType"
          minOccurs="0" maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType" />

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

```
2338      </complexType>
2339    </element>
2340
```

2341   **Semantic Rules**

2342   1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The
2343   following steps will eliminate instances in CN that do not satisfy the conditions of the
2344   specified filters.

2345   a) If CN is empty then continue to the next numbered rule.

2346   b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a
2347   classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove
2348   x from CN. If CN is empty then continue to the next numbered rule.

2349   c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x
2350   be a remaining classification node in CN. If the defining classification scheme of x does
2351   not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x
2352   from CN. If CN is empty then continue to the next numbered rule.

2353   d) If a ClassificationNodeParentBranch element is not specified, then go to the next step;
2354   otherwise, let x be a remaining classification node in CN and execute the following
2355   paragraph with n=x.
2356   Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base
2357   level node), then remove x from CN and go to the next step; otherwise, let p be the parent
2358   node of n. If a ClassificationNodeFilter element is directly contained in the
2359   ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter,
2360   then remove x from CN. If CN is empty then continue to the next numbered rule. If a
2361   ClassificationSchemeQuery element is directly contained in the
2362   ClassificationNodeParentBranch and if defining classification scheme of p does not
2363   satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then
2364   continue to the next numbered rule.
2365   If another ClassificationNodeParentBranch element is directly contained within this
2366   ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.

2367   e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next
2368   numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the
2369   parent node of some ClassificationNode instance, then remove x from CN and if CN is
2370   empty continue to the next numbered rule; otherwise, treat each
2371   ClassificationNodeChildrenBranch element separately and execute the following
2372   paragraph with n = x.

2373   Let n be a classification node instance. If a ClassificationNodeFilter element is not
2374   specified within the ClassificationNodeChildrenBranch element then let CNC be the set
2375   of all classification nodes that have n as their parent node; otherwise, let CNC be the set
2376   of all classification nodes that satisfy the ClassificationNodeFilter and have n as their
2377   parent node. If CNC is empty, then remove x from CN and if CN is empty continue to the
2378   next numbered rule; otherwise, let c be any member of CNC. If a
2379   ClassificationSchemeQuery element is directly contained in the
2380   ClassificationNodeChildrenBranch and if the defining classification scheme of c does not
2381   satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then
2382   remove x from CN. If CN is empty then continue to the next numbered rule; otherwise,
2383   let y be an element of CNC and continue with the next paragraph.
2384   If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly
2385   contain another ClassificationNodeChildrenBranch element, then continue to the next
2386   numbered rule; otherwise, repeat the previous paragraph with the new
2387   ClassificationNodeChildrenBranch element and with n = y.

2388   f)   Let CN be the set of remaining ClassificationNode instances. Evaluate inherited
2389   RegistryObjectQuery over CN as explained in Section 8.2.2.

2390 2. If CN is empty, then raise the warning: **classification node query result is empty**; otherwise
2391   set CN to be the result of the ClassificationNodeQuery.

2392 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2393   within the RegistryResponse.

2394 **Path Filter Expression usage in ClassificationNodeFilter**

2395 The path filter expression is used to match classification nodes in ClassificationNodeFilter
2396 elements involving the path attribute of the ClassificationNode class as defied by the getPath
2397 method in [ebRIM].

2398 The path filter expressions are based on a very small and proper sub-set of location path syntax
2399 of XPath.

2400 The path filter expression syntax includes support for matching multiple nodes by using wild
2401 card syntax as follows:
2402 •   Use of '*' as a wildcard in place of any path element in the pathFilter
2403 •   Use of '//' syntax to denote any descendent of a node in the pathFilter

2404 It is defined by the following BNF grammar:
2405
2406
2407
2408
2409
2410
2411

```
pathFilter      ::= '/' schemeId nodePath
nodePath        ::= slashes nodeCode
                |    slashes '*'
                |    slashes nodeCode ( nodePath )?
Slashes ::= '/' | '//'
```

2412 In the above grammer, schemeId is the id attribute of the ClassificationScheme instance. In the
2413 above grammar nodeCode is defined by NCName production as defined by
2414 http://www.w3.org/TR/REC-xml-names/#NT-NCName.

2415 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a
2416 filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause
2417 is a PATH Filter expression.

2418 This is illustrated in the following example that matches all second level nodes in
2419 ClassificationScheme with id 'Geography-id' and with code 'Japan':

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

2431 **Use Cases and Examples of Path Filter Expressions**

2432 The following table lists various use cases and examples using the sample Geography scheme
2433 below:

```
<ClassificationScheme id='Geography-id' name="Geography"/>

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

<ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
<ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
<ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

2444 **Table 10: Path Filter Expressions for Use Cases**

Use Case	PATH Expression	Description
Match all nodes in first level that have a specified value	/Geography-id/NorthAmerica	Find all first level nodes whose code is 'NorthAmerica'
Find all children of first level node whose code is "NorthAmerica"	/Geography-id/NorthAmerica/*	Match all nodes whose first level path element has code "NorthAmerica"
Match all nodes that have a specified value regardless of level	/ Geography-id//Japan	Find all nodes with code "Japan"
Match all nodes in the second level that have a specified value	/Geography-id/*/Japan	Find all second level nodes with code 'Japan'
Match all nodes in the 3rd level that have a specified value	/ Geography-id/*/*/Tokyo	Find all third level nodes with code 'Tokyo'

2445 **Examples**

2446 A client application wishes to identify all of the classification nodes in the first three levels of a
2447 classification scheme hierarchy. The client knows that the name of the underlying classification

2448 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three
2449 levels.
2450
```
2451  <AdhocQueryRequest>
2452    <ResponseOption returnType = "LeafClass"/>
2453    <FilterQuery>
2454      <ClassificationNodeQuery>
2455        <ClassificationNodeFilter>
2456          <Clause>
2457            <SimpleClause leftArgument = "levelNumber">
2458              <RationalClause logicalPredicate = "LE">
2459               <IntClause>3</IntClause>
2460              </RationalClause>
2461            </SimpleClause>
2462          </Clause>
2463        </ClassificationNodeFilter>
2464        <ClassificationSchemeQuery>
2465          <NameBranch>
2466            <LocalizedStringFilter>
2467              <Clause>
2468                <SimpleClause leftArgument = "value">
2469                 <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
2470                </SimpleClause>
2471              </Clause>
2472            </LocalizedStringFilter>
2473          </NameBranch>
2474        </ClassificationSchemeQuery>
2475      </ClassificationNodeQuery>
2476    </FilterQuery>
2477  </AdhocQueryRequest>
```
2478

2479 If, instead, the client wishes all levels returned, they could simply delete the
2480 ClassificationNodeFilter element from the query.

2481 The following query finds all children nodes of a first level node whose code is NorthAmerica.
2482
```
2483  <AdhocQueryRequest>
2484    <ResponseOption returnType = "LeafClass"/>
2485    <FilterQuery>
2486      <ClassificationNodeQuery>
2487        <ClassificationNodeFilter>
2488          <Clause>
2489            <SimpleClause leftArgument = "path">
2490              <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
2491            </SimpleClause>
2492          </Clause>
2493        </ClassificationNodeFilter>
2494      </ClassificationNodeQuery>
2495    </FilterQuery>
2496  </AdhocQueryRequest>
```
2497

2498 The following query finds all third level nodes with code of Tokyo.
2499
```
2500  <AdhocQueryRequest>
2501    <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2502    <FilterQuery>
```

```
2503        <ClassificationNodeQuery>
2504          <ClassificationNodeFilter>
2505            <Clause>
2506              <SimpleClause leftArgument = "path">
2507               <StringClause stringPredicate = "Equal">/Geography-id/*/*/Tokyo</StringClause>
2508              </SimpleClause>
2509            </Clause>
2510          </ClassificationNodeFilter>
2511        </ClassificationNodeQuery>
2512      </FilterQuery>
2513  </AdhocQueryRequest>
2514
```

2515 8.2.8 ClassificationSchemeQuery

2516 **Purpose**

2517 To identify a set of classification scheme instances as the result of a query over selected registry
2518 metadata.

2519 **ebRIM Binding**
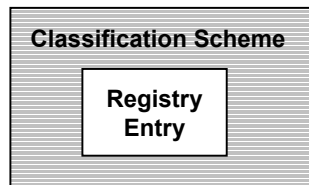
2520   **Figure 22: ebRIM Binding for ClassificationSchemeQuery**

2521 **Definition**
2522
```
2523  <complexType name="ClassificationSchemeQueryType">
2524    <complexContent>
2525      <extension base="tns:RegistryEntryQueryType">
2526        <sequence>
2527          <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" />
2528        </sequence>
2529      </extension>
2530    </complexContent>
2531  </complexType>
2532  <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
2533
```

2534 **Semantic Rules**

2535 1. Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The
2536   following steps will eliminate instances in CS that do not satisfy the conditions of the
2537   specified filters.

2538   a) If CS is empty then continue to the next numbered rule.

2539   b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x
2540   be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,
2541   then remove x from CS. If CS is empty then continue to the next numbered rule.

2542 c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited
2543   RegistryEntryQuery over CS as explained in Section 8.2.3.

2544 2. If CS is empty, then raise the warning: *classification scheme query result is empty*; otherwise,
2545   set CS to be the result of the ClassificationSchemeQuery.

2546 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2547   within the RegistryResponse.

2548 **Examples**

2549 A client application wishes to identify all classification scheme instances in the Registry.
2550 `<AdhocQueryRequest>`
2551 `  <ResponseOption returnType = "LeafClass"/>`
2552 `  <FilterQuery>`
2553 `    <ClassificationSchemeQuery/>`
2554 `  </FilterQuery>`
2555 `</AdhocQueryRequest>`

2556

2557 ## 8.2.9 RegistryPackageQuery

2558 **Purpose**

2559 To identify a set of registry package instances as the result of a query over selected registry
2560 metadata.

2561 **ebRIM Binding**
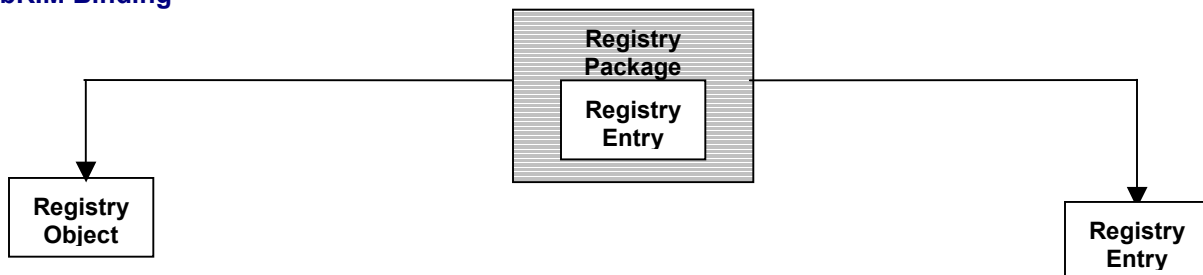
2562 **Figure 23: ebRIM Binding for RegistryPackageQuery**

2563 **Definition**
2564
2565 `<complexType name="RegistryPackageQueryType">`
2566 `  <complexContent>`
2567 `    <extension base="tns:RegistryEntryQueryType">`
2568 `      <sequence>`
2569 `        <element ref="tns:RegistryPackageFilter" minOccurs="0" maxOccurs="1" />`
2570 `        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="unbounded" />`
2571 `        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="unbounded" />`
2572 `      </sequence>`
2573 `    </extension>`
2574 `  </complexContent>`
2575 `</complexType>`
2576 `<element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType" />`
2577
2578 `<element name="RegistryPackageQueryResult">`

