Creating A Single Global Electronic Market

1

ebXML Registry Services Specification v1.0

ebXML Registry Project Team

10 May 2001

5

1 Status of this Document

7

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

9

Distribution of this document is unlimited.

11

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

13

This version:

http://www.ebxml.org/specs/ebRS.pdf

16

***Latest version*:**

http://www.ebxml.org/specs/ebRS.pdf

19

2 ebXML participants

ebXML Registry Services, v1.0 was developed by the ebXML Registry Project Team.
At the time this specification was approved, the membership of the ebXML Registry
Project Team was as follows:

24

Lisa Carnahan, NIST
Joe Dalman, Tie
Philippe DeSmedt, Viquity

28   Sally Fuger, AIAG
29   Len Gallagher, NIST
30   Steve Hanna, Sun Microsystems
31   Scott Hinkelman, IBM
32   Michael Kass, NIST
33   Jong.L Kim, Innodigital
34   Kyu-Chul Lee, Chungnam National University
35   Sangwon Lim, Korea Institute for Electronic Commerce
36   Bob Miller, GXS
37   Kunio Mizoguchi, Electronic Commerce Promotion Council of Japan
38   Dale Moberg, Sterling Commerce
39   Ron Monzillo, Sun Microsystems
40   JP Morgenthal, eThink Systems, Inc.
41   Joel Munter, Intel
42   Farrukh Najmi, Sun Microsystems
43   Scott Nieman, Norstan Consulting
44   Frank Olken, Lawrence Berkeley National Laboratory
45   Michael Park, eSum Technologies
46   Bruce Peat, eProcess Solutions
47   Mike Rowley, Excelon Corporation
48   Waqar Sadiq, Vitria
49   Krishna Sankar, Cisco Systems Inc.
50   Kim Tae Soo, Government of Korea
51   Nikola Stojanovic, Encoda Systems, Inc.
52   David Webber, XML Global
53   Yutaka Yoshida, Sun Microsystems
54   Prasad Yendluri, webmethods
55   Peter Z. Zhoo, Knowledge For the new Millennium

56

Table of Contents

210 ## Table of Figures

229   Table of Tables

232 3 Introduction

233 3.1 Summary of Contents of Document

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

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

239 3.2 General Conventions

240 The following conventions are used throughout this document:

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

243 o The term *"repository item"* is used to refer to an object that has been submitted to a
244   Registry for storage and safekeeping (e.g. an XML document or a DTD). Every
245   repository item is described by a RegistryEntry instance.

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

248 o *Capitalized Italic* words are defined in the ebXML Glossary.

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

252 3.3 Audience

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

255 o Implementers of ebXML Registry Services

256 o Implementers of ebXML Registry Clients

257 3.4 Related Documents

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

260   a) *ebXML Registry Information Model [ebRIM]*

261   b) *ebXML Message Service Specification* [ebMS]

262   c) *ebXML Business Process Specification Schema* [ebBPM]

263   d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

264 4   Design Objectives

265 4.1 Goals

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

267   o   Communicate functionality of Registry services to software developers

268   o   Specify the interface for Registry clients and the Registry

269   o   Provide a basis for future support of more complete ebXML Registry requirements

270   o   Be compatible with other ebXML specifications

271 4.2 Caveats and Assumptions

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

275   It is assumed that:

276   1. Interoperability requirements dictate that the ebXML Message Services
277   Specification is used between an ebXML Registry and an ebXML Registry
278   Client. The use of other communication means is not precluded; however, in
279   those cases interoperability cannot be assumed. Other communication means
280   are outside the scope of this specification.

281   2. All access to the Registry content is exposed via the interfaces defined for the
282   Registry Services.

283   3. The Registry makes use of a Repository for storing and retrieving persistent
284   information required by the Registry Services. This is an implementation detail
285   that will not be discussed further in this specification.

286 5   System Overview

287 5.1 What The ebXML Registry Does

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

293 5.2 How The ebXML Registry Works

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

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

5.2.1   Schema Documents Are Submitted

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

5.2.2   Business Process Documents Are Submitted

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

5.2.3   Seller's Collaboration Protocol Profile Is Submitted

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

5.2.4   Buyer Discovers The Seller

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

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

5.2.5   CPA Is Established

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

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

5.3 Where the Registry Services May Be Implemented

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

5.4 Implementation Conformance

337 An implementation is a *conforming* ebXML Registry if the implementation meets the
338 conditions in Section 5.4.1. An implementation is a conforming ebXML Registry Client if
339 the implementation meets the conditions in Section 5.4.2. An implementation is a
340 conforming ebXML Registry and a conforming ebXML Registry Client if the
341 implementation conforms to the conditions of Section 5.4.1 and Section 5.4.2. An
342 implementation shall be a conforming ebXML Registry, a conforming ebXML Registry
343 Client, or a conforming ebXML Registry and Registry Client.

5.4.1 Conformance as an ebXML Registry

345 An implementation conforms to this specification as an ebXML registry if it meets the
346 following conditions:

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

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

3. Supports the defined ebXML Registry DTD (Appendix A)

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

5.4.2 Conformance as an ebXML Registry Client

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

1. Supports the ebXML CPA and bootstrapping process.

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

3. Supports the defined ebXML Error Message DTD.

4. Supports the defined ebXML Registry DTD.

360 6 Registry Architecture

361 The ebXML Registry architecture consists of an ebXML Registry and ebXML Registry
362 Clients. The Registry Client interfaces may be local to the registry or local to the user.
363 Figure 1 depicts the two possible topologies supported by the registry architecture with
364 respect to the Registry and Registry Clients.

365 The picture on the left side shows the scenario where the Registry provides a web
366 based "thin client" application for accessing the Registry that is available to the user
367 using a common web browser. In this scenario the Registry Client interfaces reside
368 across the internet and are local to the Registry from the user's view.

369 The picture on the right side shows the scenario where the user is using a "fat client"
370 Registry Browser application to access the registry. In this scenario the Registry Client
371 interfaces reside within the Registry Browser tool and are local to the Registry from the
372 user's view. The Registry Client interfaces communicate with the Registry over the
373 internet in this scenario.

374 A third topology made possible by the registry architecture is where the Registry Client
375 interfaces reside in a server side business component such as a Purchasing business
376 component. In this topology there may be no direct user interface or user intervention
377 involved. Instead the Purchasing business component may access the Registry in an
378 automated manner to select possible sellers or service providers based current
379 business needs.
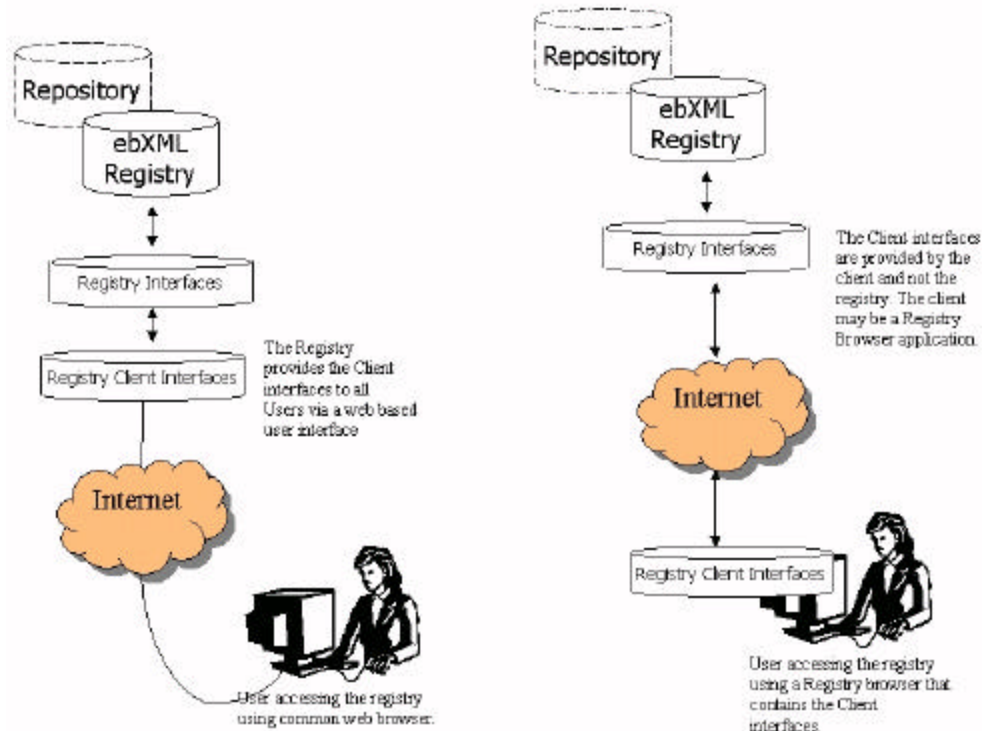
380

381   **Figure 1: Registry Architecture Supports Flexible Topologies**

382 Clients communicate with the Registry using the ebXML Messaging Service in the same
383 manner as any two ebXML applications communicating with each other.

384 Future versions of this specification may provide additional services to explicitly extend
385 the Registry architecture to support distributed registries. However this current version
386 of the specification does not preclude ebXML Registries from cooperating with each
387 other to share information, nor does it preclude owners of ebXML Registries from
388 registering their ebXML registries with other registry systems, catalogs, or directories.

389 Examples include:

390 • an ebXML Registry of Registries that serves as a centralized registration point;

391 • cooperative ebXML Registries, where registries register with each other in a
392 federation;

393 • registration of ebXML Registries with other Registry systems that act as white
394 pages or yellow pages. The document [ebXML-UDDI] provides an example of
395 ebXML Registries being discovered through a system of emerging white/yellow
396 pages known as UDDI.

397 6.1 ebXML Registry Profiles and Agreements

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

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

407 The CPA between clients and the Registry should describe the interfaces that the
408 Registry and the client expose to each other for Registry-specific interactions. These
409 interfaces are described in Figure 2 and subsequent sections. The definition of the
410 Registry CPP template and a Registry Client CPP template are beyond the scope of this
411 document.

412 6.2 Client To Registry Communication Bootstrapping

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

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

423   For the duration of the client's session with the Registry, messages may be exchanged
424   bidirectionally as required by the interaction protocols defined in this specification.
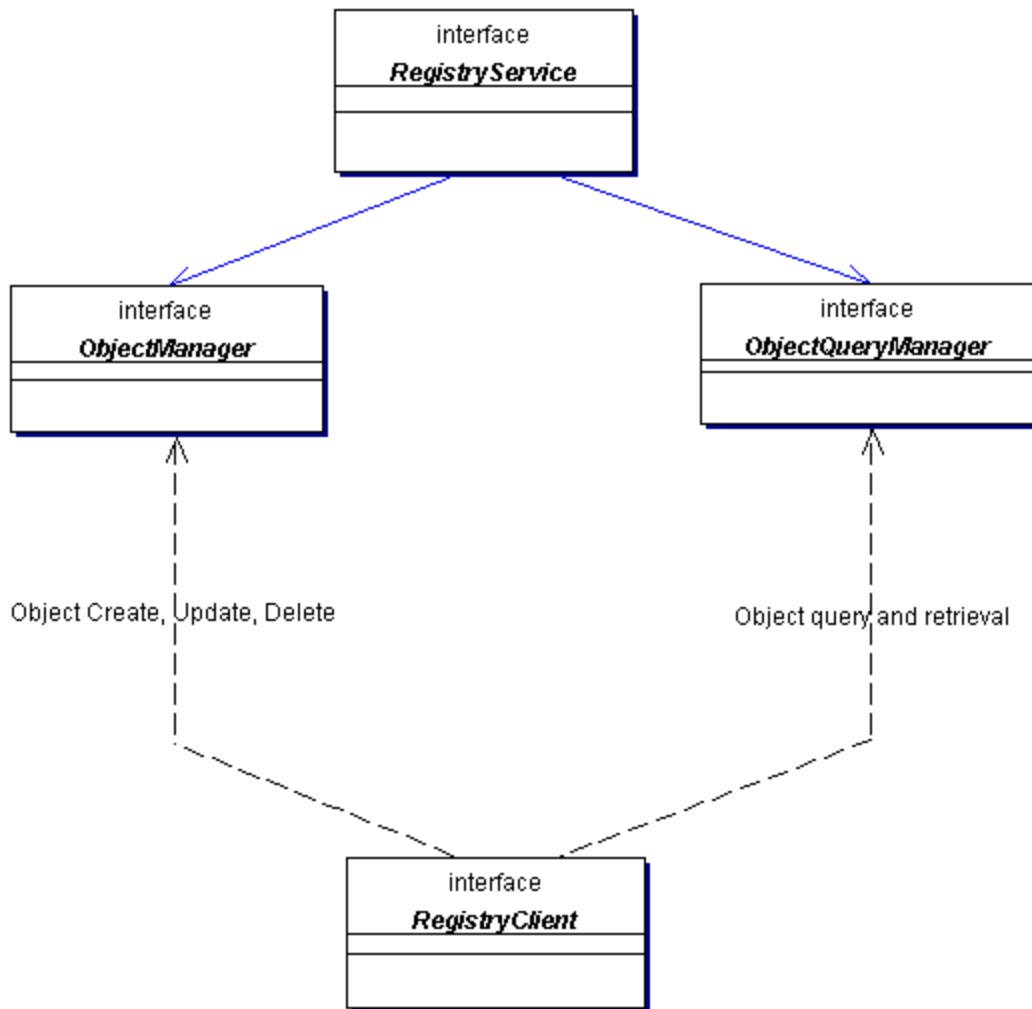
425
426   **Figure 2: ebXML Registry Interfaces**

427   6.3 Interfaces

428   This specification defines the interfaces exposed by both the Registy (Section 6.4) and
429   the Registry Client (Section 6.5). Figure 2 shows the relationship between the
430   interfaces and the mapping of specific Registy interfaces with specific Registry Client
431   interfaces.

6.4 Interfaces Exposed By The Registry

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

- The value of the Service element in the MessageHeader is an ebXML Registry
  Service interface name (e.g., "ObjectManager"). The type attribute of the Service
  element should have a value of "ebXMLRegistry".