```
2579        <complexType>
2580          <choice minOccurs="0" maxOccurs="unbounded">
2581            <element ref="rim:ObjectRef" />
2582            <element ref="rim:RegistryEntry" />
2583            <element ref="rim:RegistryObject" />
2584            <element ref="rim:RegistryPackage" />
2585          </choice>
2586        </complexType>
2587      </element>
2588
```

2589   **Semantic Rules**

2590   1. Let RP denote the set of all persistent RegistryPackage instances in the Registry. The
2591   following steps will eliminate instances in RP that do not satisfy the conditions of the
2592   specified filters.

2593   a) If RP is empty then continue to the next numbered rule.

2594   b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule;
2595   otherwise, let x be a registry package instance in RP. If x does not satisfy the
2596   RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next
2597   numbered rule.

2598   c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery
2599   element then treat each RegistryObjectQuery as follows: let RO be the set of
2600   RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2
2601   and let PO be the subset of RO that are members of the package x. If PO is empty, then
2602   remove x from RP. If RP is empty then continue to the next numbered rule. If a
2603   RegistryEntryQuery element is directly contained in the RegistryPackageQuery element
2604   then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry
2605   instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be
2606   the subset of RE that are members of the package x. If PE is empty, then remove x from
2607   RP. If RP is empty then continue to the next numbered rule.

2608   d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited
2609   RegistryEntryQuery over RP as explained in Section 8.2.3.

2610   2. If RP is empty, then raise the warning: ***registry package query result is empty***; otherwise set
2611   RP to be the result of the RegistryPackageQuery.

2612   3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2613   within the RegistryResponse.

2614   **Examples**

2615   A client application wishes to identify all package instances in the Registry that contain an
2616   Invoice extrinsic object as a member of the package.
2617

```
2618    <AdhocQueryRequest>
2619      <ResponseOption returnType = "LeafClass"/>
2620      <FilterQuery>
2621        <RegistryPackageQuery>
2622          <RegistryEntryQuery>
2623            <RegistryEntryFilter>
2624              <Clause>
```

```
2625            <SimpleClause leftArgument = "objectType">
2626              <StringClause stringPredicate = "Equal">Invoice</StringClause>
2627            </SimpleClause>
2628          </Clause>
2629        </RegistryEntryFilter>
2630      </RegistryEntryQuery>
2631    </RegistryPackageQuery>
2632  </FilterQuery>
2633 </AdhocQueryRequest>
2634
```

2635 A client application wishes to identify all package instances in the Registry that are not empty.
2636

```
2637 <AdhocQueryRequest>
2638   <ResponseOption returnType = "LeafClass"/>
2639   <FilterQuery>
2640     <RegistryPackageQuery>
2641       <RegistryObjectQuery/>
2642     </RegistryPackageQuery>
2643   </FilterQuery>
2644 </AdhocQueryRequest>
2645
```

2646 A client application wishes to identify all package instances in the Registry that are empty. Since
2647 the RegistryPackageQuery is not set up to do negations, clients will have to do two separate
2648 RegistryPackageQuery requests, one to find all packages and another to find all non-empty
2649 packages, and then do the set difference themselves. Alternatively, they could do a more
2650 complex RegistryEntryQuery and check that the packaging association between the package and
2651 its members is non-existent.

2652 Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by
2653 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an
2654 equivalent RegistryEntryQuery using appropriate "Source" and "Target" associations. However,
2655 the equivalent RegistryEntryQuery is often more complicated to write.

2656 ## 8.2.10 ExtrinsicObjectQuery

2657 **Purpose**

2658 To identify a set of extrinsic object instances as the result of a query over selected registry
2659 metadata.
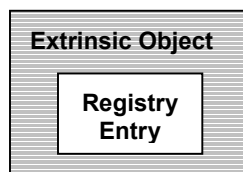
2660 **ebRIM Binding**
2661   **Figure 24: ebRIM Binding for ExtrinsicObjectQuery**

2662 **Definition**
2663

```
2664    <complexType name="ExtrinsicObjectQueryType">
2665      <complexContent>
2666        <extension base="tns:RegistryEntryQueryType">
2667          <sequence>
2668            <element ref="tns:ExtrinsicObjectFilter" minOccurs="0" maxOccurs="1" />
2669          </sequence>
2670        </extension>
2671      </complexContent>
2672    </complexType>
2673    <element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType" />
2674
2675    <element name="ExtrinsicObjectQueryResult">
2676      <complexType>
2677        <choice minOccurs="0" maxOccurs="unbounded">
2678          <element ref="rim:ObjectRef" />
2679          <element ref="rim:RegistryEntry" />
2680          <element ref="rim:RegistryObject" />
2681          <element ref="rim:ExtrinsicObject" />
2682        </choice>
2683      </complexType>
2684    </element>
2685
```

Semantic Rules

2687 1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The
2688   following steps will eliminate instances in EO that do not satisfy the conditions of the
2689   specified filters.

2690   a) If EO is empty then continue to the next numbered rule.

2691   b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an
2692   extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from
2693   EO. If EO is empty then continue to the next numbered rule.

2694   c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited
2695   RegistryEntryQuery over EO as explained in Section 8.2.3.

2696 2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set
2697   EO to be the result of the ExtrinsicObjectQuery.

2698 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2699   within the RegistryResponse.

8.2.11 OrganizationQuery

Purpose

2702 To identify a set of organization instances as the result of a query over selected registry
2703 metadata.
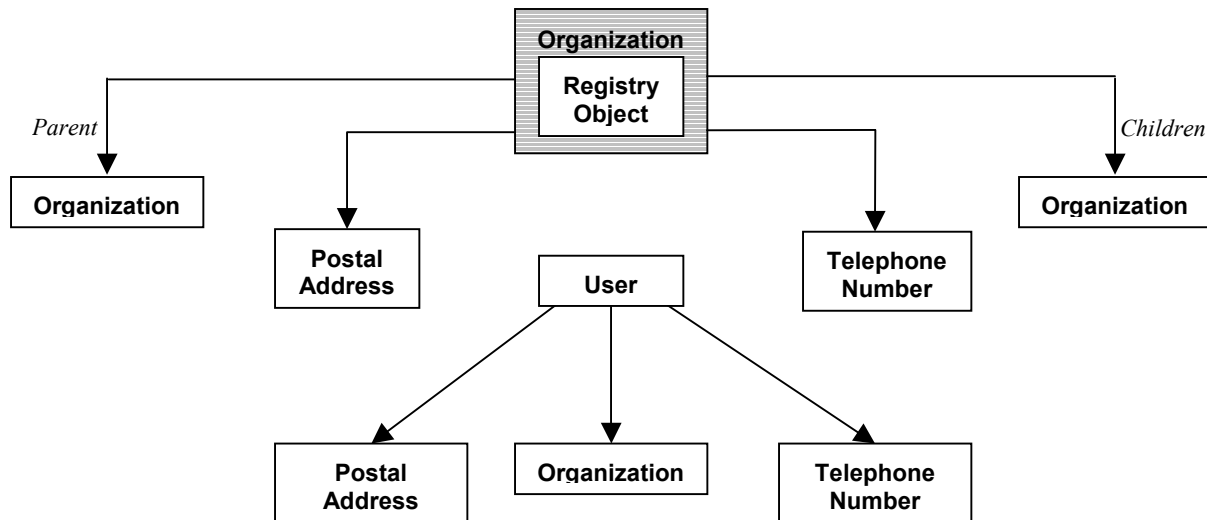
ebRIM Binding

Figure 25: ebRIM Binding for OrganizationQuery

2705

Definition

```
<complexType name="OrganizationQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:OrganizationFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:PostalAddressFilter" minOccurs="0" maxOccurs="1" />
        <element ref="tns:TelephoneNumberFilter" minOccurs="0" maxOccurs="unbounded" />
        <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
        <element name="OrganizationParentBranch" type="tns:OrganizationQueryType" minOccurs="0
          " maxOccurs="1" />
        <element name="OrganizationChildrenBranch" type="tns:OrganizationQueryType" minOccurs="0"
          maxOccurs="unbounded" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="OrganizationQuery" type="tns:OrganizationQueryType" />

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

Semantic Rules

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

   a) If ORG is empty then continue to the next numbered rule.

2741   b) If an OrganizationFilter element is not directly contained in the OrganizationQuery
2742   element, then go to the next step; otherwise let x be an organization instance in ORG. If x
2743   does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then
2744   continue to the next numbered rule.

2745   c) If a PostalAddressFilter element is not directly contained in the OrganizationQuery
2746   element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal
2747   address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is
2748   empty then continue to the next numbered rule.

2749   d) If no TelephoneNumberFilter element is directly contained in the OrganizationQuery
2750   element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of
2751   the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then
2752   remove x from ORG. If ORG is empty then continue to the next numbered rule.

2753   e) If a UserBranch element is not directly contained in the OrganizationQuery element then
2754   go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user
2755   instance that is affiliated with x. If a UserFilter element is specified within the
2756   UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a
2757   PostalAddressFilter element is specified within the UserBranch, and if the postal address
2758   of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s)
2759   are specified within the UserBranch and if any of the TelephoneNumberFilters isn't
2760   satisfied by some of the telephone numbers of x then remove x from ORG. If
2761   EmailAddressFilter(s) are specified within the UserBranch and if any of the
2762   EmailAddressFilters isn't satisfied by some of the email addresses of x then remove x
2763   from ORG. If an OrganizationQuery element is specified within the UserBranch, then let
2764   o be the Organization instance that is identified by the organization that u is affiliated
2765   with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x
2766   from ORG. If ORG is empty then continue to the next numbered rule.

2767   f) If a OrganizationParentBranch element is not specified within the OrganizationQuery,
2768   then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the
2769   following paragraph with o = x:
2770   Let o be an organization instance. If an OrganizationFilter is not specified within the
2771   OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the
2772   Organization hierarchy), then remove x from ORG; otherwise, let p be the parent
2773   organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.
2774   If ORG is empty then continue to the next numbered rule.
2775   If another OrganizationParentBranch element is directly contained within this
2776   OrganizationParentBranch element, then repeat the previous paragraph with o = p.

2777   g) If a OrganizationChildrenBranch element is not specified, then continue to the next
2778   numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent
2779   node of some organization instance, then remove x from ORG and if ORG is empty
2780   continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch
2781   element separately and execute the following paragraph with n = x.

2782   Let n be an organization instance. If an OrganizationFilter element is not specified within
2783   the OrganizationChildrenBranch element then let ORGC be the set of all organizations
2784   that have n as their parent node; otherwise, let ORGC be the set of all organizations that
2785   satisfy the OrganizationFilter and have n as their parent node. If ORGC is empty, then
2786   remove x from ORG and if ORG is empty continue to the next numbered rule; otherwise,
2787   let c be any member of ORGC. If a PostalAddressFilter element is directly contained in
2788   the OrganizationChildrenBranch and if the postal address of c does not satisfy the
2789   PostalAddressFilter then remove c from ORGC. If ORGC is empty then remove x from
2790   ORG. If ORG is empty then continue to the next numbered rule. If no
2791   TelephoneNumberFilter element is directly contained in the OrganizationChildrenBranch
2792   and if If any of the TelephoneNumberFilters isn't satisfied by some of the telephone
2793   numbers of c then remove c from ORGC. If ORGC is empty then remove x from ORG. If
2794   ORG is empty then continue to the next numbered rule; otherwise, let y be an element of
2795   ORGC and continue with the next paragraph.

2796   If the OrganizationChildrenBranch element is terminal, i.e. if it does not directly contain
2797   another OrganizationChildrenBranch element, then continue to the next numbered rule;
2798   otherwise, repeat the previous paragraph with the new OrganizationChildrenBranch
2799   element and with n = y.

2800   h) Let ORG be the set of remaining Organization instances. Evaluate inherited
2801   RegistryObjectQuery over ORG as explained in Section 8.2.2.

2802   2. If ORG is empty, then raise the warning: *organization query result is empty*; otherwise set
2803   ORG to be the result of the OrganizationQuery.

2804   3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2805   within the RegistryResponse.

2806   **Examples**

2807   A client application wishes to identify a set of organizations, based in France, that have
2808   submitted a PartyProfile extrinsic object this year.

```
2810  <AdhocQueryRequest>
2811    <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
2812    <FilterQuery>
2813        <OrganizationQuery>
2814            <SourceAssociationBranch>
2815                <AssociationFilter>
2816                    <Clause>
2817                        <SimpleClause leftArgument = "associationType">
2818                            <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
2819                        </SimpleClause>
2820                    </Clause>
2821                </AssociationFilter>
2822                <RegistryObjectQuery>
2823                    <RegistryObjectFilter>
2824                        <Clause>
2825                            <SimpleClause leftArgument = "objectType">
2826                                <StringClause stringPredicate = "Equal">CPP</StringClause>
2827                            </SimpleClause>
2828                        </Clause>
2829                    </RegistryObjectFilter>
2830                    <AuditableEventQuery>
2831                        <AuditableEventFilter>
2832                            <Clause>
2833                                <SimpleClause leftArgument = "timestamp">
2834                                  <RationalClause logicalPredicate = "GE">
2835                                     <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
2836                                  </RationalClause>
```

```
2837                                          </SimpleClause>
2838                                      </Clause>
2839                                  </AuditableEventFilter>
2840                              </AuditableEventQuery>
2841                          </RegistryObjectQuery>
2842                  </SourceAssociationBranch>
2843                  <PostalAddressFilter>
2844                      <Clause>
2845                          <SimpleClause leftArgument = "country">
2846                              <StringClause stringPredicate = "Equal">France</StringClause>
2847                          </SimpleClause>
2848                      </Clause>
2849                  </PostalAddressFilter>
2850          </OrganizationQuery>
2851      </FilterQuery>
2852  </AdhocQueryRequest>
2853
```

2854   A client application wishes to identify all organizations that have Corporation named XYZ as a
2855   parent.
2856
```
2857  <AdhocQueryRequest>
2858      <ResponseOption returnType = "LeafClass"/>
2859      <FilterQuery>
2860          <OrganizationQuery>
2861              <OrganizationParentBranch>
2862                  <NameBranch>
2863                      <LocalizedStringFilter>
2864                          <Clause>
2865                              <SimpleClause leftArgument = "value">
2866                                  <StringClause stringPredicate = "Equal">XYZ</StringClause>
2867                              </SimpleClause>
2868                          </Clause>
2869                      </LocalizedStringFilter>
2870                  </NameBranch>
2871              </OrganizationParentBranch>
2872          </OrganizationQuery>
2873      </FilterQuery>
2874  </AdhocQueryRequest>
2875
```

2876   ## 8.2.12 ServiceQuery

2877   **Purpose**

2878

2879   To identify a set of service instances as the result of a query over selected registry metadata.
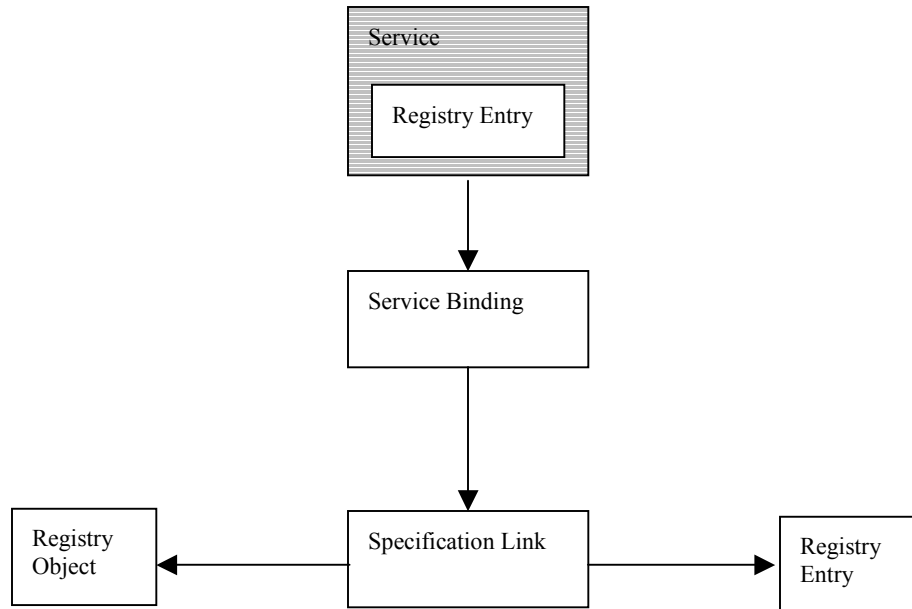
2880   **ebRIM Binding**

Figure 26: ebRIM Binding for ServiceQuery

Definition

```
<complexType name="ServiceQueryType">
   <complexContent>
      <extension base="tns:RegistryEntryQueryType">
         <sequence>
            <element ref="tns:ServiceFilter" minOccurs="0"
               maxOccurs="1" />
            <element ref="tns:ServiceBindingBranch" minOccurs="0"
               maxOccurs="unbounded" />
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name="ServiceQuery" type="tns:ServiceQueryType" />

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

Semantic Rules

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

    a) If S is empty then continue to the next numbered rule.

2912   b) If a ServicetFilter is not specified then go to the next step; otherwise, let x be a service in
2913   S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then
2914   continue to the next numbered rule.

2915   c) If a ServiceBindingBranch is not specified then continue to the next numbered rule;
2916   otherwise, consider each ServiceBindingBranch element separately as follows:
2917   Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the
2918   member of SB. If a ServiceBindingFilter element is specified within the
2919   ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If
2920   SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
2921   If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then
2922   continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch
2923   element separately as follows:
2924   Let sb be a remaining service binding in SB. Let SL be the set of all specification link
2925   instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
2926   specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
2927   remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove
2928   x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery
2929   element is specified within the SpecificationLinkBranch then let sl be a remaining
2930   specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
2931   result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
2932   specification link for some registry object in RO, then remove sl from SL. If SL is empty
2933   then remove sb from SB. If SB is empty then remove x from S. If S is empty then
2934   continue to the next numbered rule. If a RegistryEntryQuery element is specified within
2935   the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
2936   RegistryEntryQuery element as follows: Let RE be the result set of the
2937   RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some
2938   registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If
2939   SB is empty then remove x from S. If S is empty then continue to the next numbered rule.

2940   d) Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery
2941   over AE as explained in Section 8.2.3.

2942 2. If S is empty, then raise the warning: *service query result is empty*; otherwise set S to be the
2943   result of the ServiceQuery.

2944 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
2945   within the RegistryResponse.

Examples

8.2.13 Registry Filters

Purpose

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

Definition