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

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

6.4.1 Synchronous and Asynchronous Responses

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

- Asynchronous response
  - MessageHeader only;
  - No registry response element (e.g., AdHocQueryResponse and
    GetContentResponse).
- Synchronous response
  - MessageHeader;
  - Registry response element including
    - a status attribute (success or failure)
    - an optional ebXML Error.

The ebXML Registry implements the following interfaces as its services (Registry
Services).

6.4.2 Interface RegistryService

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

Method Summary of RegistryService	
ObjectManager	getObjectManager()

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

462   ### 6.4.3   Interface ObjectManager

463

464   This is the interface e xposed by the Registry Service that implements the Object life
465   cycle management functionality of the Registry. Its methods are invoked by the Registry
466   Client. For example, the client may use this interface to submit objects, to classify and
467   associate objects and to deprecate and remove objects. For this specification the
468   semantic meaning of submit, classify, associate, deprecate and remove is found in
469   [ebRIM].

470

Method Summary of ObjectManager

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

471   ### 6.4.4   Interface ObjectQueryManager

472

473   This is the interface exposed by the Registry that implements the Object Query
474   management service of the Registry. Its methods are invoked by the Registry Client.

ebXML Registry Services Specification   Page 16

475 For example, the client may use this interface to perform browse and drill down queries
476 or ad hoc queries on registry content.

477

Method Summary of ObjectQueryManager	
RegistryResponse	**getClassificationTree**(GetClassificationTreeRequest req) Returns the ClassificationNode Tree under the ClassificationNode specified in GetClassificationTreeRequest.
RegistryResponse	**getClassifiedObjects**(GetClassifiedObjectsRequest req) Returns a collection of references to RegistryEntries classified under specified ClassificationItem.
RegistryResponse	**getContent**() Returns the content of the specified Repository Item. The response includes all the content specified in the request as additional payloads within the response message.
RegistryResponse	**getRootClassificationNodes**(GetRootClassificationNodesRequest req) Returns all root ClassificationNodes that match the namePattern attribute in GetRootClassificationNodesRequest request.
RegistryResponse	**submitAdhocQuery**(AdhocQueryRequest req) Submit an ad hoc query request.

6.5 Interfaces Exposed By Registry Clients

479 An ebXML Registry client implements the following interface.

6.5.1 Interface RegistryClient

481

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

487 The registry sends all asynchronous responses to operations to the onResponse
488 method.

489 _____

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

490

6.6 Registry Response Class Hierarchy

492 Since many of the responses from the registry have common attributes they are
493 arranged in the following class hierarchy. This hierarchy is reflected in the registry DTD.
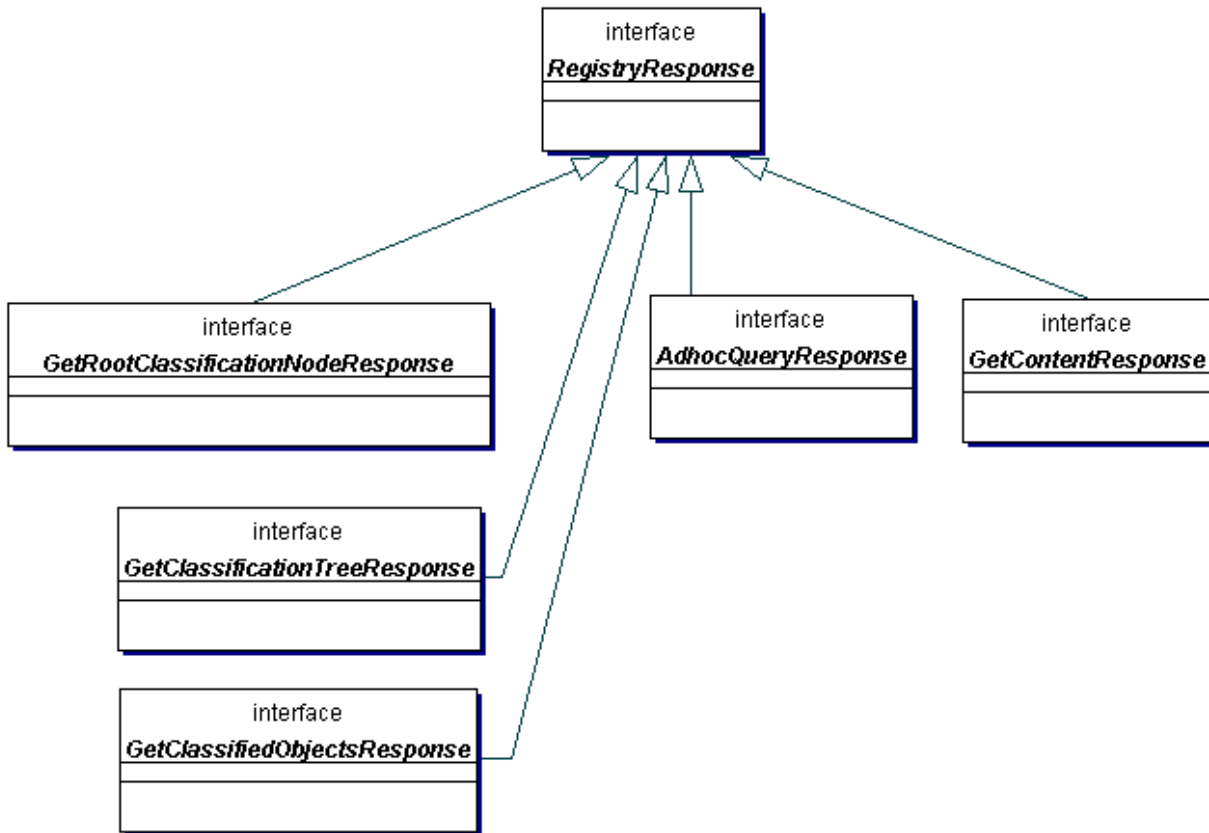
494

495   **Figure 3: Registry Reponse Class Hierarchy**

496
497

497
498
499

7   Object Management Service

501

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

The minimum *security policy* for an ebXML registry is to accept content from any client if
the content is digitally signed by a certificate issued by a Certificate Authority
recognized by the ebXML registry. Submitting Organizations do not have to register
prior to submitting content.

7.1   Life Cycle of a Repository Item

The main purpose of the ObjectManagement service is to manage the life cycle of
repository items.

Figure 4 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.
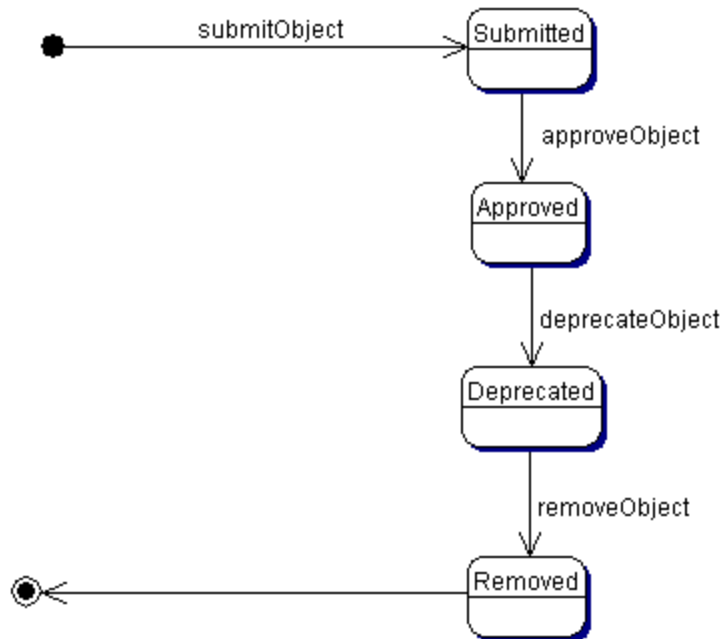
518

519   **Figure 4: Life Cycle of a Repository Item**

7.2 RegistryObject Attributes

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

7.3 The Submit Objects Protocol

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

532

533
534   **Figure 5: Submit Objects Sequence Diagram**

535 For details on the schema for the *Business documents* shown in this process refer to
536 Appendix A.

537 The SubmitObjectRequest message includes a RegistrEntryList element.

538 The RegistryEntryList element specifies one or more ExtrinsicObjects or other
539 RegistryEntries such as Classifications, Associations, ExternalLinks, or Packages.

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

544   In the event of success, the registry sends a RegistryResponse with a status of
545   "success" back to the client. In the event of failure, the registry sends a
546   RegistryResponse with a status of "failure" back to the client.

7.3.1   Universally Unique ID Generation

547

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

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

552   This id is usually generated by the registry. The `id` attribute for submitted objects may
553   optionally be supplied by the client. If the client supplies the `id` and it conforms to the
554   format of a URN that specifies a DCE 128 bit UUID then the registry assumes that the
555   client wishes to specify the `id` for the object. In this case, the registry must honor a
556   client-supplied `id` and use it as the `id` attribute of the object in the registry. If the `id` is
557   found by the registry to not be globally unique, the registry must raise the error
558   condition: InvalidIdError.

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

7.3.2   ID Attribute And Object References

563

564   The id attribute of an object may be used by other objects to reference the first object.
565   Such references are common both within the SubmitObjectsRequest as well as within
566   the registry. Within a SubmitObjectsRequest, the id attribute may be used to refer to an
567   object within the SubmitObjectsRequest as well as to refer to an object within the
568   registry. An object in the SubmitObjectsRequest that needs to be referred to within the
569   request document may be assigned an id by the submitter so that it can be referenced
570   within the request. The submitter may give the object a proper uuid URN, in which case
571   the id is permanently assigned to the object within the registry. Alternatively, the
572   submitter may assign an arbitrary id (not a proper uuid URN) as long as the id is unique
573   within the request document. In this case the id serves as a linkage mechanism within
574   the request document but must be ignored by the registry and replaced with a registry
575   generated id upon submission.

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

580 ### 7.3.3 Sample SubmitObjectsRequest

581 The following example shows several different use cases in a single
582 SubmitObjectsRequest. It does not show the complete ebXML Message with the
583 message header and additional payloads in the message for the repository items.

584 A SubmitObjectsRequest includes a RegistryEntryList which contains any number of
585 objects that are being submitted. It may also contain any number of ObjectRefs to link
586 objects being submitted to objects already within the registry.

587
```
588 <?xml version = "1.0" encoding = "UTF-8"?>
589 <!DOCTYPE SubmitObjectsRequest SYSTEM "file:////home/najmi/Registry.dtd">
590
591 <SubmitObjectsRequest>
592   <RegistryEntryList>
593
594     <!—
595     The following 3 objects package specified ExtrinsicObject in specified
596     Package, where both the Package and the ExtrinsicObject are
597     being  submitted
598     -->
599     <Package id = "acmePackage1" name = "Package #1" description = "ACME's package #1"/>
600     <ExtrinsicObject id = "acmeCPP1" contentURI = "CPP1"
601       objectType = "CPP" name = "Widget Profile"
602       description = "ACME's profile for selling widgets"/>
603     <Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages"
604       sourceObject = "acmePackage1" targetObject = "acmeCPP1"/>
605
606     <!—
607     The following 3 objects package specified ExtrinsicObject in specified Package,
608     Where the Package is being submitted and the ExtrinsicObject is
609     already in registry
610     -->
611     <Package id = "acmePackage2" name = "Package #2" description = "ACME's package #2"/>
612     <ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
613     <Association id = "acmePackage2-alreadySubmittedCPP-Assoc"
614       associationType = "Packages" sourceObject = "acmePackage2"
615       targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
616
617     <!—
618     The following 3 objects package specified ExtrinsicObject in specified Package,
619     where the Package and the ExtrinsicObject are already in registry
620     -->
621     <ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
622     <ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
623     <!-- id is unspecified implying that registry must create a uuid for this object -->
624     <Association associationType = "Packages"
625       sourceObject = "urn:uuid:b2345678-1234-1234-123456789012"
626       targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
627
628     <!—
629     The following 3 objects externally link specified ExtrinsicObject using
630     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
631     are being submitted
632     -->
633     <ExternalLink id = "acmeLink1" name = "Link #1" description = "ACME's Link #1"/>
634     <ExtrinsicObject id = "acmeCPP2" contentURI = "CPP2" objectType = "CPP"
```

```
635          name = "Sprockets Profile" description = "ACME's profile for selling sprockets"/>
636        <Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"
637          sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
638
639        <!--
640        The following 2 objects externally link specified ExtrinsicObject using specified
641        ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
642        is already in registry. Note that the targetObject points to an ObjectRef in a
643        previous line
644        -->
645        <ExternalLink id = "acmeLink2" name = "Link #2" description = "ACME's Link #2"/>
646        <Association id = "acmeLink2-alreadySubmittedCPP-Assoc"
647          associationType = "ExternallyLinks" sourceObject = "acmeLink2"
648          targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
649
650        <!--
651        The following 2 objects externally identify specified ExtrinsicObject using specified
652        ExternalIdentifier, where the ExternalIdentifier is being submitted and the
653        ExtrinsicObject is already in registry. Note that the targetObject points to an
654        ObjectRef in a previous line
655        -->
656        <ExternalIdentifier id = "acmeDUNSId" name = "DUNS" description = "DUNS ID for ACME"
657          value = "13456789012"/>
658        <Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc"
659          associationType = "ExternallyIdentifies" sourceObject = "acmeDUNSId"
660          targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
661
662        <!--
663        The following show submission of a brand new classification scheme in its entirety
664        -->
665        <ClassificationNode id = "geographyNode" name = "Geography"
666          description = "The Geography scheme example from Registry Services Spec" />
667        <ClassificationNode id = "asiaNode" name = "Asia"
668          description = "The Asia node under the Geography node" parent="geographyNode" />
669        <ClassificationNode id = "japanNode" name = "Japan"
670          description ="The Japan node under the Asia node" parent="asiaNode" />
671        <ClassificationNode id = "koreaNode" name = "Korea"
672          description ="The Korea node under the Asia node" parent="asiaNode" />
673        <ClassificationNode id = "europeNode" name = "Europe"
674          description = "The Europe node under the Geography node" parent="geographyNode" />
675        <ClassificationNode id = "germanyNode" name = "Germany"
676          description ="The Germany node under the Asia node" parent="europeNode" />
677        <ClassificationNode id = "northAmericaNode" name = "North America"
678          description = "The North America node under the Geography node"
679          parent="geographyNode" />
680        <ClassificationNode id = "usNode" name = "US"
681          description ="The US node under the Asia node" parent="northAmericaNode" />
682
683        <!--
684        The following show submission of a Automotive sub-tree of ClassificationNodes that
685        gets added to an existing classification scheme named 'Industry'
686        that is already in the registry
687        -->
688        <ObjectRef id="urn:uuid:d2345678-1234-1234-123456789012" />
689        <ClassificationNode  id = "automotiveNode" name = "Automotive"
690          description = "The Automotive sub-tree under Industry scheme"
691          parent = "urn:uuid:d2345678-1234-1234-123456789012"/>
692        <ClassificationNode id = "partSuppliersNode" name = "Parts Supplier"
693          description = "The Parts Supplier node under the Automotive node"
694          parent="automotiveNode" />
695        <ClassificationNode id = "engineSuppliersNode" name = "Engine Supplier"
696          description = "The Engine Supplier node under the Automotive node"
697          parent="automotiveNode" />
698
699        <!--
700        The following show submission of 2 Classifications of an object  that is already in
701        the registry using 2 ClassificationNodes. One ClassificationNode
702        is being submitted in this request (Japan) while the other is already in the registry.
703        -->
704        <Classification id = "japanClassification"
```

```
705        description = "Classifies object by /Geography/Asia/Japan node"
706        classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
707        classificationNode="japanNode" />
708      <Classification id = "classificationUsingExistingNode"
709        description = "Classifies object using a node in the registry"
710        classifiedObject="urn:uuid:a2345678-1234-1234-123456789012"
711        classificationNode="urn:uuid:e2345678-1234-1234-123456789012" />
712      <ObjectRef id="urn:uuid:e2345678-1234-1234-123456789012" />
713
714
715    </RegistryEntryList>
716  </SubmitObjectsRequest>
```

7.4 The Add Slots Protocol

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

721

722   **Figure 7: Add Slots Sequence Diagram**

723 In the event of success, the registry sends a RegistryResponse with a status of
724 "success" back to the client. In the event of failure, the registry sends a
725 RegistryResponse with a status of "failure" back to the client.