```
<complexType name="FilterType">
```

```
2954        <sequence>
2955            <element ref="tns:Clause" />
2956        </sequence>
2957    </complexType>
2958    <element name="RegistryObjectFilter" type="tns:FilterType" />
2959    <element name="RegistryEntryFilter" type="tns:FilterType" />
2960    <element name="ExtrinsicObjectFilter" type="tns:FilterType" />
2961    <element name="RegistryPackageFilter" type="tns:FilterType" />
2962    <element name="OrganizationFilter" type="tns:FilterType" />
2963    <element name="ClassificationNodeFilter" type="tns:FilterType" />
2964    <element name="AssociationFilter" type="tns:FilterType" />
2965    <element name="ClassificationFilter" type="tns:FilterType" />
2966    <element name="ClassificationSchemeFilter" type="tns:FilterType" />
2967    <element name="ExternalLinkFilter" type="tns:FilterType" />
2968    <element name="ExternalIdentifierFilter" type="tns:FilterType" />
2969    <element name="SlotFilter" type="tns:FilterType" />
2970    <element name="AuditableEventFilter" type="tns:FilterType" />
2971    <element name="UserFilter" type="tns:FilterType" />
2972    <element name="SlotValueFilter" type="tns:FilterType" />
2973    <element name="PostalAddressFilter" type="tns:FilterType" />
2974    <element name="TelephoneNumberFilter" type="tns:FilterType" />
2975    <element name="ServiceFilter" type="tns:FilterType" />
2976    <element name="ServiceBindingFilter" type="tns:FilterType" />
2977    <element name="SpecificationLinkFilter" type="tns:FilterType" />
2978    <element name="LocalizedStringFilter" type="tns:FilterType" />
2979
```

2980   **Semantic Rules**

2981   1. The Clause element is defined in Section 8.2.14.

2982   2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
2983   SimpleClause shall identify a public attribute of the RegistryObject UML class defined in
2984   [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set
2985   of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the
2986   Clause predicate.

2987   3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing
2988   SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in
2989   [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
2990   returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
2991   for the Clause predicate.

2992   4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
2993   SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in
2994   [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter
2995   returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to
2996   *True* for the Clause predicate.

2997   5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing
2998   SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in
2999   [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns
3000   a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for
3001   the Clause predicate.

3002 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing
3003   SimpleClause shall identify a public attribute of the Organization or PostalAddress UML
3004   classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The
3005   OrganizationFilter returns a set of identifiers for Organization instances whose attribute
3006   values evaluate to *True* for the Clause predicate.

3007 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
3008   containing SimpleClause shall identify a public attribute of the ClassificationNode UML
3009   class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the
3010   leftAttribute is the visible attribute "path" then if stringPredicate of the StringClause is not
3011   "Equal" then raise exception: *classification node path attribute error*. The
3012   ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose
3013   attribute values evaluate to *True* for the Clause predicate.

3014 8. For every AssociationFilter XML element, the leftArgument attribute of any containing
3015   SimpleClause shall identify a public attribute of the Association UML class defined in
3016   [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a
3017   set of identifiers for Association instances whose attribute values evaluate to *True* for the
3018   Clause predicate.

3019 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing
3020   SimpleClause shall identify a public attribute of the Classification UML class defined in
3021   [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter
3022   returns a set of identifiers for Classification instances whose attribute values evaluate to *True*
3023   for the Clause predicate.

3024 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any
3025   containing SimpleClause shall identify a public attribute of the ClassificationNode UML
3026   class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The
3027   ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances
3028   whose attribute values evaluate to *True* for the Clause predicate.

3029 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing
3030   SimpleClause shall identify a public attribute of the ExternalLink UML class defined in
3031   [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns
3032   a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the
3033   Clause predicate.

3034 12. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any containing
3035   SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in
3036   [ebRIM]. If not, raise exception: *external identifier attribute error*. The
3037   ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose
3038   attribute values evaluate to *True* for the Clause predicate.

3039 13. For every SlotFilter XML element, the leftArgument attribute of any containing
3040   SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If
3041   not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot
3042   instances whose attribute values evaluate to *True* for the Clause predicate.

3043   14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
3044   SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in
3045   [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter
3046   returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to
3047   *True* for the Clause predicate.

3048   15. For every UserFilter XML element, the leftArgument attribute of any containing
3049   SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If
3050   not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User
3051   instances whose attribute values evaluate to *True* for the Clause predicate.

3052   16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one
3053   SlotValue instance for each "value" in the "values" list of a Slot instance. The visible
3054   attribute of SlotValue is "value". It is a character string. The dynamic instances of SlotValue
3055   are derived from the "values" attribute defined in ebRIM for a Slot instance. For every
3056   SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause
3057   shall identify the "value" attribute of the SlotValue class just defined. If not, raise exception:
3058   *slot element attribute error*. The SlotValueFilter returns a set of Slot instances whose "value"
3059   attribute evaluates to *True* for the Clause predicate.

3060   17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing
3061   SimpleClause shall identify a public attribute of the PostalAddress UML class defined in
3062   [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter
3063   returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*
3064   for the Clause predicate.

3065   18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any
3066   containing SimpleClause shall identify a public attribute of the TelephoneNumber UML
3067   class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.
3068   The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances
3069   whose attribute values evaluate to *True* for the Clause predicate.

3070   19. For every ServiceFilter XML element, the leftArgument attribute of any containing
3071   SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].
3072   If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for
3073   Service instances whose attribute values evaluate to *True* for the Clause predicate.

3074   20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
3075   SimpleClause shall identify a public attribute of the ServiceBinding UML class defined in
3076   [ebRIM]. If not, raise exception: *service binding attribute error*. The ServiceBindingFilter
3077   returns a set of identifiers for ServiceBinding instances whose attribute values evaluate to
3078   *True* for the Clause predicate.

3079   21. For every SpecificationLinkFilter XML element, the leftArgument attribute of any
3080   containing SimpleClause shall identify a public attribute of the SpecificationLink UML class
3081   defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The
3082   SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose
3083   attribute values evaluate to *True* for the Clause predicate.

3084   22. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing
3085   SimpleClause shall identify a public attribute of the LocalizedString UML class defined in
3086   [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter
3087   returns a set of identifiers for LocalizedString instances whose attribute values evaluate to
3088   *True* for the Clause predicate.

8.2.14 XML Clause Constraint Representation

3089

Purpose

3090

3091   The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses.
3092   Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to
3093   simply as Clauses in this specification.

Conceptual Diagram

3094

3095   The following is a conceptual diagram outlining the Clause structure.

3096

3097
3098   **Figure 27: The Clause Structure**
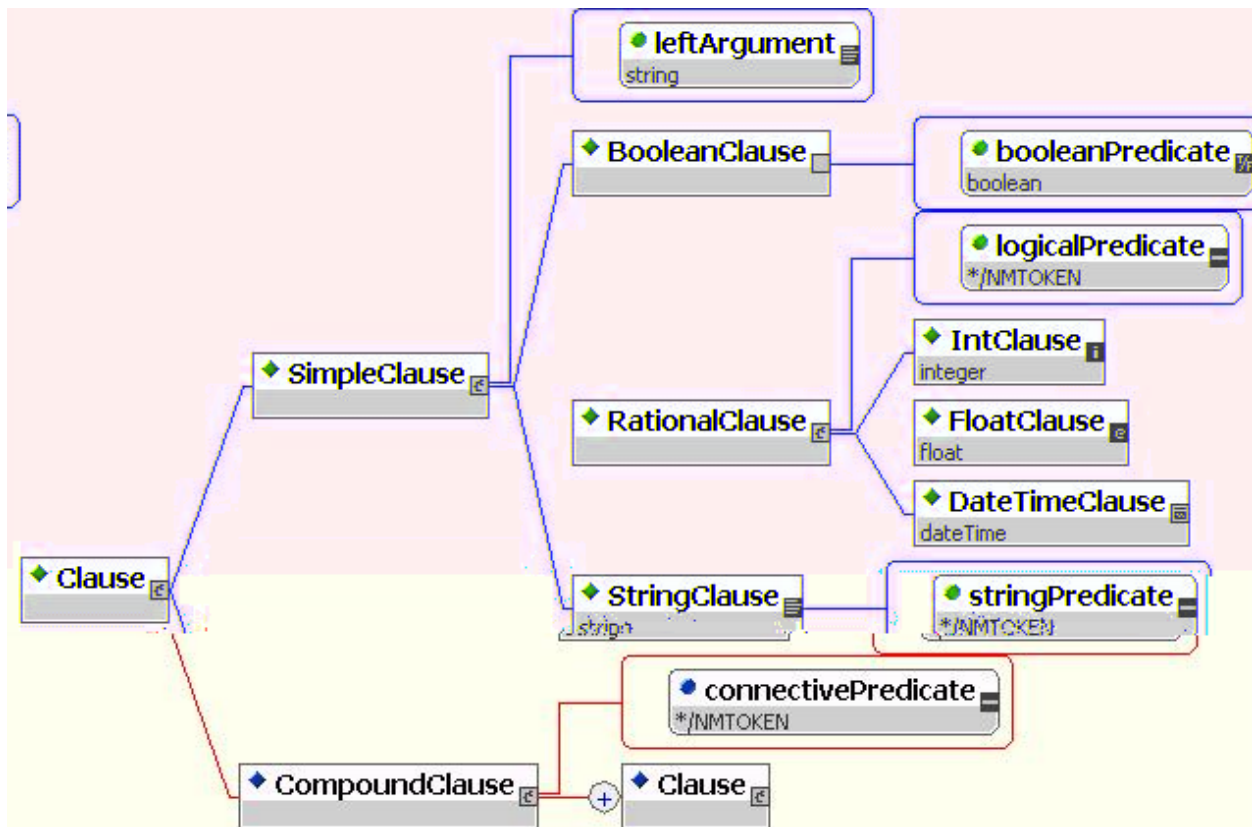
Semantic Rules

3099

3100   Predicates and Arguments are combined into a "LeftArgument - Predicate - RightArgument"
3101   format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.
3102   ***SimpleClauses***

3103   A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the

3104   Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.
3105   SimpleClause is extended to support BooleanClause, StringClause, and RationalClause
3106   (abstract).

3107   BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
3108   boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-
3109   compare operations and a right argument as the element's text data. Rational number support is
3110   provided through a common RationalClause providing an enumeration of appropriate rational
3111   number compare operations, which is further extended to IntClause and FloatClause, each with
3112   appropriate signatures for the right argument.

3113   ***CompoundClauses***

3114   A CompoundClause contains two or more Clauses (Simple or Compound) and a connective
3115   predicate. This provides for arbitrarily complex Clauses to be formed.

3116   **Definition**
3117
```
3118        <element name = "Clause">
3119            <annotation>
3120                <documentation xml:lang = "en">
3121        The following lines define the XML syntax for Clause.
3122
3123                </documentation>
3124            </annotation>
3125            <complexType>
3126                <choice>
3127                    <element ref = "tns:SimpleClause"/>
3128                    <element ref = "tns:CompoundClause"/>
3129                </choice>
3130            </complexType>
3131        </element>
3132        <element name = "SimpleClause">
3133            <complexType>
3134                <choice>
3135                    <element ref = "tns:BooleanClause"/>
3136                    <element ref = "tns:RationalClause"/>
3137                    <element ref = "tns:StringClause"/>
3138                </choice>
3139                <attribute name = "leftArgument" use = "required" type =
3140        "string"/>
3141            </complexType>
3142        </element>
3143        <element name = "CompoundClause">
3144            <complexType>
3145                <sequence>
3146                    <element ref = "tns:Clause" maxOccurs = "unbounded"/>
3147                </sequence>
3148                <attribute name = "connectivePredicate" use = "required">
3149                    <simpleType>
3150                        <restriction base = "NMTOKEN">
3151                            <enumeration value = "And"/>
3152                            <enumeration value = "Or"/>
3153                        </restriction>
3154                    </simpleType>
3155                </attribute>
3156            </complexType>
3157        </element>
```

```
3158        <element name = "BooleanClause">
3159           <complexType>
3160              <attribute name = "booleanPredicate" use = "required" type =
3161    "boolean"/>
3162           </complexType>
3163        </element>
3164        <element name = "RationalClause">
3165           <complexType>
3166              <choice>
3167                 <element ref = "tns:IntClause"/>
3168                 <element ref = "tns:FloatClause"/>
3169                 <element ref = "tns:DateTimeClause"/>
3170              </choice>
3171              <attribute name = "logicalPredicate" use = "required">
3172                 <simpleType>
3173                    <restriction base = "NMTOKEN">
3174                       <enumeration value = "LE"/>
3175                       <enumeration value = "LT"/>
3176                       <enumeration value = "GE"/>
3177                       <enumeration value = "GT"/>
3178                       <enumeration value = "EQ"/>
3179                       <enumeration value = "NE"/>
3180                    </restriction>
3181                 </simpleType>
3182              </attribute>
3183           </complexType>
3184        </element>
3185        <element name = "IntClause" type = "integer"/>
3186        <element name = "FloatClause" type = "float"/>
3187        <element name = "DateTimeClause" type = "dateTime"/>
3188
3189        <element name = "StringClause">
3190           <complexType>
3191              <simpleContent>
3192                 <extension base = "string">
3193                    <attribute name = "stringPredicate" use = "required">
3194                       <simpleType>
3195                          <restriction base = "NMTOKEN">
3196                             <enumeration value = "Contains"/>
3197                             <enumeration value = "-Contains"/>
3198                             <enumeration value = "StartsWith"/>
3199                             <enumeration value = "-StartsWith"/>
3200                             <enumeration value = "Equal"/>
3201                             <enumeration value = "-Equal"/>
3202                             <enumeration value = "EndsWith"/>
3203                             <enumeration value = "-EndsWith"/>
3204                          </restriction>
3205                       </simpleType>
3206                    </attribute>
3207                 </extension>
3208              </simpleContent>
3209           </complexType>
3210        </element>
3211
```

3212   **Examples**

3213   Simple BooleanClause: "Smoker" = True

```
3214
3215    <Clause>
3216       <SimpleClause leftArgument="Smoker">
3217          <BooleanClause booleanPredicate="True"/>
3218       </SimpleClause>
3219    </Clause>
3220
```

Simple StringClause: "Smoker" contains "mo"

```
3221
3222
3223    <Clause>
3224       <SimpleClause leftArgument = "Smoker">
3225          <StringClause stringPredicate = "Contains">mo</StringClause>
3226       </SimpleClause>
3227    <Clause>
```

Simple IntClause: "Age" >= 7

```
3228
3229
3230    <Clause>
3231       <SimpleClause leftArgument="Age">
3232          <RationalClause logicalPredicate="GE">
3233             <IntClause>7</IntClause>
3234          </RationalClause>
3235       </SimpleClause>
3236    </Clause>
3237
```

Simple FloatClause: "Size" = 4.3

```
3238
3239
3240    <Clause>
3241       <SimpleClause leftArgument="Size">
3242          <RationalClause logicalPredicate="Equal">
3243             <FloatClause>4.3</FloatClause>
3244          </RationalClause>
3245       </SimpleClause>
3246    </Clause>
3247
```

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

```
3248
3249
3250    <Clause>
3251       <CompoundClause connectivePredicate="And">
3252          <Clause>
3253             <SimpleClause leftArgument="Smoker">
3254                <BooleanClause booleanPredicate="False"/>
3255             </SimpleClause>
3256          </Clause>
3257          <Clause>
3258             <SimpleClause leftArgument="Age">
3259                <RationalClause logicalPredicate="LE">
3260                   <IntClause>45</IntClause>
3261                </RationalClause>
3262             </SimpleClause>
3263          </Clause>
3264       </CompoundClause>
3265    </Clause>
3266
```

3267 Coumpound with one Simple and one Compound

3268 (("Smoker" = False)And(("Age" =< 45)Or("American"=True)))

```
3270    <Clause>
3271       <CompoundClause connectivePredicate="And">
3272          <Clause>
3273             <SimpleClause leftArgument="Smoker">
3274                <BooleanClause booleanPredicate="False"/>
3275             </SimpleClause>
3276          </Clause>
3277          <Clause>
3278             <CompoundClause connectivePredicate="Or">
3279                <Clause>
3280                   <SimpleClause leftArgument="Age">
3281                      <RationalClause logicalPredicate="LE">
3282                         <IntClause>45</IntClause>
3283                      </RationalClause>
3284                   </SimpleClause>
3285                </Clause>
3286                <Clause>
3287                   <SimpleClause leftArgument="American">
3288                      <BooleanClause booleanPredicate="True"/>
3289                   </SimpleClause>
3290                </Clause>
3291             </CompoundClause>
3292          </Clause>
3293       </CompoundClause>
3294    <Clause>
```

8.3 SQL Query Support

3297 The Registry may optionally support an SQL based query capability that is designed for Registry
3298 clients that demand more advanced query capability. The optional SQLQuery element in the
3299 AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query
3300 language.

3301 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of
3302 the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database
3303 Language SQL [SQL], extended to include `<sql invoked routines>` (also known as
3304 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined
3305 in template form in Appendix D.3. The syntax of the Registry query language is defined by the
3306 BNF grammar in D.1.

3307 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to use
3308 relational databases in a Registry implementation.

8.3.1 SQL Query Syntax Binding To [ebRIM]

3310 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational
3311 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as
3312 described in the following sections.

8.3.1.1   Class Binding

A subset of the class names defined in [ebRIM] map to table names that may be queried by an SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an SQL query.

The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3 is as follows:

- Classes that have concrete instances are mapped to relational tables. In addition entity classes (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- The intermediate classes in the inheritance hierarchy, namely RegistryObject and RegistryEntry, map to relational views.
- The names of relational tables and views are the same as the corresponding [ebRIM] class name. However, the name binding is case insensitive.
- Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in Appendix D.3 where the column definitions are based on a subset of attributes defined for that class in [ebRIM]. The attributes that map to columns include the inherited attributes for the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed which column definitions.

An SQLQuery against a table not defined in Appendix D.3 may raise an error condition: InvalidQueryException.

The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumn definitions.

8.3.1.2   Primitive Attributes Binding

Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same way as column names in SQL. Again the exact attribute names are defined in the class definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is therefore valid for a query to contain attribute names that do not exactly match the case defined in [ebRIM].

8.3.1.3   Reference Attribute Binding

A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject class returns a reference to an instance of an AccessControlPolicy object.

In such cases the reference maps to the `id` attribute for the referenced object. The name of the resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data type for the column is VARCHAR(64) as defined in Appendix D.3.

When a reference attribute value holds a null reference, it maps to a null value in the SQL binding and may be tested with the <null specification> ("IS [NOT] NULL" syntax) as defined by [SQL].

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

8.3.1.4   Complex Attribute Binding

A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of

3353   a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type
3354   TelephoneNumber, Contact, PersonName etc. in class Organization and class User.
3355   The SQL query schema does not map complex attributes as columns in the table for the class for
3356   which the attribute is defined. Instead the complex attributes are mapped to columns in the table
3357   for the domain class that represents the data type for the complex attribute (e.g.
3358   TelephoneNumber). A column links the row in the domain table to the row in the parent table
3359   (e.g. User). An additional column named 'attribute_name' identifies the attribute name in the
3360   parent class, in case there are multiple attributes with the same complex attribute type.

3361   This mapping also easily allows for attributes that are a collection of a complex type. For
3362   example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the
3363   TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier
3364   and an attribute_name.

8.3.1.5   Binding of Methods Returning Collections

3366   Several of the [ebRIM] classes define methods in addition to attributes, where these methods
3367   return collections of references to instances of classes defined by [ebRIM]. For example, the
3368   getPackages method of the ManagedObject class returns a Collection of references to instances
3369   of Packages that the object is a member of.

3370   Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in
3371   Appendix D.3 such that these stored procedures return a collection of `id` attribute values. The
3372   returned value of these stored procedures can be treated as the result of a table sub-query in SQL.

3373   These stored procedures may be used as the right-hand-side of an SQL IN clause to test for
3374   membership of an object in such collections of references.

8.3.2   Semantic Constraints On Query Syntax

3376   This section defines simplifying constraints on the query syntax that cannot be expressed in the
3377   BNF for the query syntax. These constraints must be applied in the semantic analysis of the
3378   query.

3379   1.   Class names and attribute names must be processed in a case insensitive manner.

3380   2.   The syntax used for stored procedure invocation must be consistent with the syntax of an
3381   SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].

3382   3.   For this version of the specification, the SQL select column list consists of exactly one
3383   column, and must always be `t.id`, where t is a table reference in the FROM clause.

3384   4.   Join operations must be restricted to simple joins involving only those columns that have an
3385   index defined within the normative SQL schema. This constraint is to prevent queries that
3386   may be computationally too expensive.

8.3.3   SQL Query Results

3388   The result of an SQL query resolves to a collection of objects within the registry. It never
3389   resolves to partial attributes. The objects related to the result set may be returned as an
3390   ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the
3391   responseOption parameter specified by the client on the AdHocQueryRequest. The entire result

3392   set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 8.1.

8.3.4 Simple Metadata Based Queries

3394   The simplest form of an SQL query is based upon metadata attributes specified for a single class
3395   within [ebRIM]. This section gives some examples of simple metadata based queries.

3396   For example, to get the collection of ExtrinsicObjects whose name contains the word 'Acme'
3397   and that have a version greater than 1.3, the following query must be submitted:

```
SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
      eo.id = nm.parent AND
      eo.majorVersion >= 1 AND
      (eo.majorVersion >= 2 OR  eo.minorVersion > 3);
```

3404   Note that the query syntax allows for conjugation of simpler predicates into more complex
3405   queries as shown in the simple example above.

8.3.5 RegistryObject Queries

3407   The schema for the SQL query defines a special view called RegistryObject that allows doing a
3408   polymorphic query against all RegistryObject instances regardless of their actual concrete type or
3409   table name.

3410   The following example is the similar to that in Section 8.3.4 except that it is applied against all
3411   RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id
3412   for all qualifying RegistryObject instances whose name contains the word 'Acme' and whose
3413   description contains the word "bicycle".

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

8.3.6 RegistryEntry Queries

3420   The schema for the SQL query defines a special view called RegistryEntry that allows doing a
3421   polymorphic query against all RegistryEntry instances regardless of their actual concrete type or
3422   table name.

3423   The following example is the same as Section 8.3.4 except that it is applied against all
3424   RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id
3425   for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a
3426   version greater than 1.3.

```
SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
      re.id = nm.parent AND
      re.majorVersion >= 1 AND
      (re.majorVersion >= 2 OR  re.minorVersion > 3);
```

8.3.7 Classification Queries

3434   This section describes the various classification related queries that must be supported.

8.3.7.1   Identifying ClassificationNodes

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

8.3.7.2   Getting ClassificationSchemes

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

```
SELECT scheme.id FROM ClassificationScheme scheme;
```

The above query returns all ClassificationSchemes. Note that the above query may also specify additional predicates (e.g. name, description etc.) if desired.

8.3.7.3   Getting Children of Specified ClassificationNode

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

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

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

8.3.7.4   Getting Objects Classified By a ClassificationNode

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

```
SELECT id FROM ExtrinsicObject
WHERE
    id IN (SELECT classifiedObject FROM Classification
           WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = '/Geography/Asia/Japan'))
  AND
    id IN (SELECT classifiedObject FROM Classification
           WHERE
               classificationNode IN (SELECT id FROM ClassificationNode
                                      WHERE path = '/Industry/Automotive'))
```

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

8.3.7.5   Getting Classifications That Classify an Object

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

```
SELECT id FROM Classification c
        WHERE c.classifiedObject = <id>;
```

3482 **8.3.8 Association Queries**

3483 This section describes the various Association related queries that must be supported.

3484 **8.3.8.1   Getting All Association With Specified Object As Its Source**

3485 To get the collection of Associations that have the specified Object as its source, the following
3486 query must be supported:

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

3490 **8.3.8.2   Getting All Association With Specified Object As Its Target**

3491 To get the collection of Associations that have the specified Object as its target, the following
3492 query must be supported:

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

3496 **8.3.8.3   Getting Associated Objects Based On Association Attributes**

3497 To get the collection of Associations that have specified Association attributes, the following
3498 queries must be supported:

3499 Select Associations that have the specified name.

```
SELECT id FROM Association WHERE name = <name>
```

3503 Select Associations that have the specified association type, where association type is a string
3504 containing the corresponding field name described in [ebRIM].

```
SELECT id FROM Association WHERE
       associationType = <associationType>
```

3509 **8.3.8.4   Complex Association Queries**

3510 The various forms of Association queries may be combined into complex predicates. The
3511 following query selects Associations that have a specific sourceObject, targetObject and
3512 associationType:

```
SELECT id FROM Association WHERE
       sourceObject = <id1> AND
       targetObject = <id2> AND
       associationType = <associationType>;
```

3519 **8.3.9 Package Queries**

3520 To find all Packages that a specified RegistryObject belongs to, the following query is specified:

```
SELECT id FROM Package WHERE id IN (RegistryObject_packages(<id>));
```

3524 **8.3.9.1   Complex Package Queries**

3525 The following query gets all Packages that a specified object belongs to, that are not deprecated
3526 and where name contains "RosettaNet."

```
3527
3528   SELECT id FROM Package p, Name n WHERE
3529        p.id IN (RegistryObject_packages(<id>)) AND
3530        nm.value LIKE '%RosettaNet%'  AND nm.parent = p.id AND
3531        p.status <> 'Deprecated'
3532
```

8.3.10 ExternalLink Queries

3534   To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is
3535   specified:
```
3536
3537   SELECT id From ExternalLink WHERE id IN (RegistryObject_externalLinks(<id>))
3538
```

3539   To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is
3540   specified:
```
3541
3542   SELECT id From ExtrinsicObject WHERE id IN (RegistryObject_linkedObjects(<id>))
3543
```

8.3.10.1   Complex ExternalLink Queries

3545   The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that
3546   contain the word 'legal' in their description and have a URL for their externalURI.
```
3547
3548   SELECT id FROM ExternalLink WHERE
3549        id IN (RegistryObject_externalLinks(<id>)) AND
3550        description LIKE '%legal%' AND
3551        externalURI LIKE '%http://%'
3552
```

8.3.11 Audit Trail Queries

3554   To get the complete collection of AuditableEvent objects for a specified ManagedObject, the
3555   following query is specified:
```
3556
3557   SELECT id FROM AuditableEvent WHERE  registryObject = <id>
3558
```

8.4 Content Retrieval

3560   A client retrieves content via the Registry by sending the GetContentRequest to the
3561   QueryManager. The GetContentRequest specifies a list of Object references for Objects that
3562   need to be retrieved. The QueryManager returns the specified content by sending a
3563   GetContentResponse message to the RegistryClient interface of the client. If there are no errors
3564   encountered, the GetContentResponse message includes the specified content as additional
3565   payloads within the message. In addition to the GetContentResponse payload, there is one
3566   additional payload for each content that was requested. If there are errors encountered, the
3567   RegistryResponse payload includes an error and there are no additional content specific
3568   payloads.

8.4.1 Identification Of Content Payloads

3570   Since the GetContentResponse message may include several repository items as additional
3571   payloads, it is necessary to have a way to identify each payload in the message. To facilitate this

3572   identification, the Registry must do the following:

3573   • Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mime-
3574   part that contains the corresponding repository item for the ExtrinsicObject

3575   • In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the
3576   repository item in the Reference element for that object in the Manifest element of the
3577   ebXMLHeader.

3578   ### 8.4.2 GetContentResponse Message Structure

3579   The following message fragment illustrates the structure of the GetContentResponse Message
3580   that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs
3581   for the requested objects.

```
Content-type: multipart/related; boundary="Boundary"; type="text/xml";

--BoundarY
Content-ID: <GetContentRequest@example.com>
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope  xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'
   xmlns:eb= 'http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
<SOAP-ENV:Header>

...ebMS header goes here if using ebMS

</SOAP-ENV:Header>
<SOAP-ENV:Body>

...ebMS manifest gooes here if using ebMS

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

<GetContentRequest>
   <ObjectRefList>
      <ObjectRef id="d8163dfb-f45a-4798-81d9-88aca29c24ff" …/>
      <ObjectRef id="212c3a78-1368-45d7-acc9-a935197e1e4f" …/>
   </ObjectRefList>
</GetContentRequest>

</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

--Boundary
Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
Content-Type: text/xml

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


--Boundary--
Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f
Content-Type: text/xml