7.5 The Remove Slots Protocol

727 This section describes the protocol of the Registry Service that allows a client to remove
728 slots to a previously submitted registry entry using the ObjectManager.

729

730   **Figure 8: Remove Slots Sequence Diagram**

731   In the event of success, the registry sends a RegistryResponse with a status of
732   "success" back to the client. In the event of failure, the registry sends a
733   RegistryResponse with a status of "failure" back to the client.

7.6 The Approve Objects Protocol

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

739

740   **Figure 9: Approve Objects Sequence Diagram**

741 In the event of success, the registry sends a RegistryResponse with a status of
742 "success" back to the client. In the event of failure, the registry sends a
743 RegistryResponse with a status of "failure" back to the client.

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

7.7 The Deprecate Objects Protocol

747 This section describes the protocol of the Registry Service that allows a client to
748 deprecate one or more previously submitted repository items using the ObjectManager.
749 Once an object is deprecated, no new references (e.g. *new*Associations,
750 Classifications and ExternalLinks) to that object can be submitted. However, existing
751 references to a deprecated object continue to function normally.

752

753   **Figure 10: Deprecate Objects Sequence Diagram**

754 In the event of success, the registry sends a RegistryResponse with a status of
755 "success" back to the client. In the event of failure, the registry sends a
756 RegistryResponse with a status of "failure" back to the client.

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

7.8 The Remove Objects Protocol

760 This section describes the protocol of the Registry Service that allows a client to remove
761 one or more RegistryEntry instances and/or repository items using the ObjectManager.

762 The RemoveObjectsRequest message is sent by a client to remove RegistryEntry
763 instances and/or repository items. The RemoveObjectsRequest element includes an
764 XML attribute called *deletionScope* which is an enumeration that can have the values as
765 defined by the following sections.

7.8.1 Deletion Scope DeleteRepositoryItemOnly

766

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

7.8.2 Deletion Scope DeleteAll

770

771 This deletionScope specifies that the request should delete both the RegistryEntry and
772 the repository item for the specified registry entries. Only if all references (e.g.
773 Associations, Classifications, ExternalLinks) to a RegistryEntry have been removed, can
774 that RegistryEntry then be removed using a RemoveObjectsRequest with
775 deletionScope DeleteAll. Attempts to remove a RegistryEntry while it still has references
776 raises an error condition: InvalidRequestError.

777 The remove object protocol is expressed in UML notation as described in Appendix B.

778

779 **Figure 11: Remove Objects Sequence Diagram**

780 In the event of success, the registry sends a RegistryResponse with a status of
781 "success" back to the client. In the event of failure, the registry sends a
782 RegistryResponse with a status of "failure" back to the client.

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

8 Object Query Management Service

785

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

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

790   1. Browse and Drill Down Query

791   2. Filtered Query

792   3. SQL Query

793   The browse and drill down query in Section 8.1 and the filtered query mechanism in
794   Section 8.2 SHALL be supported by every Registry implementation. The SQL query
795   mechanism is an optional feature and MAY be provided by a registry implementation.
796   However, if a vendor provides an SQL query capability to an ebXML Registry it SHALL
797   conform to this document. As such this capability is a normative yet optional capability.

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

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

8.1   Browse and Drill Down Query Support

803   The browse and drill drown query style is supported by a set of interaction protocols
804   between the ObjectQueryManagerClient and the ObjectQueryManager. Sections 8.1.1,
805   8.1.2 and 8.1.3 describe these protocols.

8.1.1   Get Root Classification Nodes Request

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

813

814   **Figure 12: Get Root Classification Nodes Sequence Diagram**

815 In the event of success, the registry sends a GetRootClassificationNodeResponse with
816 a status of "success" back to the client. In the event of failure, the registry sends a
817 GetRootClassificationNodeResponse with a status of "failure" back to the client.

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

820 ### 8.1.2   Get Classification Tree Request

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

827

828

829 **Figure 14: Get Classification Tree Sequence Diagram**

830 In the event of success, the registry sends a GetClassificationTreeResponse with a
831 status of "success" back to the client. In the event of failure, the registry sends a
832 GetClassificationTreeResponse with a status of "failure" back to the client.

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

835 ### 8.1.3   Get Classified Objects Request

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

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

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

844   1. Either directly classified by the specified ClassificationNode

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

846   **8.1.3.1   Get Classified Objects Request Example**

847

848   **Figure 16: A Sample Geography Classification**

849   Let us say a classification tree has the structure shown in Figure 16:

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

854   • If the Asia node is specified in the GetClassifiedObjectsRequest then the
855   GetClassifiedObjectsResponse should include all RegistryEntries that are directly
856   classified by Asia *or* Japan *or* Korea.

857   • If the Japan *and* Korea nodes are specified in the GetClassifiedObjectsRequest
858   then the GetClassifiedObjectsResponse should include all RegistryEntries that
859   are directly classified by both Japan *and* Korea.

860   • If the North America *and* Asia node is specified in the
861   GetClassifiedObjectsRequest then the GetClassifiedObjectsResponse should
862   include all RegistryEntries that are directly classified by (North America *or* US)
863   *and* (Asia *or* Japan *or* Korea).

864

865   **Figure 17: Get Classified Objects Sequence Diagram**

866   In the event of success, the registry sends a GetClassifiedObjectsResponse with a
867   status of "success" back to the client. In the event of failure, the registry sends a
868   GetClassifiedObjectsResponse with a status of "failure" back to the client.

869 **8.2 Filter Query Support**

870 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
871 conforming Registry implementation. Each query alternative is directed against a single
872 class defined by the ebXML Registry Information Model (ebRIM). The result of such a
873 query is a set of identifiers for instances of that class. A FilterQuery may be a stand-
874 alone query or it may be the initial action of a ReturnRegistryEntry query or a
875 ReturnRepositoryItem query.

876 A client submits a FilterQuery, a ReturnRegistryEntry query, or a ReturnRepositoryItem
877 query to the ObjectQueryManager as part of an AdhocQueryRequest. The
878 ObjectQueryManager sends an AdhocQueryResponse back to the client, enclosing the
879 appropriate FilterQueryResponse, ReturnRegistryEntryResponse, or
880 ReturnRepositoryItemResponse specified herein. The sequence diagrams for
881 AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.4.

882 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a
883 hierarchy of classes derived from a single class and its associations with other classes
884 as defined by ebRIM. Each choice of a class pre-determines a virtual XML document
885 that can be queried as a tree. For example, let C be a class, let Y and Z be classes that
886 have direct associations to C, and let V be a class that is associated with Z. The ebRIM
887 Binding for C might be as in Figure 19.
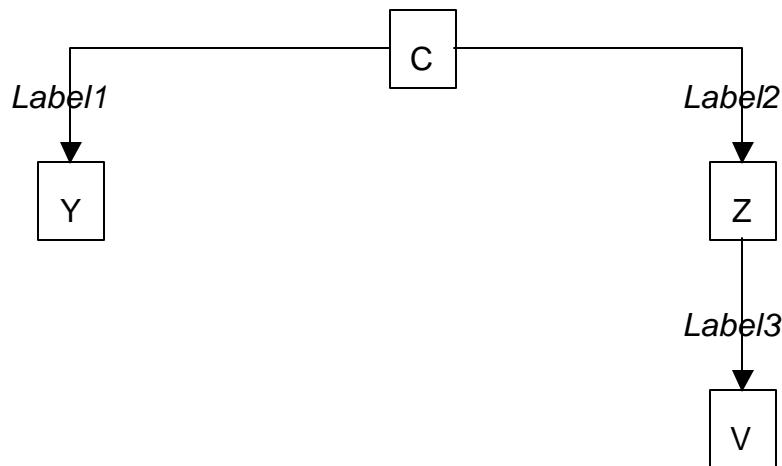
888
889   C
                Label1   Label2
890
891
892   Y   Z
893
894   Label3
895
896   V
897   **Figure 19: Example ebRIM Binding**

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

905 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter
906 is a restricted *predicate clause* over the attributes of a single class. The supported class
907 filters are specified in Section 8.2.9 and the supported predicate clauses are defined in
908 Section 8.2.10. A FilterQuery will be composed of elements that traverse the tree to
909 determine which branches satisfy the designated class filters, and the query result will
910 be the set of root node instances that support such a branch.

911 In the above example, the CQuery element will have three subelements, one a CFilter
912 on the C class to eliminate C instances that do not satisfy the predicate of the CFilter,
913 another a YFilter on the Y class to eliminate branches from C to Y where the target of
914 the association does not satisfy the YFilter, and a third to eliminate branches along a
915 path from C through Z to V. The third element is called a *branch* element because it
916 allows class filters on each class along the path from X to V. In general, a branch
917 element will have subelements that are themselves class filters, other branch elements,
918 or a full-blown query on the terminal class in the path.

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

924 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is
925 intended to be stable, the FilterQuery syntax is stable. However, if new structures are
926 added to the ebRIM, then the FilterQuery syntax and semantics can be extended at the
927 same time.

928 Support for FilterQuery is required of every conforming ebXML Registry implementation,
929 but other query options are possible. The Registry will hold a self-describing CPP that
930 identifies all supported AdhocQuery options. This profile is described in Section 6.1.

931 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.6 below identify the virtual
932 hierarchy for each FilterQuery alternative. The Semantic Rules for each query
933 alternative specify the effect of that binding on query semantics.

934 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a
935 way to structure an XML document as an expansion of the result of a
936 RegistryEntryQuery. The ReturnRegistryEntry element specified in Section 8.2.7 allows
937 one to specify what metadata one wants returned with each registry entry identified in
938 the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section
939 8.2.8 allows one to specify what repository items one wants returned based on their
940 relationships to the registry entries identified by the result of a RegistryEntryQuery.
941

941   **8.2.1 FilterQuery**

942   **Purpose**

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

947   **Definition**
948
```
949        <!ELEMENT FilterQuery
950         (     RegistryEntryQuery
951           |   AuditableEventQuery
952           |   ClassificationNodeQuery
953           |   RegistryPackageQuery
954           |   OrganizationQuery         )>
955
956        <!ELEMENT FilterQueryResult
957          (     RegistryEntryQueryResult
958           |   AuditableEventQueryResult
959           |   ClassificationNodeQueryResult
960           |   RegistryPackageQueryResult
961           |   OrganizationQueryResult    )>
962
963        <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
964
965        <!ELEMENT RegistryEntryView EMPTY >
966        <!ATTLIST RegistryEntryView
967           objectURN     CDATA     #REQUIRED
968           contentURI    CDATA     #IMPLIED
969           objectID      CDATA     #IMPLIED  >
970
971        <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
972
973        <!ELEMENT AuditableEventView EMPTY >
974        <!ATTLIST AuditableEventView
975           objectID      CDATA     #REQUIRED
976           timestamp     CDATA     #REQUIRED  >
977
978        <!ELEMENT ClassificationNodeQueryResult
979                    (ClassificationNodeView*)>
980
981        <!ELEMENT ClassificationNodeView EMPTY >
982        <!ATTLIST ClassificationNodeView
983           objectURN     CDATA     #REQUIRED
984           contentURI    CDATA     #IMPLIED
985           objectID      CDATA     #IMPLIED  >
986
987        <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
988
989        <!ELEMENT RegistryPackageView EMPTY >
990        <!ATTLIST RegistryPackageView
```

ebXML Registry Services Specification   Page 34

```
991         objectURN       CDATA       #REQUIRED
992         contentURI      CDATA       #IMPLIED
993         objectID        CDATA       #IMPLIED  >
994
995     <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
996
997     <!ELEMENT OrganizationView EMPTY >
998     <!ATTLIST OrganizationView
999         orgURN          CDATA       #REQUIRED
1000        objectID        CDATA       #IMPLIED  >
1001
1002
```

Semantic Rules

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

2. Each FilterQueryResult is a set of XML reference elements to identify each instance of the result set. Each XML attribute carries a value derived from the value of an attribute specified in the Registry Information Model as follows:

   a) objectID is the value of the ID attribute of the RegistryObject class,

   b) objectURN and orgURN are URN values derived from the object ID,

   c) contentURI is a URL value derived from the contentURI attribute of the RegistryEntry class,

   d) timestamp is a literal value to represent the value of the timestamp attribute of the AuditableEvent class.

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

4. If no error conditions are raised during execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to "success" and an appropriate query result element must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of the RegistryErrorList is set to "warning" and the serverity attribute of each RegistryError is set to "warning".

1027 **8.2.2 RegistryEntryQuery**

1028 **Purpose**

1029 To identify a set of registry entry instances as the result of a query over selected registry
1030 metadata.

1031 **ebRIM Binding**

1032
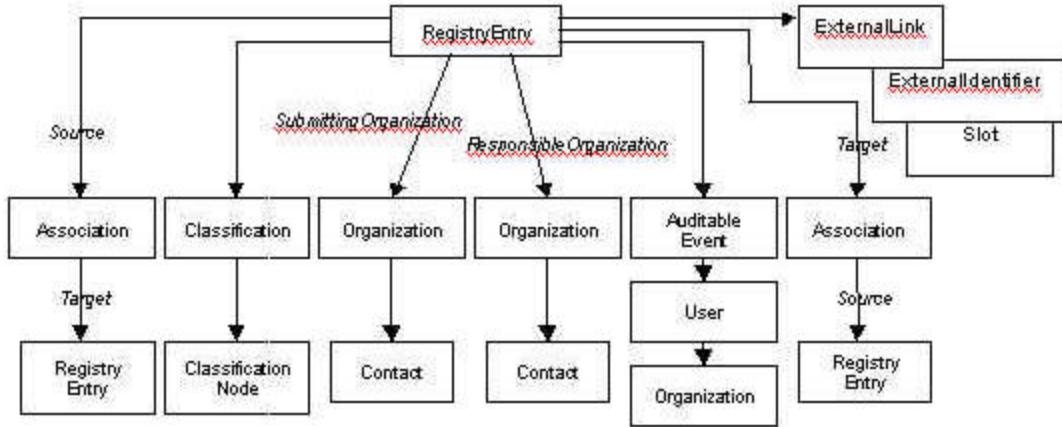
1033

1034 **Definition**
1035
```
1036      <!ELEMENT RegistryEntryQuery
1037        ( RegistryEntryFilter?,
1038          SourceAssociationBranch*,
1039          TargetAssociationBranch*,
1040          HasClassificationBranch*,
1041          SubmittingOrganizationBranch?,
1042          ResponsibleOrganizationBranch?,
1043          ExternalIdentifierFilter*,
1044          ExternalLinkFilter*,
1045          SlotFilter*,
1046          HasAuditableEventBranch*        )>
1047
1048      <!ELEMENT SourceAssociationBranch
1049        ( AssociationFilter?,
1050          RegistryEntryFilter?             )>
1051
1052      <!ELEMENT TargetAssociationBranch
1053        ( AssociationFilter?,
1054          RegistryEntryFilter?             )>
1055
1056      <!ELEMENT HasClassificationBranch
1057        ( ClassificationFilter?,
1058          ClassificationNodeFilter?        )>
```

ebXML Registry Services Specification   Page 36

1059
1060   <!ELEMENT SubmittingOrganizationBranch
1061   (OrganizationFilter?,
1062   ContactFilter?   )>
1063
1064   <!ELEMENT ResponsibleOrganizationBranch
1065   (OrganizationFilter?,
1066   ContactFilter?   )>
1067
1068   <!ELEMENT HasAuditableEventBranch
1069   (AuditableEventFilter?,
1070   UserFilter?,
1071   OrganizationFilter?   )>

1072   **Semantic Rules**

1073   1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The
1074   following steps will eliminate instances in RE that do not satisfy the conditions of the
1075   specified filters.

1076   a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below;
1077   otherwise, let x be a registry entry in RE. If x does not satisfy the
1078   RegistryEntryFilter as defined in Section 8.2.9, then remove x from RE.

1079   b) If a SourceAssociationBranch element is not specified, or if RE is empty, then
1080   continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1081   source object of some Association instance, then remove x from RE; otherwise,
1082   treat each SourceAssociationBranch element separately as follows:

1083   If no AssociationFilter is specified within SourceAssociationBranch, then let AF
1084   be the set of all Association instances that have x as a source object; otherwise,
1085   let AF be the set of Association instances that satisfy the AssociationFilter and
1086   have x as the source object. If AF is empty, then remove x from RE. If no
1087   RegistryEntryFilter is specified within SourceAssociationBranch, then let RET be
1088   the set of all RegistryEntry instances that are the target object of some element
1089   of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the
1090   RegistryEntryFilter and are the target object of some element of AF. If RET is
1091   empty, then remove x from RE.

1092   c) If a TargetAssociationBranch element is not specified, or if RE is empty, then
1093   continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1094   target object of some Association instance, then remove x from RE; otherwise,
1095   treat each TargetAssociationBranch element separately as follows:

1096   If no AssociationFilter is specified within TargetAssociationBranch, then let AF be
1097   the set of all Association instances that have x as a target object; otherwise, let
1098   AF be the set of Association instances that satisfy the AssociationFilter and have
1099   x as the target object. If AF is empty, then remove x from RE. If no
1100   RegistryEntryFilter is specified within TargetAssociationBranch, then let RES be
1101   the set of all RegistryEntry instances that are the source object of some element
1102   of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the
1103   RegistryEntryFilter and are the source object of some element of AF. If RES is
1104   empty, then remove x from RE.

d) If a HasClassificationBranch element is not specified, or if RE is empty, then
1105   continue below; otherwise, let x be a remaining registry entry in RE. If x is not the
1106   source object of some Classification instance, then remove x from RE; otherwise,
1107   treat each HasClassificationBranch element separately as follows:
1108

1109   If no ClassificationFilter is specified within the HasClassificationBranch, then let
1110   CL be the set of all Classification instances that have x as a source object;
1111   otherwise, let CL be the set of Classification instances that satisfy the
1112   ClassificationFilter and have x as the source object. If CL is empty, then remove
1113   x from RE. If no ClassificationNodeFilter is specified within
1114   HasClassificationBranch, then let CN be the set of all ClassificationNode
1115   instances that are the target object of some element of CL; otherwise, let CN be
1116   the set of RegistryEntry instances that satisfy the ClassificationNodeFilter and
1117   are the target object of some element of CL. If CN is empty, then remove x from
1118   RE.

e) If a SubmittingOrganizationBranch element is not specified, or if RE is empty,
1119   then continue below; otherwise, let x be a remaining registry entry in RE. If x
1120   does not have a submitting organization, then remove x from RE. If no
1121   OrganizationFilter is specified within SubmittingOrganizationBranch, then let SO
1122   be the set of all Organization instances that are the submitting organization for x;
1123   otherwise, let SO be the set of Organization instances that satisfy the
1124   OrganizationFilter and are the submitting organization for x. If SO is empty, then
1125   remove x from RE. If no ContactFilter is specified within
1126   SubmittingOrganizationBranch, then let CT be the set of all Contact instances
1127   that are the contacts for some element of SO; otherwise, let CT be the set of
1128   Contact instances that satisfy the ContactFilter and are the contacts for some
1129   element of SO. If CT is empty, then remove x from RE.
1130

1131   f)   If a ResponsibleOrganizationBranch element is not specified, or if RE is empty,
1132   then continue below; otherwise, let x be a remaining registry entry in RE. If x
1133   does not have a responsible organization, then remove x from RE. If no
1134   OrganizationFilter is specified within ResponsibleOrganizationBranch, then let
1135   RO be the set of all Organization instances that are the responsible organization
1136   for x; otherwise, let RO be the set of Organization instances that satisfy the
1137   OrganizationFilter and are the responsible organization for x. If RO is empty, then
1138   remove x from RE. If no ContactFilter is specified within
1139   SubmittingOrganizationBranch, then let CT be the set of all Contact instances
1140   that are the contacts for some element of RO; otherwise, let CT be the set of
1141   Contact instances that satisfy the ContactFilter and are the contacts for some
1142   element of RO. If CT is empty, then remove x from RE.

1143   g)   If an ExternalLinkFilter element is not specified, or if RE is empty, then continue
1144   below; otherwise, let x be a remaining registry entry in RE. If x is not linked to
1145   some ExternalLink instance, then remove x from RE; otherwise, treat each
1146   ExternalLinkFilter element separately as follows:

1147   Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter and
1148   are linked to x. If EL is empty, then remove x from RE.

1149   h)   If an ExternalIdentifierFilter element is not specified, or if RE is empty, then
1150   continue below; otherwise, let x be a remaining registry entry in RE. If x is not
1151   linked to some ExternalIdentifier instance, then remove x from RE; otherwise,
1152   treat each ExternalIdentifierFilter element separately as follows:

1153   Let EI be the set of ExternalIdentifier instances that satisfy the
1154   ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RE.

1155   i)   If a SlotFilter element is not specified, or if RE is empty, then continue below;
1156   otherwise, let x be a remaining registry entry in RE. If x is not linked to some Slot
1157   instance, then remove x from RE; otherwise, treat each SlotFilter element
1158   separately as follows:

1159   Let SL be the set of Slot instances that satisfy the SlotFilter and are linked to x. If
1160   SL is empty, then remove x from RE.

1161   j)   If a HasAuditableEventBranch element is not specified, or if RE is empty, then
1162   continue below; otherwise, let x be a remaining registry entry in RE. If x is not
1163   linked to some AuditableEvent instance, then remove x from RE; otherwise, treat
1164   each HasAuditableEventBranch element separately as follows:

1165   If an AuditableEventFilter is not specified within HasAuditableEventBranch, then
1166   let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the
1167   set of AuditableEvent instances that satisfy the AuditableEventFilter and are
1168   auditable events for x. If AE is empty, then remove x from RE. If a UserFilter is
1169   not specified within HasAuditableEventBranch, then let AI be the set of all User
1170   instances linked to an element of AE; otherwise, let AI be the set of User
1171   instances that satisfy the UserFilter and are linked to an element of AE.

1172   If AI is empty, then remove x from RE. If an OrganizationFilter is not specified
1173   within HasAuditableEventBranch, then let OG be the set of all Organization
1174   instances that are linked to an element of AI; otherwise, let OG be the set of
1175   Organization instances that satisfy the OrganizationFilter and are linked to an
1176   element of AI. If OG is empty, then remove x from RE.

1177   2. If RE is empty, then raise the warning: *registry entry query result is empty*.

1178   3. Return RE as the result of the RegistryEntryQuery.

1179

1180   **Examples**

1181   A client wants to establish a trading relationship with XYZ Corporation and wants to
1182   know if they have registered any of their business documents in the Registry. The
1183   following query returns a set of registry entry identifiers for currently registered items
1184   submitted by any organization whose name includes the string "XYZ". It does not return
1185   any registry entry identifiers for superceded, replaced, deprecated, or withdrawn items.
1186

```
1187    <RegistryEntryQuery>
1188        <RegistryEntryFilter>
1189            status EQUAL "Approved"        -- code by Clause, Section 8.2.10
1190        </RegistryEntryFilter>
1191        <SubmittingOrganizationBranch>
1192            <OrganizationFilter>
1193                name CONTAINS "XYZ"        -- code by Clause, Section 8.2.10
1194            </OrganizationFilter>
1195        </SubmittingOrganizationBranch>
1196    </RegistryEntryquery>
```

1197

1198   A client is using the United Nations Standard Product and Services Classification
1199   (UNSPSC) scheme and wants to identify all companies that deal with products
1200   classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client
1201   knows that companies have registered their party profile documents in the Registry, and
1202   that each profile has been classified by the products the company deals with. The
1203   following query returns a set of registry entry identifiers for profiles of companies that
1204   deal with integrated circuit components.
1205

```
1206    <RegistryEntryQuery>
1207        <RegistryEntryFilter>
1208            objectType EQUAL "CPP" AND        -- code by Clause, Section 8.2.10
1209            status EQUAL "Approved"
1210        </RegistryEntryFilter>
1211        <HasClassificationBranch>
1212            <ClassificationNodeFilter>
1213                id STARTSWITH "urn:un:spsc:321118"   -- code by Clause, Section 8.2.10
1214            </ClassificationNodeFilter>
1215        <HasClassificationBranch>
1216    </RegistryEntryQuery>
```

1217 A client application needs all items that are classified by two different classification
1218 schemes, one based on "Industry" and another based on "Geography". Both schemes
1219 have been defined by ebXML and are registered. The root nodes of each scheme are
1220 identified by "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The
1221 following query identifies registry entries for all registered items that are classified by
1222 "Industry/Automotive" and by "Geography/Asia/Japan".

```
1223
1224      <RegistryEntryQuery>
1225         <HasClassificationBranch>
1226            <ClassificationNodeFilter>
1227               id STARTSWITH "urn:ebxml:cs:industry" AND
1228               path EQUAL "Industry/Automotive"        -- code by Clause, Section 8.2.10
1229            </ClassificationNodeFilter>
1230            <ClassificationNodeFilter>
1231               id STARTSWITH "urn:ebxml:cs:geography" AND
1232               path EQUAL "Geography/Asia/Japan"      -- code by Clause, Section 8.2.10
1233            </ClassificationNodeFilter>
1234         </HasClassificationBranch>
1235      </RegistryEntryQuery>
```

1236 A client application wishes to identify all registry Package instances that have a given
1237 registry entry as a member of the package. The following query identifies all registry
1238 packages that contain the registry entry identified by URN "urn:path:myitem" as a
1239 member:

```
1240
1241      <RegistryEntryQuery>
1242         <RegistryEntryFilter>
1243            objectType EQUAL "RegistryPackage"        -- code by Clause, Section 8.2.10
1244         </RegistryEntryFilter>
1245         <SourceAssociationBranch>
1246            <AssociationFilter>                        -- code by Clause, Section 8.2.10
1247               associationType EQUAL "HasMember" AND
1248               targetObject EQUAL "urn:path:myitem"
1249            </AssociationFilter>
1250         </SourceAssociationBranch>
1251      </RegistryEntryQuery>
```