<CPP>
.....
</CPP>

--Boundary—
```

9 Registry Security

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [ebRIM]. Security glossary terms can be referenced from RFC 2828.

9.1 Security Concerns

In the current version of this specification, we address data integrity and source integrity (item 1 in Appendix F.1). We have used a minimalist approach to address the access control concern as in item 2 of Appendix F.1. Essentially, "any known entity (Submitting Organization) can publish content and anyone can view published content." The Registry information model has been designed to allow more sophisticated security policies in future versions of this specification.

9.2 Integrity of Registry Content

It is assumed that most business registries do not have the resources to validate the veracity of the content submitted to them. "The mechanisms described in this section can be used to ensure that any tampering with the content submitted by a Submitting Organization can be detected. Furthermore, these mechanisms support unambiguous identification of the Responsible Organization for any registry content. The Registry Client has to sign the contents before submission – otherwise the content will be rejected. Note that in the discussions in this section we assume a Submitting Organization to be also the Responsible Organization. Future version of this specification may provide more examples and scenarios where a Submitting Organization and Responsible Organization are different.

9.2.1 Message Payload Signature

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

- Any tampering of the content can be detected.
- The content's veracity can be ascertained by its association with a specific Submitting Organization.

This section specifies the requirements for generation, packaging and validation of payload signatures. A payload signature is packaged with the payload. Therefore the requirements apply regardless of whether the Registry Client and the Registration Authority communicate over vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML Messaging Service does not specify the generation, validation and packaging of payload signatures. The specification of payload signatures is left upto the application (such as Registry). So the requirements on the payload signatures augment the [ebMS] specification.

Use Case

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

- RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the content along with the payload signature to the Registry.
- RC2 (Registry Client 2) retrieves RC1's content from the Registry.

3672 • RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
3673   the content, the response from the Registration Authority to RC2 contains the following:

3674   – Payload containing the content that has been published by RC1.

3675   – RC1's payload signature (represented by a ds:Signature element) over RC1's published
3676   content.

3677   – The public key for validating RC1's payload signature in ds:Signature element (using the
3678   KeyInfo element as specified in [XMLDSIG]) so RC2 can obtain the public key for
3679   signature (e.g. retrieve a certificate containing the public key for RC1).

3680   – A ds:Signature element containing the header signature. Note that the Registration
3681   Authority (not RC1) generates this signature.

3682   ### 9.2.2 Payload Signature Requirements

3683   #### 9.2.2.1   Payload Signature Packaging Requirements

3684   A payload signature is represented by a ds:Signature element. The payload signature must be
3685   packaged with the payload as specified here. This packaging assumes that the payload is always
3686   signed.

3687 • The payload and its signature must be enclosed in a MIME multipart message with a
3688   Content-Type of multipart/Related.

3689 • The first body part must contain the XML signature as specified in Section 9.2.2.2, "Payload
3690   Signature Generation Requirements".

3691 • The second through nth body part must be the content.

3692 The packaging of the payload signature with one payload is as follows:

3693
```
3694 MIME-Version: 1.0

3695 Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;

3696 Content-Description: ebXML Message

3697

3698 -- MIME_boundary

3699 Content-Type: text/xml; charset=UTF-8

3700 Content-Transfer-Encoding: 8bit

3701 Content-ID: http://claiming-it.com/claim061400a.xml

3702

3703 <?xml version='1.0' encoding="utf-8"?>

3704 <SOAP-ENV: Envelope>

3705   …

3706  SOAP-ENV: Envelope>

3707

3708 --MIME_boundary

3709 Content-Type: multipart/Related; boundary=PAYLOAD_boundary

3710

3711 --PAYLOAD_boundary

3712 Content-Type: text/xml; charset=UTF-8

3713 Content-Transfer-Encoding: 8bit

3714 Content-ID: payload1
```

```
3715   <ds:Signature>
3716      …. Payload signature
3717   </ds: Signature>
3718
3719   --PAYLOAD_boundary
3720   Content-Type: text/xml; charset=UTF-8
3721   Content-Transfer-Encoding: 8bit
3722   Content-ID: payload2
3723   <SubmitObjectsRequest>…</SubmitObjectsRequest>
3724   --MIME_boundary
3725
```

9.2.2.2   Payload Signature Generation Requirements

3727 The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in
3728 this section. Note: the "ds" name space reference is to http://www.w3.org/2000/09/xmldsig#

3729 • ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
3730   using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a
3731   client may use any of these attributes. However, signing using the following Algorithm
3732   attribute: http://www.w3.org/2000/09/xmldsig/#dsa-sha1 will allow interoperability with all
3733   XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this
3734   algorithm.

3735 The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The
3736 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported
3737   http://www.w3.org/TR/2001/REC-xml-c14n-2001315

3738 • One ds:Reference element to reference each of the payloads that needs to be signed must be
3739   created. The ds:Reference element:

3740 − Must identify the payload to be signed using the URI attribute of the ds:Reference
3741   element.

3742 − Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must be
3743   support the following digest algorithm:
3744   http://www.w3.org/2000/09/xmldsig/#sha1

3745 − Must contain a <ds:DigestValue> which is computed as specified in [XMLDSIG].

3746 The ds:SignedValue must be generated as specified in [XMLDSIG].

3747 The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject
3748 to the requirements stated in Section 9.4, "KeyDistrbution and KeyInfo element".

9.2.2.3   Message Payload Signature Validation

3750 The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].

9.2.2.4   Payload Signature Example

3752 The following example shows the format of the payload signature:

```
3754   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3755   <ds:SignedInfo>
3756       <SignatureMethod Algorithm="http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1"/>
```

```
3757        <ds:CanonicalizationMethod>
3758            Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
3759        </ds:CanonicalizationMethod>
3760         <ds:Reference URI=#Payload1>
3761            <ds:DigestMethod DigestAlgorithm="http://www.w3.org/TR/2000/09/xmldsig#sha1">
3762             <ds:DigestValue> ... </ds:DigestValue>
3763        </ds:Reference>
3764 </ds:SignedInfo>
3765 <ds:SignatureValue>  ...  </ds:SignatureValue>
3766 </ds:Signature>
3767
```

9.3 Authentication

3769 The Registry must be able to authenticate the identity of the Principal associated with client
3770 requests. The identity of the Principal can be identified by verifying the message header
3771 signature with the certificate of the Principal. The certificate may be in the message itself or
3772 provided to the registry through means unspecified in this specification. If not provided in the
3773 message, this specification does not specify how the Registry correlates a specific message with
3774 a certificate. Authentication of each payload must also be possible by using the signature
3775 associated with each payload. Authentication is also required to identify the "privileges" a
3776 Principal is authorized ("authorization") to have with respect to specific objects in the Registry.

3777 The Registry must perform authentication on a per message basis. From a security point of view,
3778 all messages are independent and there is no concept of a session encompassing multiple
3779 messages or conversations. Session support may be added as an optimization feature in future
3780 versions of this specification.

3781 It is important to note that the message header signature can only guarantee data integrity and it
3782 may be used for Authentication knowing that it is vulnerable to replay types of attacks. True
3783 support for authentication requires timestamps or nonce (nonrecurring series of numbers to
3784 identify each message) that are signed.

9.3.1 Message Header Signature

3786 Message headers are signed to provide data integrity while the message is in transit. Note that the
3787 signature within the message header also signs the digests of the payloads.

3788 Header Signature Requirements

3789 Message headers can be signed and are referred to as a header signature. This section specifies
3790 the requirements for generation, packaging and validation of a header signature. These
3791 requirements apply when the Registry Client and Registration Authority communicate using
3792 vanilla SOAP with Attachments. When ebXML MS is used for communication, then the
3793 message handler (i.e. [ebMS]) specifies the generation, packaging and validation of XML
3794 signatures in the SOAP header. Therefore the header signature requirements do not apply when
3795 the ebXML MS is used for communication. However, payload signature generation requirements
3796 (specified elsewhere in this document) do apply whether vanilla SOAP with Attachments or
3797 ebXML MS is used for communication.

3798 **9.3.1.1 Packaging Requirements**

3799 A header signature is represented by a ds:Signature element. The ds:Signature element generated
3800 must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature
3801 element in the SOAP header field is shown below.
3802

```
3803  MIME-Version: 1.0
3804  Content-Type: Multipart/Related; boundary=MIME_boundary; type=text/xml;
3805  Content-Description: ebXML Message
3806
3807  -- MIME_boundary
3808  Content-Type: text/xml; charset=UTF-8
3809  Content-Transfer-Encoding: 8bit
3810  Content-ID: http://claiming-it.com/claim061400a.xml
3811
3812  <?xml version='1.0' encoding="utf-8"?>
3813  <SOAP-ENV:Envelope
3814      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
3815      <SOAP-ENV:Header>
3816          <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3817              …signature over soap envelope
3818          </ds:Signature>
3819      </SOAP-ENV: Header>
3820      <SOAP-ENV: Body>
3821          …
3822      </SOAP-ENV: Body>
3823  </SOAP-ENV: Envelope>
3824
```

3825 **9.3.1.2 Header Signature Generation Requirements**

3826 The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in
3827 this section. A ds:Signature element contains:

3828 • ds:SignedInfo

3829 • ds:SignatureValue

3830 • ds:KeyInfo

3831 The ds:SignedInfo element must be generated as follows:

3832 1. ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified
3833   using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute,
3834   a client must be capable of signing using only the following Algorithm attribute:
3835   http://www.w3.org/2000/09/xmldsig/#dsa-sha1 This algorithm is being chosen because all
3836   XMLDSIG implementations conforming to the [XMLDSIG] specification support it.

3837 2. The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The
3838   following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:

3839   http://www.w3.org/TR/2001/REC-xml-c14n-20010315

3840   3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.
3841   This signs the entire ds:Reference element and:
3842   −   Must include the following ds:Transform:
3843   http://www.w3.org/2000/09/xmldsig#enveloped-signature
3844   This ensures that the signature (which is embedded in the <SOAP-ENV:Header>
3845   element) is not included in the signature calculation.
3846   −   Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the
3847   ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.) .
3848   The URI attribute must be "".
3849   −   Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support
3850   the following digest algorithm: http://www.w3.org/2000/09/xmldsig/#sha1
3851   −   Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].

3852   The ds:SignedValue must be generated as specified in [XMLDSIG].

3853   The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in
3854   Section 9.4, "KeyDistrbution and KeyInfo element".

9.3.1.3   Header Signature Validation Requirements

3856   The ds:Signature element for the ebXML message header must be validated by the recipient as
3857   specified by [XMLDSIG].

9.3.1.4   Header Signature Example

3859   The following example shows the format of a header signature:
3860