1252 A client application wishes to identify all ClassificationNode instances that have some
1253 given keyword as part of their name or description. The following query identifies all
1254 registry classification nodes that contain the keyword "transistor" as part of their name
1255 or as part of their description.

```
1256
1257      <RegistryEntryQuery>
1258         <RegistryEntryFilter>
1259            ObjectType="ClassificationNode" AND
1260            (name CONTAINS "transistor" OR      -- code by Clause, Section 8.2.10
1261            description CONTAINS "transistor")
1262         </RegistryEntryFilter>
1263      </RegistryEntryQuery>
1264
```

1264 ## 8.2.3 AuditableEventQuery

1265 **Purpose**

1266 To identify a set of auditable event instances as the result of a query over selected
1267 registry metadata.

1268 ebRIM Binding
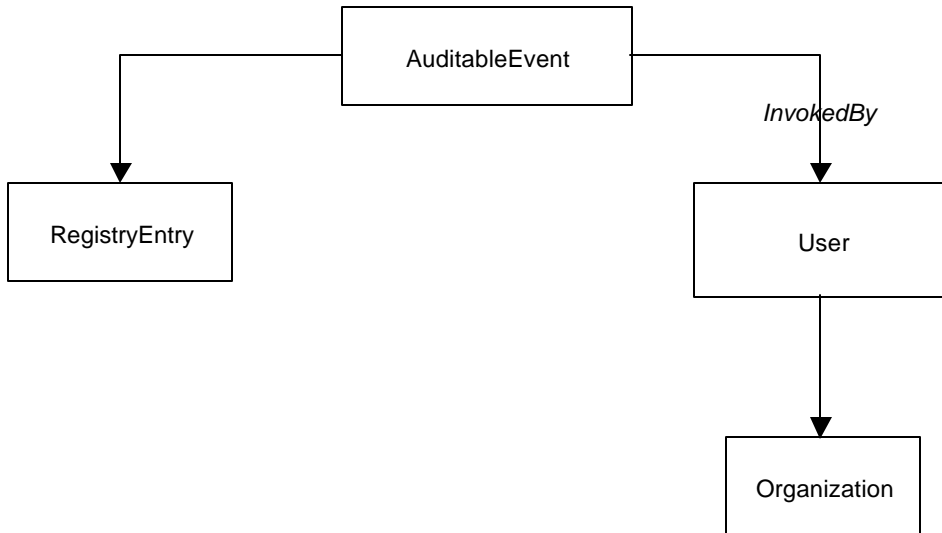
1269 **Definition**

1270
```
1271      <!ELEMENT AuditableEventQuery
1272        (   AuditableEventFilter?,
1273            RegistryEntryQuery*,
1274            InvokedByBranch?     )>
1275
1276      <!ELEMENT InvokedByBranch
1277        (   UserFilter?,
1278            OrganizationQuery?   )>
```
1279

1280 **Semantic Rules**

1281 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The
1282   following steps will eliminate instances in AE that do not satisfy the conditions of the
1283   specified filters.

1284

ebXML Registry Services Specification   Page 42

1285   a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below;
1286   otherwise, let x be an auditable event in AE. If x does not satisfy the
1287   AuditableEventFilter as defined in Section 8.2.9, then remove x from AE.

1288   b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue
1289   below; otherwise, let x be a remaining auditable event in AE. Treat each
1290   RegistryEntryQuery element separately as follows:

1291   Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If
1292   x is not an auditable event for some registry entry in RE, then remove x from AE.

1293   c) If an InvokedByBranch element is not specified, or if AE is empty, then continue
1294   below; otherwise, let x be a remaining auditable event in AE.

1295 Let u be the user instance that invokes x. If a UserFilter element is specified within the
1296 InvokedByBranch, and if u does not satisfy that filter, then remove x from AE; otherwise,
1297 continue below.

1298   If an OrganizationQuery element is not specified within the InvokedByBranch,
1299   then continue below; otherwise, let OG be the set of Organization instances that
1300   are identified by the organization attribute of u and are in the result set of the
1301   OrganizationQuery. If OG is empty, then remove x from AE.

1302 2. If AE is empty, then raise the warning: *auditable event query result is empty*.

1303 3. Return AE as the result of the AuditableEventQuery.

1304

Examples

1306 A Registry client has registered an item and it has been assigned a URN identifier
1307 "urn:path:myitem". The client is now interested in all events since the beginning of the
1308 year that have impacted that item. The following query will return a set of
1309 AuditableEvent identifiers for all such events.

1310
```
1311    <AuditableEventquery>
1312       <AuditableEventFilter>
1313          timestamp GE "2001-01-01" AND          -- code by Clause, Section 8.2.10
1314          registryEntry EQUAL "urn:path:myitem"
1315       </AuditableEventFilter>
1316    </AuditableEventQuery>
```
1317

1318 A client company has many registered objects in the Registry. The Registry allows
1319 events submitted by other organizations to have an impact on your registered items,
1320 e.g. new classifications and new associations. The following query will return a set of
1321 identifiers for all auditable events, invoked by some other party, that had an impact on
1322 an item submitted by "myorg" and for which "myorg" is the responsible organization.

1323
```
1324    <AuditableEventQuery>
1325       <RegistryEntryQuery>
```

```
1326              <SubmittingOrganizationBranch>
1327                 <OrganizationFilter>
1328                    id EQUAL "urn:somepath:myorg"      -- code by Clause, Section 8.2.10
1329                 </OrganizationFilter>
1330              </SubmittingOrganizationBranch>
1331              <ResponsibleOrganizationBranch>
1332                 <OrganizationFilter>
1333                    id EQUAL "urn:somepath:myorg"      -- code by Clause, Section 8.2.10
1334                 </OrganizationFilter>
1335              </ResponsibleOrganizationBranch>
1336         </RegistryEntryQuery>
1337         <InvokedByBranch>
1338            <OrganizationQuery>
1339               <OrganizationFilter>
1340                  id -EQUAL "urn:somepath:myorg"      -- code by Clause, Section 8.2.10
1341               </OrganizationFilter>
1342            </OrganizationQuery>
1343         </InvokedByBranch>
1344      </AuditableEventQuery>
1345
```

1345   **8.2.4   ClassificationNodeQuery**

1346   **Purpose**

1347   To identify a set of classification node instances as the result of a query over selected
1348   registry metadata.

1349   **ebRIM Binding**
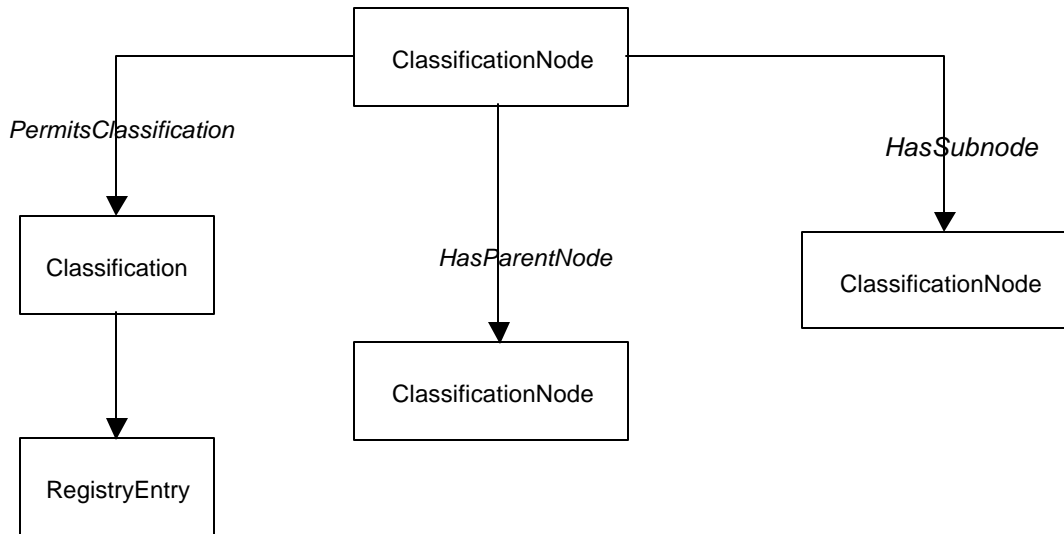
1350   **Definition**

```
1351
1352      <!ELEMENT ClassificationNodeQuery
1353         (  ClassificationNodeFilter?,
1354            PermitsClassificationBranch*,
1355            HasParentNode?,
1356            HasSubnode*              )>
1357
1358      <!ELEMENT PermitsClassificationBranch
1359         (  ClassificationFilter?,
1360            RegistryEntryQuery?       )>
1361
1362      <!ELEMENT HasParentNode
1363         (  ClassificationNodeFilter?,
1364            HasParentNode?            )>
1365
1366      <!ELEMENT HasSubnode
1367         (  ClassificationNodeFilter?,
1368            HasSubnode*               )>
1369
1370
```

1371 **Semantic Rules**

1372 1. Let CN denote the set of all persistent ClassificationNode instances in the Registry.
1373   The following steps will eliminate instances in CN that do not satisfy the conditions of
1374   the specified filters.

1375   a) If a ClassificationNodeFilter is not specified, or if CN is empty, then continue
1376   below; otherwise, let x be a classification node in CN. If x does not satisfy the
1377   ClassificationNodeFilter as defined in Section 8.2.9, then remove x from AE.

1378   b) If a PermitsClassificationBranch element is not specified, or if CN is empty, then
1379   continue below; otherwise, let x be a remaining classification node in CN. If x is
1380   not the target object of some Classification instance, then remove x from CN;
1381   otherwise, treat each PermitsClassificationBranch element separately as follows:

1382   If no ClassificationFilter is specified within the PermitsClassificationBranch
1383   element, then let CL be the set of all Classification instances that have x as the
1384   target object; otherwise, let CL be the set of Classification instances that satisfy
1385   the ClassificationFilter and have x as the target object. If CL is empty, then
1386   remove x from CN. If no RegistryEntryQuery is specified within the
1387   PermitsClassificationBranch element, then let RES be the set of all RegistryEntry
1388   instances that are the source object of some classification instance in CL;
1389   otherwise, let RE be the result set of the RegistryEntryQuery as defined in
1390   Section 8.2.2 and let RES be the set of all instances in RE that are the source
1391   object of some classification in CL. If RES is empty, then remove x from CN.

1392   c) If a HasParentNode element is not specified, or if CN is empty, then continue
1393   below; otherwise, let x be a remaining classification node in CN and execute the
1394   following paragraph with n=x.

1395   Let n be a classification node instance. If n does not have a parent node (i.e. if n
1396   is a root node), then remove x from CN. Let p be the parent node of n. If a
1397   ClassificationNodeFilter element is directly contained in HasParentNode and if p
1398   does not satisfy the ClassificationNodeFilter, then remove x from CN.

1399   If another HasParentNode element is directly contained within this
1400   HasParentNode element, then repeat the previous paragraph with n=p.

1401   d) If a HasSubnode element is not specified, or if CN is empty, then continue below;
1402   otherwise, let x be a remaining classification node in CN. If x is not the parent
1403   node of some ClassificationNode instance, then remove x from CN; otherwise,
1404   treat each HasSubnode element separately and execute the following paragraph
1405   with n = x.

1406   Let n be a classification node instance. If a ClassificationNodeFilter is not
1407   specified within the HasSubnode element then let CNC be the set of all
1408   classification nodes that have n as their parent node; otherwise, let CNC be the
1409   set of all classification nodes that satisfy the ClassificationNodeFilter and have n
1410   as their parent node. If CNC is empty then remove x from CN; otherwise, let y be
1411   an element of CNC and continue with the next paragraph.

1412   If the HasSubnode element is terminal, i.e. if it does not directly contain another
1413   HasSubnode element, then continue below; otherwise, repeat the previous
1414   paragraph with the new HasSubnode element and with n = y.

1415   2. If CN is empty, then raise the warning: *classification node query result is empty*.

1416   3. Return CN as the result of the ClassificationNodeQuery.

1417

1418   **Examples**

1419   A client application wishes to identify all classification nodes defined in the Registry that
1420   are root nodes and have a name that contains the phrase "product code" or the phrase
1421   "product type". Note: By convention, if a classification node has no parent (i.e. is a root
1422   node), then the parent attribute of that instance is set to null and is represented as a
1423   literal by a zero length string.
1424
1425   ```
       <ClassificationNodeQuery>
1426   <ClassificationNodeFilter>
1427   (name CONTAINS "product code" OR   -- code by Clause, Section 8.2.10
1428   name CONTAINS "product type") AND
1429   parent EQUAL ""
1430   </ClassificationNodeFilter>
1431   </ClassificationNodeQuery>
       ```

1432

1433   A client application wishes to identify all of the classification nodes at the third level of a
1434   classification scheme hierarchy. The client knows that the URN  identifier for the root
1435   node is "urn:ebxml:cs:myroot". The following query identifies all nodes at the second
1436   level under "myroot" (i.e. third level overall).
1437
1438   ```
       <ClassificationNodeQuery>
1439       <HasParentNode>
1440          <HasParentNode>
1441             <ClassificationNodeFilter>
1442                id EQ "urn:ebxml:cs:myroot"   -- code by Clause, Section 8.2.10
1443             </ClassificationNodeFilter>
1444          </HasParentNode>
1445       </HasParentNode>
1446    </ClassificationNodeQuery>
       ```
1447

1447   **8.2.5 RegistryPackageQuery**

1448   **Purpose**

1449   To identify a set of registry package instances as the result of a query over selected
1450   registry metadata.

1451   **ebRIM Binding**

Package

HasMember
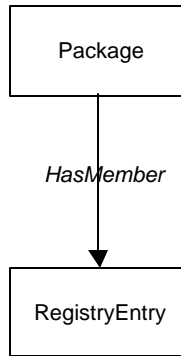
RegistryEntry

1452   **Definition**
1453
1454   `<!ELEMENT RegistryPackageQuery`
1455   `(    PackageFilter?,`
1456   `HasMemberBranch*      )>`
1457
1458   `<!ELEMENT HasMemberBranch`
1459   `(    RegistryEntryQuery?   )>`
1460

1461   **Semantic Rules**

1462   1. Let RP denote the set of all persistent Package instances in the Registry. The
1463   following steps will eliminate instances in RP that do not satisfy the conditions of the
1464   specified filters.

1465   a) If a PackageFilter is not specified, or if RP is empty, then continue below;
1466   otherwise, let x be a package instance in RP. If x does not satisfy the
1467   PackageFilter as defined in Section 8.2.9, then remove x from RP.

1468   b) If a HasMemberBranch element is not directly contained in the
1469   RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x
1470   be a remaining package instance in RP. If x is an empty package, then remove x
1471   from RP; otherwise, treat each HasMemberBranch element separately as
1472   follows:

1473

1474   If a RegistryEntryQuery element is not directly contained in the
1475   HasMemberBranch element, then let PM be the set of all RegistryEntry instances
1476   that are members of the package x; otherwise, let RE be the set of RegistryEntry
1477   instances returned by the RegistryEntryQuery as defined in Section 8.2.2 and let
1478   PM be the subset of RE that are members of the package x. If PM is empty, then
1479   remove x from RP.

1480   2. If RP is empty, then raise the warning: *registry package query result is empty.*

1481   3. Return RP as the result of the RegistryPackageQuery.

1482

1483 **Examples**

1484 A client application wishes to identify all package instances in the Registry that contain
1485 an Invoice extrinsic object as a member of the package.
1486

```
1487     <RegistryPackageQuery>
1488        <HasMemberBranch>
1489           <RegistryEntryQuery>
1490              <RegistryEntryFilter>
1491                 objectType EQ "Invoice"        -- code by Clause, Section 8.2.10
1492              </RegistryEntryFilter>
1493           </RegistryEntryQuery>
1494        </HasMemberBranch>
1495     </RegistryPackageQuery>
```

1496

1497 A client application wishes to identify all package instances in the Registry that are not
1498 empty.
1499

```
1500     <RegistryEntryQuery>
1501        <HasMemberBranch/>
1502     </RegistryEntryQuery>
```

1503

1504 A client application wishes to identify all package instances in the Registry that are
1505 empty. Since the RegistryPackageQuery is not set up to do negations, clients will have
1506 to do two separate RegistryPackageQuery requests, one to find all packages and
1507 another to find all non-empty packages, and then do the set difference themselves.
1508 Alternatively, they could do a more complex RegistryEntryQuery and check that the
1509 packaging association between the package and its members is non-existent.

1510 Note: A registry package is an intrinsic RegistryEntry instance that is completely
1511 determined by its associations with its members. Thus a RegistryPackageQuery can
1512 always be re-specified as an equivalent RegistryEntryQuery using appropriate "Source"
1513 and "Target" associations. However, the equivalent RegistryEntryQuery is often more
1514 complicated to write.

1515

1515 **8.2.6 OrganizationQuery**

1516 **Purpose**

1517 To identify a set of organization instances as the result of a query over selected registry
1518 metadata.

1519 **ebRIM Binding**
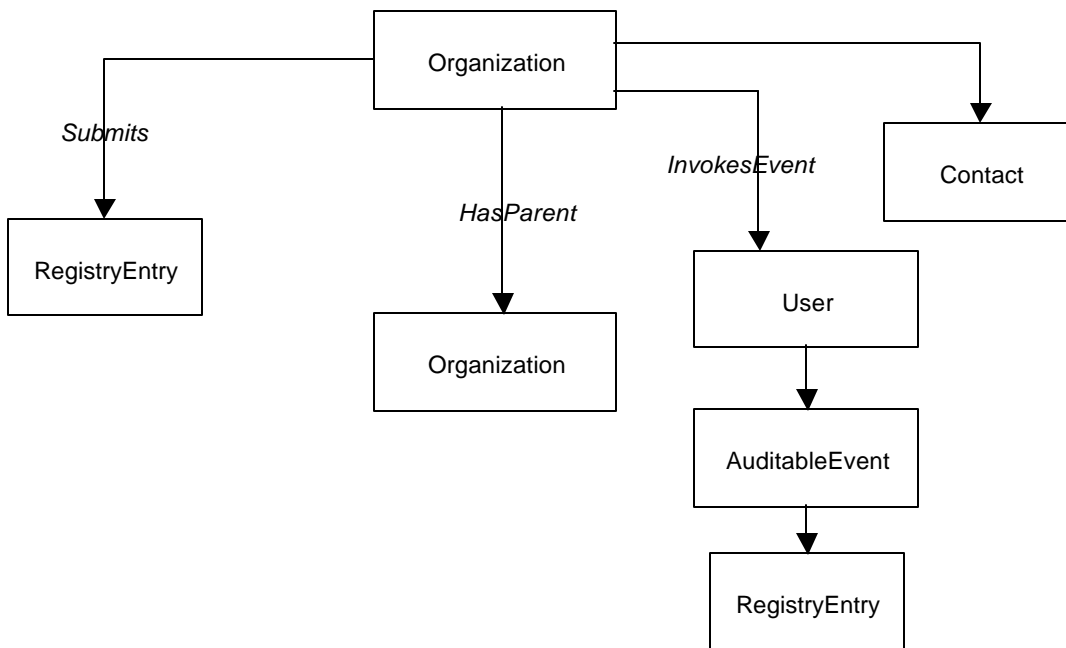
1520

1521 **Definition**
1522
```
1523      <!ELEMENT OrganizationQuery
1524        (   OrganizationFilter?,
1525            SubmitsRegistryEntry*,
1526            HasParentOrganization?,
1527            InvokesEventBranch*,
1528            ContactFilter          )>
1529
1530      <!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>
1531
1532      <!ELEMENT HasParentOrganization
1533        (   OrganizationFilter?,
1534            HasParentOrganization?   )>
1535
1536      <!ELEMENT InvokesEventBranch
1537        (   UserFilter?,
1538            AuditableEventFilter?,
1539            RegistryEntryQuery?      )>
```

ebXML Registry Services Specification   Page 50

1540 **Semantic Rules**

1541 1. Let ORG denote the set of all persistent Organization instances in the Registry. The
1542   following steps will eliminate instances in ORG that do not satisfy the conditions of
1543   the specified filters.

1544   a) If an OrganizationFilter element is not directly contained in the
1545   OrganizationQuery element, or if ORG is empty, then continue below; otherwise,
1546   let x be an organization instance in ORG. If x does not satisfy the
1547   OrganizationFilter as defined in Section 8.2.9, then remove x from RP.

1548   b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery,
1549   or if ORG is empty, then continue below; otherwise, consider each
1550   SubmitsRegistryEntry element separately as follows:

1551   If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element,
1552   then let RES be the set of all RegistryEntry instances that have been submitted
1553   to the Registry by organization x; otherwise, let RE be the result of the
1554   RegistryEntryQuery as defined in Section 8.2.2 and let RES be the set of all
1555   instances in RE that have been submitted to the Registry by organization x. If
1556   RES is empty, then remove x from ORG.

1557   c) If a HasParentOrganization element is not specified within the
1558   OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute
1559   the following paragraph with o = x:

1560   Let o be an organization instance. If an OrganizationFilter is not specified within
1561   the HasParentOrganization and if o has no parent (i.e. if o is a root organization
1562   in the Organization hierarchy), then remove x from ORG; otherwise, let p be the
1563   parent organization of o. If p does not satisfy the OrganizationFilter, then remove
1564   x from ORG.

1565   If another HasParentOrganization element is directly contained within this
1566   HasParentOrganization element, then repeat the previous paragraph with o = p.

1567   d) If an InvokesEventBranch element is not specified within the OrganizationQuery,
1568   or if ORG is empty, then continue below; otherwise, consider each
1569   InvokesEventBranch element separately as follows:

1570   If an UserFilter is not specified, and if x is not the submitting organization of some
1571   AuditableEvent instance, then remove x from ORG. If an AuditableEventFilter is
1572   not specified, then let AE be the set of all AuditableEvent instances that have x
1573   as the submitting organization; otherwise, let AE be the set of AuditableEvent
1574   instances that satisfy the AuditableEventFilter and have x as the submitting
1575   organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is
1576   not specified in the InvokesEventBranch element, then let RES be the set of all
1577   RegistryEntry instances associated with an event in AE; otherwise, let RE be the
1578   result set of the RegistryEntryQuery, as specified in Section 8.2.2, and let RES
1579   be the subset of RE of entries submitted by x. If RES is empty, then remove x
1580   from ORG.

1581 e) If a ContactFilter is not specified within the OrganizationQuery, or if ORG is
1582   empty, then continue below; otherwise, consider each ContactFilter separately as
1583   follows:

1584   Let CT be the set of Contact instances that satisfy the ContactFilter and are the
1585   contacts for organization x. If CT is empty, then remove x from ORG.

1586 2. If ORG is empty, then raise the warning: *organization query result is empty*.

1587 3. Return ORG as the result of the OrganizationQuery.

1588

1589 **Examples**

1590 A client application wishes to identify a set of organizations, based in France, that have
1591 submitted a PartyProfile extrinsic object this year.
1592

```
1593      <OrganizationQuery>
1594         <OrganizationFilter>
1595            country EQUAL "France"          -- code by Clause, Section 8.2.10
1596         </OrganizationFilter>
1597         <SubmitsRegistryEntry>
1598            <RegistryEntryQuery>
1599               <RegistryEntryFilter>
1600                  objectType EQUAL "CPP"   -- code by Clause, Section 8.2.10
1601               </RegistryEntryFilter>
1602               <HasAuditableEventBranch>
1603                  <AuditableEventFilter>
1604                     timestamp GE "2001-01-01"   -- code by Clause, Section 8.2.10
1605                  </AuditableEventFilter>
1606               </HasAuditableEventBranch>
1607            </RegistryEntryQuery>
1608         </SubmitsRegistryEntry>
1609      </OrganizationQuery>
```

1610

1611 A client application wishes to identify all organizations that have XYZ, Corporation as a
1612 parent. The client knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is
1613 no guarantee that subsidiaries of XYZ have a URN that uses the same format, so a full
1614 query is required.
1615

```
1616      <OrganizationQuery>
1617         <HasParentOrganization>
1618            <OrganizationFilter>
1619               id EQUAL "urn:ebxml:org:xyz"     -- code by Clause, Section 8.2.10
1620            </OrganizationFilter>
1621         </HasParentOrganization>
1622      </OrganizationQuery>
```

1623

1623 ### 8.2.7 ReturnRegistryEntry

1624 **Purpose**

1625 To construct an XML document that contains selected registry metadata associated with
1626 the registry entries identified by a RegistryEntryQuery. NOTE: Initially, the
1627 RegistryEntryQuery could be the URN identifier for a single registry entry.

1628 **Definition**

```
1629
1630    <!ELEMENT ReturnRegistryEntry
1631      (   RegistryEntryQuery,
1632          WithClassifications?,
1633          WithSourceAssociations?,
1634          WithTargetAssociations?,
1635          WithAuditableEvents?,
1636          WithExternalLinks?                )>
1637
1638    <!ELEMENT WithClassifications  ( ClassificationFilter? )>
1639    <!ELEMENT WithSourceAssociations  ( AssociationFilter? )>
1640    <!ELEMENT WithTargetAssociations  ( AssociationFilter? )>
1641    <!ELEMENT WithAuditableEvents  ( AuditableEventFilter? )>
1642    <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
1643
1644    <!ELEMENT ReturnRegistryEntryResult
1645        (   RegistryEntryMetadata*)>
1646
1647    <!ELEMENT RegistryEntryMetadata
1648        ( RegistryEntry,
1649          Classification*,
1650          SourceAssociations?,
1651          TargetAssociations?,
1652          AuditableEvent*,
1653          ExternalLink*                          )>
1654
1655    <!ELEMENT SourceAssociations ( Association* )>
1656    <!ELEMENT TargetAssociations ( Association* )>
```

1657 **Semantic Rules**

1658 1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink
1659   elements contained in the ReturnRegistryEntryResult are defined by the ebXML
1660   Registry DTD specified in Appendix A.

1661 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in
1662   Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let
1663   S be the set of warnings and errors returned. If any element in S is an error
1664   condition, then stop execution and return the same set of warnings and errors along
1665   with the ReturnRegistryEntryResult.

1666 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the
1667   ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*.
1668   Add this warning to the error list returned by the RegistryEntryQuery and return this
1669   enhanced error list with the ReturnRegistryEntryResult.

1670 4. For each registry entry E referenced by an element of R, use the attributes of E to
1671   create a new RegistryEntry element as defined in Appendix A. Then create a new
1672   RegistryEntryMetadata element as defined above to be the parent element of that
1673   RegistryEntry element.

1674 5. If no With option is specified, then the resulting RegistryEntryMetadata element has
1675   no Classification, SourceAssociations, TargetAssociations, AuditableEvent, or
1676   ExternalData subelements. The set of RegistryEntryMetadata elements, with the
1677   Error list from the RegistryEntryQuery, is returned as the ReturnRegistryEntryResult.

1678 6. If WithClassifications is specified, then for each E in R do the following: If a
1679   ClassificationFilter is not present, then let C be any classification instance linked to
1680   E; otherwise, let C be a classification instance linked to E that satisfies the
1681   ClassificationFilter (Section 8.2.9). For each such C, create a new Classification
1682   element as defined in Appendix A. Add these Classification elements to their parent
1683   RegistryEntryMetadata element.

1684 7. If WithSourceAssociations is specified, then for each E in R do the following: If an
1685   AssociationFilter is not present, then let A be any association instance whose source
1686   object is E; otherwise, let A be an association instance that satisfies the
1687   AssociationFilter (Section 8.2.9) and whose source object is E. For each such A,
1688   create a new Association element as defined in Appendix A. Add these Association
1689   elements as subelements of the WithSourceAssociations and add that element to its
1690   parent RegistryEntryMetadata element.

1691 8. If WithTargetAssociations is specified, then for each E in R do the following: If an
1692   AssociationFilter is not present, then let A be any association instance whose target
1693   object is E; otherwise, let A be an association instance that satisfies the
1694   AssociationFilter (Section 8.2.9) and whose target object is E. For each such A,
1695   create a new Association element as defined in Appendix A. Add these Association
1696   elements as subelements of the WithTargetAssociations and add that element to its
1697   parent RegistryEntryMetadata element.

1698 9. If WithAuditableEvents is specified, then for each E in R do the following: If an
1699   AuditableEventFilter is not present, then let A be any auditable event instance linked
1700   to E; otherwise, let A be any auditable event instance linked to E that satisfies the
1701   AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent
1702   element as defined in Appendix A. Add these AuditableEvent elements to their
1703   parent RegistryEntryMetadata element.