```
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>
        <SignatureMethod Algorithm=http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/>
        <ds:CanonicalizationMethod>
            Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
        </ds:CanonicalizationMethod>
        <ds:Reference URI= "">
            <ds:Transform>
                http://www.w3.org/2000/09/xmldsig#enveloped-signature
            </ds:Transform>
            <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
            <ds:DigestValue> ... </ds:DigestValue>
        </ds:Reference>
    </ds:SignedInfo>
    <ds:SignatureValue>  ...  </ds:SignatureValue>
</ds:Signature>
```
3877

9.4 Key Distribution and KeyInfo Element

3879   To validate a signature, the recipient of the signature needs the public key corresponding to the
3880   signer's public key. The participants may use the KeyInfo field of ds:Signature, or distribute the

3881 public keys out-of-band. In this section we consider the case when the public key is sent in the
3882 KeyInfo field. The following use cases need to be handled:

3883 • Registration Authority needs the public key of the Registry Client to validate the signature

3884 • Registry Client needs the public key of the Registration Authority to validate the Registry's
3885   signature.

3886 • Registry Client RC1 needs the public key of Registry Client (RC2) to validate the content
3887   signed by RC1.

3888 • [XMLDSIG] provides a ds:*KeyInfo* element that can be used to pass the recipient
3889   information for retrieving the public key. ds:*KeyInfo* is an optional element as specified in
3890   [XMLDSIG]. This field together with the procedures outlined in this section is used to
3891   securely pass the public key to a recipient. ds:Keyinfo can be used to pass information such
3892   as keys, certificates, names etc. The intended usage of KeyInfo field is to send the X509
3893   Certificate, and subsequently extract the public key from the certificate. Therefore, the
3894   KeyInfo field must contain a X509 Certificate as specified in [XMLDSIG], if the KeyInfo
3895   field is present.

3896 The following assumptions are also made:

3897 1. A Certificate is associated both with the Registration Authority and a Registry Client.

3898 2. A Registry Client registers its certificate with the Registration Authority. The mechanism
3899   used for this is not specified here.

3900 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local
3901   key store. The mechanism is not specified here.

3902 Couple of scenarios on the use of KeyInfo field is in Appendix F.8.

9.5 Confidentiality

9.5.1 On-the-wire Message Confidentiality

3905 It is suggested but not required that message payloads exchanged between clients and the
3906 Registry be encrypted during transmission. This specification does not specify how payload
3907 encryption is to be done.

9.5.2 Confidentiality of Registry Content

3909 In the current version of this specification, there are no provisions for confidentiality of Registry
3910 content. All content submitted to the Registry may be discovered and read by any client. This
3911 implies that the Registry and the client need to have an a priori agreement regarding encryption
3912 algorithm, key exchange agreements, etc. This service is not addressed in this specification.

9.6 Authorization

3914 The Registry must provide an authorization mechanism based on the information model defined
3915 in [ebRIM]. In this version of the specification the authorization mechanism is based on a default
3916 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of
3917 this specification will allow for custom Access Control Policies to be defined by the Submitting
3918 Organization. The authorization is going to be applied on a specific set of privileges. A

3919   privelege is the ability to carry a specific action.

9.6.1 Actions

3921   Life Cycle Actions

3922   submitObjects

3923   updateObjects

3924   addSlots

3925   removeSlots

3926   approveObjects

3927   deprecateObjects

3928   removeObjects

3929   Read Actions

3930   The various getXXX() methods in QueryManagement Service.

9.7 Access Control

3932   The Registry must create a default AccessControlPolicy object that grants the default
3933   permissions to Registry users based upon their assigned role. The following table defines the
3934   Permissions granted by the Registry to the various pre-defined roles for Registry users based
3935   upon the default AccessControlPolicy. Note that we have "ContentOwner" as a role. This role
3936   maps to the Submitting Organization in the current version of the specification.

3937   **Table 11: Default Access Control Policies**

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

3938   The Registry must implement the default AccessControlPolicy and associate it with all Objects
3939   in the Registry. The following list summarizes the default role-based AccessControlPolicy:

3940   • Anyone can publish content, but needs to be a Registered User

3941   • Anyone can access the content without requiring authentication

3942   • The ContentOwner has access to all methods for Registry Objects created by it.

3943   • The RegistryAdministrator has access to all methods on all Registry Objects

3944   • Unauthenticated clients can access all read-only (getXXX) methods

3945   • At the time of content submission, the Registry must assign the default ContentOwner role to
3946   the Submitting Organization (SO) as authenticated by the credentials in the submission
3947   message. In the current version of this specification, the Submitting Organization will be the
3948   DN as identified by the certificate
3949   • Clients that browse the Registry need not use certificates. The Registry must assign the
3950   default RegistryGuest role to such clients.

Appendix A   Web Service Architecture

A.1 Registry Service Abstract Specification

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

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

A.2 Registry Service SOAP Binding

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

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

Appendix B   ebXML Registry Schema Definitions

B.1 RIM Schema

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

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

B.2 Query Schema

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

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

B.3 Registry Services Interface Schema

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

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

B.4 Examples of Instance Documents

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

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

Appendix C   Interpretation of UML Diagrams

3980

3981 This section describes in *abstract terms* the conventions used to define ebXML business process
3982 description in UML.

C.1 UML Class Diagram

3983

3984 A UML class diagram is used to describe the Service Interfaces required to implement an
3985 ebXML Registry Services and clients. The UML class diagram contains:
3986

3987 1. A collection of UML interfaces where each interface represents a Service Interface for a
3988   Registry service.
3989 2. Tabular description of methods on each interface where each method represents an
3990   Action (as defined by [ebCPP]) within the Service Interface representing the UML
3991   interface.
3992 3. Each method within a UML interface specifies one or more parameters, where the type of
3993   each method argument represents the ebXML message type that is exchanged as part of
3994   the Action corresponding to the method. Multiple arguments imply multiple payload
3995   documents within the body of the corresponding ebXML message.

C.2 UML Sequence Diagram

3996

3997 A UML sequence diagram is used to specify the business protocol representing the interactions
3998 between the UML interfaces for a Registry specific ebXML business process. A UML sequence
3999 diagram provides the necessary information to determine the sequencing of messages, request to
4000 response association as well as request to error response association.

4001 Each sequence diagram shows the sequence for a specific conversation protocol as method calls
4002 from the requestor to the responder. Method invocation may be synchronous or asynchronous
4003 based on the UML notation used on the arrow-head for the link. A half arrow-head represents
4004 asynchronous communication. A full arrow-head represents synchronous communication.

4005 Each method invocation may be followed by a response method invocation from the responder to
4006 the requestor to indicate the ResponseName for the previous Request. Possible error response is
4007 indicated by a conditional response method invocation from the responder to the requestor. See
4008 Figure 7 on page 27 for an example.

4009 # Appendix D   SQL Query

4010 ## D.1 SQL Query Syntax Specification

4011 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The
4012 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax
4013 conforms to the <query specification>, modulo the restrictions identified below:

4014 1. A **<select list>** may contain at most one **<select sublist>.**

4015 2. In a **<select list>** must be is a single column whose data type is UUID, from the table in the
4016 **<from clause>.**

4017 3. A **<derived column>** may not have an **<as clause>**.

4018 4. **<table expression>** does not contain the optional **<group by clause>** and **<having clause>**
4019 clauses.

4020 5. A **<table reference>** can only consist of **<table name>** and **<correlation name>.**

4021 6. A **<table reference>** does not have the optional AS between **<table name>** and
4022 **<correlation name>.**

4023 7. There can only be one **<table reference>** in the **<from clause>**.

4024 8. Restricted use of sub-queries is allowed by the syntax as follows. The **<in predicate>** allows
4025 for the right hand side of the **<in predicate>** to be limited to a restricted **<query
4026 specification>** as defined above.

4027 9. A **<search condition>** within the **<where clause>** may not include a **<query expression>**.

4028 10. Simple joins are allowed only if they are based on indexed columns within the relational
4029 schema.

4030 11. The SQL query syntax allows for the use of **<sql invoked routines>** invocation from
4031 [SQL/PSM] as the RHS of the **<in predicate>**.

4032 ## D.2 Non-Normative BNF for Query Syntax Grammar

4033 The following BNF exemplifies the grammar for the registry query syntax. It is provided here as
4034 an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative
4035 syntax. For the normative syntax rules see Appendix D.1.
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054

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

RegistryQuery = SQLSelect [";"]

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

SQLSelectCols =  ID

SQLTableList = SQLTableRef

SQLTableRef = ID

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*
```

```
4055    SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*
4056
4057    SQLNotExpr = [ "NOT" ] SQLCompareExpr
4058
4059    SQLCompareExpr =
4060        (SQLColRef "IS") SQLIsClause
4061      | SQLSumExpr [ SQLCompareExprRight ]
4062
4063
4064    SQLCompareExprRight =
4065        SQLLikeClause
4066      | SQLInClause
4067      | SQLCompareOp SQLSumExpr
4068
4069    SQLCompareOp =
4070        "="
4071      | "<>"
4072      | ">"
4073      | ">="
4074      | "<"
4075      | "<="
4076
4077    SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
4078
4079    SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
4080
4081    SQLLValueElement = "NULL" | SQLSelect
4082
4083    SQLIsClause =  SQLColRef "IS" [ "NOT" ] "NULL"
4084
4085    SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
4086
4087    SQLPattern = STRING_LITERAL
4088
4089    SQLLiteral =
4090        STRING_LITERAL
4091      | INTEGER_LITERAL
4092      | FLOATING_POINT_LITERAL
4093
4094    SQLColRef = SQLLvalue
4095
4096    SQLLvalue =  SQLLvalueTerm
4097
4098    SQLLvalueTerm = ID ( "."  ID )*
4099
4100    SQLSumExpr = SQLProductExpr (( "+" | "-" ) SQLProductExpr )*
4101
4102    SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*
4103
4104    SQLUnaryExpr = [ ( "+" | "-") ] SQLTerm
4105
4106    SQLTerm = "(" SQLOrExpr ")"
4107      | SQLColRef
4108      | SQLLiteral
4109
4110    INTEGER_LITERAL = (["0"-"9"])+
4111
4112    FLOATING_POINT_LITERAL =
4113            (["0"-"9"])+ "." (["0"-"9"])+ (EXPONENT)?
4114          | "." (["0"-"9"])+ (EXPONENT)?
4115          | (["0"-"9"])+ EXPONENT
4116          | (["0"-"9"])+ (EXPONENT)?
4117
4118    EXPONENT = ["e","E"] (["+","-"])? (["0"-"9"])+
4119
4120    STRING_LITERAL: "'" (~["'"])* ( "''" (~["'"])* )* "'"
4121
4122    ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
4123    LETTER = ["A"-"Z", "a"-"z"]
4124    DIGIT = ["0"-"9"]
```

4125   D.3   Relational Schema For SQL Queries

4126   The normative Relational Schema definition for SQL queries can be found at the following
4127   location on the web:
4128   http://www.oasis-open.org/committees/regrep/documents/2.0/sql/database.sql
4129

4130   The stored procedures that must be supported by the SQL query feature are defined at the following
4131   location on the web:
4132   http://www.oasis-open.org/committees/regrep/documents/2.0/sql/storedProcedures.sql
4133

4134 Appendix E   Non-normative Content Based Ad Hoc Queries

4135 The Registry SQL query capability supports the ability to search for content based not only on
4136 metadata that catalogs the content but also the data contained within the content itself. For
4137 example it is possible for a client to submit a query that searches for all Collaboration Party
4138 Profiles that define a role named "seller" within a RoleName element in the CPP document itself.
4139 Currently content-based query capability is restricted to XML content.

4140 E.1 Automatic Classification of XML Content

4141 Content-based queries are indirectly supported through the existing classification mechanism
4142 supported by the Registry.

4143 A submitting organization may define logical indexes on any XML schema or DTD when it is
4144 submitted. An instance of such a logical index defines a link between a specific attribute or
4145 element node in an XML document tree and a ClassificationNode in a classification scheme
4146 within the registry.

4147 The registry utilizes this index to automatically classify documents that are instances of the
4148 schema at the time the document instance is submitted. Such documents are classified according
4149 to the data contained within the document itself.

4150 Such automatically classified content may subsequently be discovered by clients using the
4151 existing classification-based discovery mechanism of the Registry and the query facilities of the
4152 QueryManager.

4153 [Note] `This approach is conceptually similar to the way databases support`
4154 `indexed retrieval. DBAs define indexes on tables in the schema. When`
4155 `data is added to the table, the data gets automatically indexed.`

4156 E.2 Index Definition

4157 This section describes how the logical indexes are defined in the SubmittedObject element
4158 defined in the Registry Schema. The complete Registry Schema is available via hyperlinks in
4159 Appendix B.

4160 A SubmittedObject element for a schema or DTD may define a collection of
4161 ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList
4162 is ignored if the content being submitted is not of the SCHEMA objectType.

4163 The ClassificationIndex element inherits the attributes of the base class RegistryObject in
4164 [ebRIM]. It then defines specialized attributes as follows:

4165 1. classificationNode: This attribute references a specific ClassificationNode by its ID.

4166 2. contentIdentifier: This attribute identifies a specific data element within the document
4167 instances of the schema using an XPATH expression as defined by [XPT].