1704 10. If WithExternalLinks is specified, then for each E in R do the following: If an
1705   ExternalLinkFilter is not present, then let L be any external link instance linked to E;
1706   otherwise, let L be any external link instance linked to E that satisfies the
1707   ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink
1708   element as defined in Appendix A. Add these ExternalLink elements to their parent
1709   RegistryEntryMetadata element.

1710 11. If any warning or error condition results, then add the code and the message to the
1711   RegistryResponse element that includes the RegistryEntryQueryResult.

1712 12. Return the set of RegistryEntryMetadata elements as the content of the
1713   ReturnRegistryEntryResult.

1714

1715 **Examples**

1716 A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by
1717 XYZ some time ago. Its URN identifier is "urn:com:xyz:po:325". The customer wishes to
1718 check on the current status of that DTD, especially if it has been superceded or
1719 replaced, and get all of its current classifications. The following query request will return
1720 an XML document with the registry entry for the existing DTD as the root, with all of its
1721 classifications, and with associations to registry entries for any items that have
1722 superceded or replaced it.
1723

```
1724      <ReturnRegistryEntry>
1725          <RegistryEntryQuery>
1726              <RegistryEntryFilter>
1727                  id EQUAL "urn:com:xyz:po:325"        -- code by Clause, Section 8.2.10
1728              </RegistryEntryFilter>
1729          </RegistryEntryQuery>
1730          <WithClassifications/>
1731          <WithSourceAssociations>
1732              <AssociationFilter>                      -- code by Clause, Section 8.2.10
1733                  associationType EQUAL "SupercededBy" OR
1734                  associationType EQUAL "ReplacedBy"
1735              </AssociationFilter>
1736          </WithSourceAssociations>
1737      </ReturnRegistryEntry>
```

1738

1739 A client of the Registry registered an XML DTD several years ago and is now thinking of
1740 replacing it with a revised version. The identifier for the existing DTD is
1741 "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the
1742 existing DTD. The client desires a list of all registered items that use the existing DTD
1743 so they can assess the impact of an incompatible change. The following query returns
1744 an XML document that is a list of all RegistryEntry elements that represent registered
1745 items that use, contain, or extend the given DTD. The document also links each
1746 RegistryEntry element in the list to an element for the identified association.

1747

```
1748
1749      <ReturnRegistryEntry>
1750         <RegistryEntryQuery>
1751            <SourceAssociationBranch>
1752               <AssociationFilter>                    -- code by Clause, Section 8.2.10
1753                  associationType EQUAL "Contains" OR
1754                  associationType EQUAL "Uses" OR
1755                  associationType EQUAL "Extends"
1756               </AssociationFilter>
1757               <RegistryEntryFilter>                  -- code by Clause, Section 8.2.10
1758                  id EQUAL "urn:xyz:dtd:po97"
1759               </RegistryEntryFilter>
1760            </SourceAssociationBranch>
1761         </RegistryEntryQuery>
1762         <WithSourceAssociations>
1763            <AssociationFilter>                       -- code by Clause, Section 8.2.10
1764               associationType EQUAL "Contains" OR
1765               associationType EQUAL "Uses" OR
1766               associationType EQUAL "Extends"
1767            </AssociationFilter>
1768         </WithSourceAssociations>
1769      </ReturnRegistryEntry>
1770
```

1771 A user has been browsing the registry and has found a registry entry that describes a
1772 package of core-components that should solve the user's problem. The package URN
1773 identifier is "urn:com:cc:pkg:ccstuff". Now the user wants to know what's in the package.
1774 The following query returns an XML document with a registry entry for each member of
1775 the package along with that member's Uses and HasMemberBranch associations.

```
1776
1777      <ReturnRegistryEntry>
1778         <RegistryEntryQuery>
1779            <TargetAssociationBranch>
1780               <AssociationFilter>                 -- code by Clause, Section 8.2.10
1781                  associationType EQUAL "HasMember"
1782               </AssociationFilter>
1783               <RegistryEntryFilter>               -- code by Clause, Section 8.2.10
1784                  id EQUAL " urn:com:cc:pkg:ccstuff "
1785               </RegistryEntryFilter>
1786            </TargetAssociationBranch>
1787         </RegistryEntryQuery>
1788         <WithSourceAssociations>
1789            <AssociationFilter>                       -- code by Clause, Section 8.2.10
1790               associationType EQUAL "HasMember" OR
1791               associationType EQUAL "Uses"
1792            </AssociationFilter>
1793         </WithSourceAssociations>
1794      </ReturnRegistryEntry>
1795
```

1795   ## 8.2.8   ReturnRepositoryItem

1796   **Purpose**

1797   To construct an XML document that contains one or more repository items, and some
1798   associated metadata, by submitting a RegistryEntryQuery to the registry/repository that
1799   holds the desired objects. NOTE: Initially, the RegistryEntryQuery could be the URN
1800   identifier for a single registry entry.

1801   **Definition**

```
1802
1803        <!ELEMENT ReturnRepositoryItem
1804        (  RegistryEntryQuery,
1805           RecursiveAssociationOption?,
1806           WithDescription?              )>
1807
1808        <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1809        <!ATTLIST RecursiveAssociationOption
1810             depthLimit   CDATA   #IMPLIED   >
1811
1812        <!ELEMENT AssociationType EMPTY >
1813        <!ATTLIST AssociationType
1814             role   CDATA   #REQUIRED  >
1815
1816        <!ELEMENT WithDescription EMPTY >
1817
1818        <!ELEMENT ReturnRepositoryItemResult
1819           ( RepositoryItem*)>
1820
1821        <!ELEMENT RepositoryItem
1822          (    ClassificationScheme
1823            | RegistryPackage
1824            | ExtrinsicObject
1825            | WithdrawnObject
1826            | ExternalLinkItem      )>
1827        <!ATTLIST RepositoryItem
1828          identifier     CDATA    #REQUIRED
1829          name           CDATA    #REQUIRED
1830          contentURI     CDATA    #REQUIRED
1831          objectType     CDATA    #REQUIRED
1832          status         CDATA    #REQUIRED
1833          stability      CDATA    #REQUIRED
1834          description    CDATA    #IMPLIED   >
1835
1836        <!ELEMENT ExtrinsicObject  (#PCDATA) >
1837        <!ATTLIST ExtrinsicObject
1838          byteEncoding   CDATA     "Base64"  >
1839
1840        <!ELEMENT WithdrawnObject EMPTY >
1841
1842        <!ELEMENT ExternalLinkItem EMPTY >
1843
1844
```

1845

Semantic Rules

1846

1847 1. If the RecursiveOption element is not present , then set Limit=0. If the
1848   RecursiveOption element is present, interpret its depthLimit attribute as an integer
1849   literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0
1850   means that no recursion occurs. A Limit of -1 means that recursion occurs
1851   indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive
1852   integer, then stop execution and raise the exception: *invalid depth limit*, otherwise,
1853   set Limit=N, where N is that positive integer. A Limit of N means that exactly N
1854   recursive steps will be executed unless the process terminates prior to that limit.

1855 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned
1856   as part of the Retur nRepositoryItemResult. Initially Result is empty. Semantic rules
1857   4 through 10 determine the content of Result.

1858 3. If the WithDescription element is present, then set WSD="yes"; otherwise, set
1859   WSD="no".

1860 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in
1861   Section 8.2.2, and let R be the result set of identifiers for registry entry instances. Let
1862   S be the set of warnings and errors returned. If any element in S is an error
1863   condition, then stop execution and return the same set of warnings and errors along
1864   with the ReturnRepositoryItem Result.

1865 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from
1866   R. After execution of these rules, if Depth is now equal to Limit, then return the
1867   content of Result as the set of RepositoryItem elements in the
1868   ReturnRepositoryItemResult element; otherwise, continue with Semantic Rule 8.

1869 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the
1870   following:

1871 a) If E.contentURI references a repository item in this registry/repository, then
1872   create a new RepositoryItem element, with values for its a ttributes derived as
1873   specified in Semantic Rule 7.

1874   1) If E.objectType="ClassificationScheme", then put the referenced
1875   ClassificationScheme DTD as the subelement of this RepositoryItem.
1876   [NOTE: Requires DTD specification!]

1877   2) If E.objectType="RegistryPackage", then put the referenced
1878   RegistryPackage DTD as the subelement of this RepositoryItem. [NOTE:
1879   Requires DTD specification!]

1880   3) Otherwise, i.e., if the object referenced by E has an unknown internal
1881   structure, then put the content of the repository item as the #PCDATA of a
1882   new ExtrinsicObject subelement of this RepositoryItem.

1883   b) If E.objectURL references a registered object in some other registry/repository,
1884   then create a new RepositoryItem element, with values for its attributes derived
1885   as specified in Semantic Rule 7, and create a new ExternalLink element as the
1886   subelement of this RepositoryItem.

1887   c) If E.objectURL is void, i.e. the object it would have referenced has been
1888   withdrawn, then create a new RepositoryItem element, with values for its
1889   attributes derived as specified in Semantic Rule 7, and create a new
1890   WithdrawnObject element as the subelement of this RepositoryItem.

1891 7. Let E be a registry entry and let RO be the RepositoryItem element created in
1892   Semantic Rule 6. Set the attributes of RO to the values derived from the
1893   corresponding attributes of E. If WSD="yes", include the value of the description
1894   attribute; otherwise, do not include it. Insert this new RepositoryItem element into the
1895   Result set.

1896 8. Let R be defined as in Semantic Rule 3. Execute Semantic Rule 9 with Y as the set
1897   of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.

1898 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty
1899   set of RegistryEntry instances. For each registry entry E in Y, and for each
1900   AssociationType A of the RecursiveAssociationOption, do the following:

1901   a) Let Z be the set of target items E' linked to E under association instances having
1902   E as the source object, E' as the target object, and A as the AssociationType.

1903   b) Add the elements of Z to NextLevel.

1904 10. Let X be the set of new registry entries that are in NextLevel but are not yet
1905   represented in the Result set.

1906   Case:

1907   a) If X is empty, then return the content of Result as the set of RepositoryItem
1908   elements in the ReturnRepositoryItemResult element.

1909   b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set.
1910   When finished, add the elements of X to Y and set Depth=Depth+1. If Depth is
1911   now equal to Limit, then return the content of Result as the set of RepositoryItem
1912   elements in the ReturnRepositoryItemResult element; otherwise, repeat
1913   Semantic Rules 9 and 10 with the new set Y of registry entries.

1914 11. If any exception, warning, or other status condition results during the execution of
1915   the above, then return appropriate RegistryError elements in the RegistryResult
1916   associated with the ReturnRepositoryItemResult element created in Semantic Rule 5
1917   or Semantic Rule 10.

1918 **Examples**

1919 A registry client has found a registry entry for a core-component item. The item's URN
1920 identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many
1921 other registered items. The client desires the collection of all items needed for a

1922 complete implementation of "goodthing". The following query returns an XML document
1923 that is a collection of all needed items.

1924
```
1925    <ReturnRepositoryItem>
1926       <RegistryEntryQuery>
1927          <RegistryEntryFilter>                  -- code by Clause, Section 8.2.10
1928             id EQUAL "urn:ebxml:cc:goodthing"
1929          </RegistryEntryFilter>
1930       </RegistryEntryQuery>
1931       <RecursiveAssociationOption>
1932          <AssociationType  role="Uses"  />
1933          <AssociationType  role="ValidatesTo"  />
1934       </RecursiveAssociationOption>
1935    </ReturnRepositoryItem>
```
1936

1937 A registry client has found a reference to a core-component routine
1938 ("urn:ebxml:cc:rtn:nice87") that implements a given business process. The client knows
1939 that all routines have a required association to its defining UML specification. The
1940 following query returns both the routine and its UML specification as a collection of two
1941 items in a single XML document.

1942
```
1943    <ReturnRepositoryItem>
1944       <RegistryEntryQuery>
1945          <RegistryEntryFilter>                  -- code by Clause, Section 8.2.10
1946             id EQUAL "urn:ebxml:cc:rtn:nice87"
1947          </RegistryEntryFilter>
1948       </RegistryEntryQuery>
1949       <RecursiveAssociationOption   depthLimit="1" >
1950          <AssociationType  role="ValidatesTo"  />
1951       </RecursiveAssociationOption>
1952    </ReturnRepositoryItem>
```
1953

1954 A user has been told that the 1997 version of the North American Industry Classification
1955 System (NAICS) is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The
1956 following query would retrieve the complete classification scheme, with all 1810 nodes,
1957 as an XML document that validates to a classification scheme DTD.

1958
```
1959    <ReturnRepositoryItem>
1960       <RegistryEntryQuery>
1961          <RegistryEntryFilter>                  -- code by Clause, Section 8.2.10
1962             id EQUAL "urn:nist:cs:naics-1997"
1963          </RegistryEntryFilter>
1964       </RegistryEntryQuery>
1965    </ReturnRepositoryItem>
```

1966

1967 Note: The ReturnRepositoryItemResult would include a single RepositoryItem that
1968 consists of a ClassificationScheme document whose content is determined by the URL
1969 ftp://xsun.sdct.itl.nist.gov/regrep/scheme/naics.txt.
1970
ebXML Registry Services Specification   Page 60

1970   **8.2.9 Registry Filters**

1971   **Purpose**

1972   To identify a subset of the set of all persistent instances of a given registry class.

1973   **Definition**
1974
1975   `<!ELEMENT ObjectFilter ( Clause )>`
1976
1977   `<!ELEMENT RegistryEntryFilter ( Clause )>`
1978
1979   `<!ELEMENT IntrinsicObjectFilter ( Clause )>`
1980
1981   `<!ELEMENT ExtrinsicObjectFilter ( Clause )>`
1982
1983   `<!ELEMENT PackageFilter ( Clause )>`
1984
1985   `<!ELEMENT OrganizationFilter ( Clause )>`
1986
1987   `<!ELEMENT ContactFilter ( Clause )>`
1988
1989   `<!ELEMENT ClassificationNodeFilter ( Clause )>`
1990
1991   `<!ELEMENT AssociationFilter ( Clause )>`
1992
1993   `<!ELEMENT ClassificationFilter ( Clause )>`
1994
1995   `<!ELEMENT ExternalLinkFilter ( Clause )>`
1996
1997   `<!ELEMENT ExternalIdentifierFilter ( Clause )>`
1998
1999   `<!ELEMENT SlotFilter ( Clause )>`
2000
2001   `<!ELEMENT AuditableEventFilter ( Clause )>`
2002
2003   `<!ELEMENT UserFilter ( Clause )>`

2004

2005   **Semantic Rules**

2006   1. The Clause element is defined in Section 8.2.10, Clause.