4168 E.3 Example Of Index Definition

4169 To define an index that automatically classifies a CPP based upon the roles defined within its
4170 RoleName elements, the following index must be defined on the CPP schema or DTD:
4171

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

E.4 Proposed XML Definition

```
<!--
A ClassificationIndexList is specified on ExtrinsicObjects of objectType
'Schema' to define an automatic Classification of instance objects of the
schema using the specified classificationNode as parent and a
ClassificationNode created or selected by the object content as selected
by the contentIdentifier
-->
<!ELEMENT ClassificationIndex EMPTY>
<!ATTLIST ClassificationIndex
        %ObjectAttributes;
        classificationNode IDREF #REQUIRED
        contentIdentifier CDATA #REQUIRED
>

<!-- ClassificationIndexList contains new ClassificationIndexes  -->
<!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

E.5 Example of Automatic Classification

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

Appendix F   Security Implementation Guideline

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

F.1 Security Concerns

The security risks broadly stem from the following concerns. After a description of these concerns and potential solutions, we identify the concerns that we address in the current specificiation

1. Is the content of the registry (data) trustworthy?

    a) How to make sure "what is in the registry" is "what is put there" by a submitting organization? This concern can be addressed by ensuring that the publisher is authenticated using digital signature (Source Integrity), message is not corrupted during transfer using digital signature (Data Integrity), and the data is not altered by unauthorized subjects based on access control policy (Authorization)

    b) How to protect data while in transmission?
        Communication integrity has two ingredients – Data Integrity (addressed in 1a) and Data Confidentiality that can be addressed by encrypting the data in transmission. How to protect against a replay attack?

    c) Is the content up to date? The versioning as well as any time stamp processing, when done securely will ensure the "latest content" is guaranteed to be the latest content.

    d) How to ensure only bona fide responsible organizations add contents to registry? Ensuring Source Integrity (as in 1a).

    e) How to ensure that bona fide publishers add contents to registry only at authorized locations? (System Integrity)

    f) What if the publishers deny modifying certain content after-the-fact? To prevent this (Nonrepudiation) audit trails may be kept which contain signed message digests.

    g) What if the reader denies getting information from the registry?

2. How to provide selective access to registry content? The broad answer is, by using an access control policy – applies to (a), (b), and (c) directly.

    a) How does a submitting organization restrict access to the content to only specific registry readers?

    b) How can a submitting organization allow some "partners" (fellow publishers) to modify content?

    c) How to provide selective access to partners the registry usage data?

    d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw failure of the registry security components? The solution to this problem is by having System Integrity.

    e) Data confidentiality of RegistryObject

4242   3. How do we make "who can see what" policy itself visible to limited parties, even excluding
4243   the administrator (self & confidential maintenance of access control policy). By making sure
4244   there is an access control policy for accessing the policies themselves.

4245   4. How to transfer credentials? The broad solution is to use credentials assertion (such as being
4246   worked on in SAML). Currently, Registry does not support the notion of a session.
4247   Therefore, some of these concerns are not releveant to the current specification.
4248   a) How to transfer credentials (authorization/authentication) to federated registries?
4249   b) How do aggregators get credentials (authorization/authentication) transferred to them?
4250   c) How to store credentials through a session?

F.2 Authentication

4252   1. As soon as a message is received, the first work is the authentication. A principal object is
4253   created.

4254   2. If the message is signed, it is verified (including the validity of the certificate) and the DN of
4255   the certificate becomes the identity of the principal. Then the Registry is searched for the
4256   principal and if found, the roles and groups are filled in.

4257   3. If the message is not signed, an empty principal is created with the role RegistryGuest. This
4258   step is for symmetry and to decouple the rest of the processing.

4259   4. Then the message is processed for the command and the objects it will act on.

F.3 Authorization

4261   For every object, the access controller will iterate through all the AccessControlPolicy objects
4262   with the object and see if there is a chain through the permission objects to verify that the
4263   requested method is permitted for the Principal. If any of the permission objects which the object
4264   is associated with has a common role, or identity, or group with the principal, the action is
4265   permitted.

F.4 Registry Bootstrap

4267   When a Registry is newly created, a default Principal object should be created with the identity
4268   of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message
4269   signed by the Registry Admin will get all the privileges.

4270   When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the
4271   default AccessControlPolicy. This includes the creation of the necessary Permission instances as
4272   well as the Privilges and Privilege attributes.

F.5 Content Submission – Client Responsibility

4274   The Registry client must sign the contents before submission – otherwise the content will be
4275   rejected.

F.6 Content Submission – Registry Responsibility

4276

4277 1. As with any other request, the client will first be authenticated. In this case, the Principal
4278   object will get the DN from the certificate.

4279 2. As per the request in the message, the RegistryEntry will be created.

4280 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.

4281 4. If a principal with the identity of the SO is not available, an identity object with the SO's DN
4282   is created.

4283 5. A principal with this identity is created.

F.7 Content Delete/Deprecate – Client Responsibility

4284

4285 The Registry client must sign the header before submission, for authentication purposes;
4286 otherwise, the request will be rejected

F.8 Content Delete/Deprecate – Registry Responsibility

4287

4288 1. As with any other request, the client will first be authenticated. In this case, the Principal
4289   object will get the DN from the certificate. As there will be a principal with this identity in
4290   the Registry, the Principal object will get all the roles from that object

4291 2. As per the request in the message (delete or deprecate), the appropriate method in the
4292   RegistryObject class will be accessed.

4293 3. The access controller performs the authorization by iterating through the Permission objects
4294   associated with this object via the singleton default AccessControlPolicy.

4295 4. If authorization succeeds then the action will be permitted. Otherwise an error response is
4296   sent back with a suitable AuthorizationException error message.

F.9 Using ds:KeyInfo Field

4297

4298 Two typical usage scenarios for ds:KeyInfo are described below.

4299 **Scenario 1**

4300 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.

4301 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.

4302 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.

4303 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature

4304 5. Registration Authority validates the header signature using the public key from the
4305   certificate.

4306 6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the
4307   certificate from the KeyInfo field of the payload signature. Note that this step is not an
4308   essential one if the onus of validation is that of the eventual user, another Registry Client, of
4309   the content.

4310 **Scenario 2**

4311   1. RC1 signs the payload and SOAP envelope using its private key and publishes to the
4312   Registry.

4313   2. The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.

4314   3. The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature.
4315   This step is required in addition to step 2 because when RC2 retrieves content, it should see
4316   RC1's signature with the payload.

4317   4. RC2 retrieves content from the Registry.

4318   5. Registration Authority signs the SOAP envelope using its private key. Registration Authority
4319   sends RC1's content and the RC1's signature (signed by RC1).

4320   6. Registration Authority need not send its certificate in the KeyInfo field sinceRC2 is assumed
4321   to have obtained the Registration Authority's certificate out of band and installed it in its
4322   local key store.

4323   7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the
4324   Registration Authority's signature.

4325   8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates
4326   the signature on the payload.

Appendix G   Native Language Support (NLS)

G.1 Definitions

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

G.1.1 Coded Character Set (CCS):

CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.

G.1.2 Character Encoding Scheme (CES):

CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are ISO-2022, UTF-8.

G.1.3 Character Set (charset):

- charset is a set of rules for mapping from a sequence of octets to a sequence of characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR.
- A list of registered character sets can be found at [IANA].

G.2 NLS And Request / Response Messages

For the accurate processing of data in both registry client and registry services, it is essential to know which character set is used. Although the body part of the transaction may contain the charset in xml encoding declaration, registry client and registry services shall specify charset parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a text/xml entity is received with the charset parameter omitted, MIME processors and XML processors MUST use the default charset value of "us-ascii". For example:

```
Content-Type: text/xml; charset=ISO-2022-JP
```

Also, when an application/xml entity is used, the charset parameter is optional, and registry client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which directly address this contingency.

If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

G.3 NLS And Storing of RegistryObject

This section provides NLS guidelines on how a registry should store RegistryObject instances.

A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple locales. Thus there is no language or character set associated with a specific RegistryObject instance.

A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows. Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in

4362   RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed
4363   as LocalizedString sub-elements within the XML element representing the I18N capable
4364   attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a
4365   specific locale. Each LocalizedString element has a charset and lang attribute as well as a value
4366   attribute of type string.

G.3.1 Character Set of *LocalizedString*

4367

4368   The character set used by a locale specific String (LocalizedString) is defined by the charset
4369   attribute. It is highly recommended to use UTF-8 or UTF-16 for maximuminter-operability.

G.3.2 Language Information of *LocalizedString*

4370

4371   The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

G.4 NLS And Storing of Repository Items

4372

4373   This section provides NLS guidelines on how a registry should store repository items.
4374   While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is
4375   always associated with a single repository item. The repository item may be in a single locale or
4376   may be in multiple locales. This specification does not specify the repository item.

G.4.1 Character Set of Repository Items

4377

4378   The MIME `Content-Type` mime header for the mime multi-part containing the repository
4379   item MAY contain a `"charset"` attribute that specifies the character set used by the repository
4380   item. For example:
4381

4382
```
Content-Type: text/xml; charset="UTF-8"
```
4383

4384   It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset
4385   of a repository item must be preserved as it is originally specified in the transaction.

G.4.2 Language information of repository item

4386

4387   The Content-language mime header for the mime bodypart containing the repository item may
4388   specify the language for a locale specific repository item. The value of the Content-language
4389   mime header property must conform to [RFC 1766].

4390   This document currently specifies only the method of sending the information of character set
4391   and language, and how it is stored in a registry. However, the language information may be used
4392   as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a
4393   language negotiation procedure, like registry client is asking a favorite language for messages
4394   from registry services, could be another functionality for the future revision of this document.

4395 Appendix H   Registry Profile

4396 Every registry must support exactly one Registry Profile. The Registry Profile is an XML
4397 document that describes the capabilities of the registry. The profile document must conform to
4398 the RegistryProfile element as described in the Registry Services Interface schema defined in
4399 Appendix B. The registry must make the RegistryProfile accessible over HTTP protocol via a
4400 URL. The URL must conform to the pattern:

4401 http://<base url>/registryProfile

4402

10 References

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

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

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

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

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

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

[XPT] XML Path Language (XPath) Version 1.0
    http://www.w3.org/TR/xpath

[SQL] Structured Query Language (FIPS PUB 127-2)
    http://www.itl.nist.gov/fipspubs/fip127-2.htm

[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
    (SQL/PSM) [ISO/IEC 9075-4:1996]

[IANA] IANA (Internet Assigned Numbers Authority).
    Official Names for Character Sets, ed. Keld Simonsen et al.
     ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

[RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
    Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
    http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html

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

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

[RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
    IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
    http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html

[RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
    IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
    http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html

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

[RFC 3023] ETF (Internet Engineering Task Force). RFC 3023:
    XML Media Types, ed. M. Murata. 2001.
    ftp://ftp.isi.edu/in-notes/rfc3023.txt

[REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
    http://www.w3.org/TR/REC-xml

[UUID] DCE 128 bit Universal Unique Identifier
    http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
    http://www.opengroup.org/publications/catalog/c706.htmhttp://www.w3.org/TR/REC-xml

[WSDL] W3C Note. Web Services Description Language (WSDL) 1.1

4447   http://www.w3.org/TR/wsdl

4448   [SOAP11] W3C Note. Simple Object Access Protocol, May 2000,
4449   http://www.w3.org/TR/SOAP

4450   [SOAPAt] W3C Note: SOAP with Attachments, Dec 2000,
4451   http://www.w3.org/TR/SOAP-attachments

4452   [XMLDSIG] XML-Signature Syntax and Processing,
4453   http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/

4454 **11 Disclaimer**

4455   The views and specification expressed in this document are those of the authors and are not
4456   necessarily those of their employers. The authors and their employers specifically disclaim
4457   responsibility for any problems arising from correct or incorrect implementation or use of this
4458   design.

4459 **12 Contact Information**

4460 Team Leader
4461   Name: Lisa J. Carnahan
4462   Company: National Institute of Standards and Technology
4463   Street: 100 Bureau Drive, Stop 8970
4464   City, State, Postal Code: Gaithersburg, Md. 20899
4465   Country: USA
4466   Phone: 301-975-3362
4467   Email: lisa.carnahan@nist.gov
4468
4469 Editor
4470   Name: Anne A. Fischer
4471   Company: Drummond Group, Inc.
4472   Street: 4700 Bryant Irvin Ct., Suite 303
4473   City, State, Postal Code: Fort Worth, Texas 76107-7645
4474   Country: USA
4475   Phone: 817-371-2367
4476   Email: anne@drummondgroup.com
4477
4478 Technical Editor
4479   Name: Farrukh S. Najmi
4480   Company: Sun Microsystems
4481   Street: 1 Network Dr., MS BUR02-302
4482   City, State, Postal Code: Burlington, MA 01803-0902
4483   Country: USA
4484   Phone: 781-442-0703
4485   Email: najmi@east.sun.com
4486

4487 # 13 Copyright Statement

4488 Portions of this document are copyright (c) 2001 OASIS and UN/CEFACT.

4507 OASIS takes no position regarding the validity or scope of any intellectual property or other
4508 rights that might be claimed to pertain to the implementation or use of the technology described
4509 in this document or the extent to which any license under such rights might or might not be
4510 available; neither does it represent that it has made any effort to identify any such rights.
4511 Information on OASIS's procedures with respect to rights in OASIS specifications can be found
4512 at the OASIS website. Copies of claims of rights made available for publication and any
4513 assurances of licenses to be made available, or the result of an attempt made to obtain a general
4514 license or permission for the use of such proprietary rights by implementors or users of this
4515 specification, can be obtained from the OASIS Executive Director.

4516 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
4517 applications, or other proprietary rights which may cover technology that may be required to
4518 implement this specification. Please address the information to the OASIS Executive Director.