2007   2. For every ObjectFilter XML element, the leftArgument attribute of any containing
2008   SimpleClause shall identify a public attribute of the RegistryObject UML class
2009   defined in [ebRIM]. If not, raise exception: *object attribute error*. The ObjectFilter
2010   returns a set of identifiers for RegistryObject instances whose attribute values
2011   evaluate to *True* for the Clause predicate.

2012   3. For every RegistryEntryFilter XML element, the leftArgument attribute of any
2013   containing SimpleClause shall identify a public attribute of the RegistryEntry UML
2014   class defined in [ebRIM].

ebXML Registry Services Specification   Page 61

2015   If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a
2016   set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*
2017   for the Clause predicate.

2018   4. For every IntrinsicObjectFilter XML element, the leftArgument attribute of any
2019   containing SimpleClause shall identify a public attribute of the IntrinsicObject UML
2020   class defined in [ebRIM]. If not, raise exception: *intrinsic object attribute error*. The
2021   IntrinsicObjectFilter returns a set of identifiers for IntrinsicObject instances whose
2022   attribute values evaluate to *True* for the Clause predicate.

2023   5. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any
2024   containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML
2025   class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The
2026   ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose
2027   attribute values evaluate to *True* for the Clause predicate.

2028   6. For every PackageFilter XML element, the leftArgument attribute of any containing
2029   SimpleClause shall identify a public attribute of the Package UML class defined in
2030   [ebRIM]. If not, raise exception: *package attribute error*. The PackageFilter returns a
2031   set of identifiers for Package instances whose attribute values evaluate to *True* for
2032   the Clause predicate.

2033   7. For every OrganizationFilter XML element, the leftArgument attribute of any
2034   containing SimpleClause shall identify a public attribute of the Organization or
2035   PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization*
2036   *attribute error*. The OrganizationFilter returns a set of identifiers for Organization
2037   instances whose attribute values evaluate to *True* for the Clause predicate.

2038   8. For every ContactFilter XML element, the leftArgument attribute of any containing
2039   SimpleClause shall identify a public attribute of the Contact or PostalAddress UML
2040   class defined in [ebRIM]. If not, raise exception: *contact attribute error*. The
2041   ContactFilter returns a set of identifiers for Contact instances whose attribute values
2042   evaluate to *True* for the Clause predicate.

2043   9. For every ClassificationNodeFilter XML element, the leftArgument attribute of any
2044   containing SimpleClause shall identify a public attribute of the ClassificationNode
2045   UML class defined in [ebRIM]. If not, raise exception: *classification node attribute*
2046   *error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode
2047   instances whose attribute values evaluate to *True* for the Clause predicate.

2048   10. For every AssociationFilter XML element, the leftArgument attribute of any
2049   containing SimpleClause shall identify a public attribute of the Association UML
2050   class defined in [ebRIM]. If not, raise exception: *association attribute error*. The
2051   AssociationFilter returns a set of identifiers for Association instances whose attribute
2052   values evaluate to *True* for the Clause predicate.

2053   11. For every ClassificationFilter XML element, the leftArgument attribute of any
2054   containing SimpleClause shall identify a public attribute of the Classification UML
2055   class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The
2056   ClassificationFilter returns a set of identifiers for Classification instances whose
2057   attribute values evaluate to *True* for the Clause predicate.

2058   12. For every ExternalLinkFilter XML element, the leftArgument attribute of any
2059   containing SimpleClause shall identify a public attribute of the ExternalLink UML
2060   class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The
2061   ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose
2062   attribute values evaluate to *True* for the Clause predicate.

2063   13. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any
2064   containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML
2065   class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The
2066   ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances
2067   whose attribute values evaluate to *True* for the Clause predicate.

2068   14. For every SlotFilter XML element, the leftArgument attribute of any containing
2069   SimpleClause shall identify a public attribute of the Slot UML class defined in
2070   [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of
2071   identifiers for Slot instances whose attribute values evaluate to *True* for the Clause
2072   predicate.

2073   15. For every AuditableEventFilter XML element, the leftArgument attribute of any
2074   containing SimpleClause shall identify a public attribute of the AuditableEvent UML
2075   class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The
2076   AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose
2077   attribute values evaluate to *True* for the Clause predicate.

2078   16. For every UserFilter XML element, the leftArgument attribute of any containing
2079   SimpleClause shall identify a public attribute of the User UML class defined in
2080   [ebRIM]. If not, raise exception: *auditable identity attribute error*. The UserFilter
2081   returns a set of identifiers for User instances whose attribute values evaluate to *True*
2082   for the Clause predicate.

2083

2084   **Example**

2085   The following is a complete example of RegistryEntryQuery combined with Clause
2086   expansion of RegistryEntryFilter to return a set of RegistryEntry instances whose
2087   objectType attibute is "CPP" and whose status attribute is "Approved".

2088
```
2089        <RegistryEntryQuery>
2090           <RegistryEntryFilter>
2091              <Clause>
2092                 <CompoundClause    connectivePredicate="And" >
2093                    <Clause>
2094                       <SimpleClause   leftArgument="objectType" >
2095                    <StringClause   stringPredicate="equal" >CPP</StringClause>
```

```
2096                    </SimpleClause>
2097                  </Clause>
2098                  <Clause>
2099                    <SimpleClause  leftArgument="status" >
2100              <StringClause  stringPredicate="equal" >Approved</StringClause>
2101                    </SimpleClause>
2102                  </Clause>
2103               </CompoundClause>
2104            </Clause>
2105        </RegistryEntryFilter>
2106    </RegistryEntryQuery>
2107
2108
```

2108 **8.2.10 XML Clause Constraint Representation**

2109 **Purpose**

2110 The simple XML FilterQuery utilizes a formal XML structure based on *Predicate*
2111 *Clauses*. Predicate Clauses are utilized to formally define the constraint mechanism,
2112 and are referred to simply as ***Clauses*** in this specification.

2113 **Conceptual UML Diagram**

2114 The following is a conceptual diagram outlining the Clause base structure. It is
2115 expressed in UML for visual depiction.
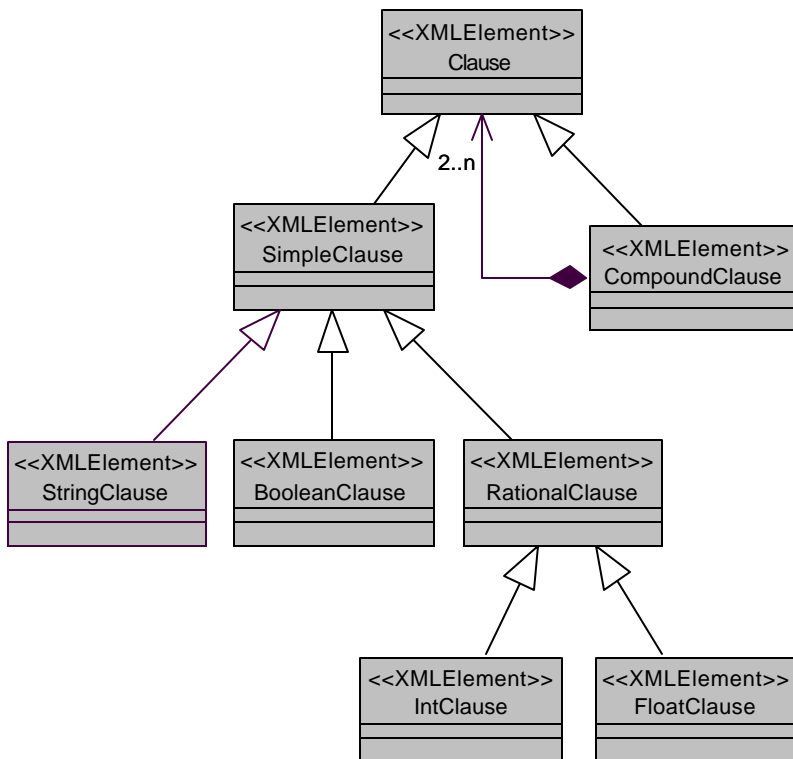
2116

2117   **Figure 20: The Clause base structure**

2118 **Semantic Rules**

2119 *Predicates* and *Arguments* are combined into a "LeftArgument - Predicate -
2120 RightArgument" format to form a *Clause*. There are two types of Clauses:
2121 *SimpleClauses* and *CompoundClauses*.

2122 SimpleClauses

2123   A SimpleClause always defines the leftArgument as a text string, sometimes referred to
2124   as the *Subject* of the Clause. SimpleClause itself is incomplete (abstract) and must be
2125   extended. SimpleClause is extended to support BooleanClause, StringClause, and
2126   RationalClause (abstract).

2127   BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
2128   boolean. StringClause defines the predicate as an enumerated attribute of appropriate
2129   string-compare operations and a right argument as the element's text data. Rational
2130   number support is provided through a common RationalClause providing an
2131   enumeration of appropriate rational number compare operations, which is further
2132   extended to IntClause and FloatClause, each with appropriate signatures for the right
2133   argument.

2134   CompoundClauses

2135   A CompoundClause contains two or more Clauses (Simple or Compound) and a
2136   connective predicate. This provides for arbitrarily complex Clauses to be formed.

2137

2138   **Definition**

2139
```
2140      <!ELEMENT Clause ( SimpleClause | CompoundClause )>
2141
2142      <!ELEMENT SimpleClause
2143         ( BooleanClause | RationalClause | StringClause )>
2144      <!ATTLIST SimpleClause
2145         leftArgument  CDATA #REQUIRED >
2146
2147      <!ELEMENT CompoundClause ( Clause, Clause+ )>
2148      <!ATTLIST CompoundClause
2149         connectivePredicate ( And | Or ) #REQUIRED>
2150
2151      <!ELEMENT BooleanClause EMPTY >
2152      <!ATTLIST BooleanClause
2153         booleanPredicate ( True | False ) #REQUIRED>
2154
2155      <!ELEMENT RationalClause ( IntClause | FloatClause )>
2156      <!ATTLIST RationalClause
2157         logicalPredicate ( LE | LT | GE | GT | EQ | NE ) #REQUIRED >
2158
2159      <!ELEMENT IntClause ( #PCDATA )
2160      <!ATTLIST IntClause
2161         e-dtype NMTOKEN #FIXED 'int' >
2162
2163      <!ELEMENT FloatClause ( #PCDATA )>
2164      <!ATTLIST FloatClause
2165         e-dtype NMTOKEN #FIXED 'float' >
2166
2167      <!ELEMENT StringClause ( #PCDATA )>
2168      <!ATTLIST StringClause
2169         stringPredicate
2170             ( contains | -contains |
2171                startswith | -startswith |
```

```
2172              equal | -equal
2173              endswith | -endswith ) #REQUIRED >
```

2174

2175 **Examples**

2176 Simple BooleanClause: "Smoker" = True

2177
```
2178    <?xml version="1.0" encoding="UTF-8"?>
2179    <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2180    <Clause>
2181      <SimpleClause leftArgument="Smoker">
2182        <BooleanClause booleanPredicate="True"/>
2183      </SimpleClause>
2184    </Clause>
```
2185

2186 Simple StringClause: "Smoker" contains "mo"

2187
```
2188    <?xml version="1.0" encoding="UTF-8"?>
2189    <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2190    <Clause>
2191      <SimpleClause leftArgument="Smoker">
2192        <StringClause stringcomparepredicate="contains">
2193          mo
2194        </StringClause>
2195      </SimpleClause>
2196    </Clause>
```

2197

2198 Simple IntClause: "Age" >= 7

2199
```
2200    <?xml version="1.0" encoding="UTF-8"?>
2201    <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2202    <Clause>
2203      <SimpleClause leftArgument="Age">
2204        <RationalClause logicalPredicate="GE">
2205          <IntClause e-dtype="int">7</IntClause>
2206        </RationalClause>
2207      </SimpleClause>
2208    </Clause>
```
2209

2210 Simple FloatClause: "Size" = 4.3

2211
```
2212    <?xml version="1.0" encoding="UTF-8"?>
2213    <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2214    <Clause>
2215      <SimpleClause leftArgument="Size">
2216        <RationalClause logicalPredicate="E">
2217          <FloatClause e-dtype="float">4.3</FloatClause>
2218        </RationalClause>
```

ebXML Registry Services Specification   Page 67

```
2219        </SimpleClause>
2220      </Clause>
2221
2222  Compound with two Simples (("Smoker" = False)AND("Age" =< 45))
2223
2224      <?xml version="1.0" encoding="UTF-8"?>
2225      <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2226      <Clause>
2227        <CompoundClause connectivePredicate="And">
2228          <Clause>
2229            <SimpleClause leftArgument="Smoker">
2230              <BooleanClause booleanPredicate="False"/>
2231            </SimpleClause>
2232          </Clause>
2233          <Clause>
2234            <SimpleClause leftArgument="Age">
2235              <RationalClause logicalPredicate="EL">
2236                <IntClause e-dtype="int">45</IntClause>
2237              </RationalClause>
2238            </SimpleClause>
2239          </Clause>
2240        </CompoundClause>
2241      </Clause>
2242
2243  Coumpound with one Simple and one Compound
2244  ( ("Smoker" = False)And(("Age" =< 45)Or("American"=True)) )
2245
2246      <?xml version="1.0" encoding="UTF-8"?>
2247      <!DOCTYPE Clause SYSTEM "Clause.dtd" >
2248      <Clause>
2249        <CompoundClause connectivePredicate="And">
2250          <Clause>
2251            <SimpleClause leftArgument="Smoker">
2252              <BooleanClause booleanPredicate="False"/>
2253            </SimpleClause>
2254          </Clause>
2255          <Clause>
2256            <CompoundClause connectivePredicate="Or">
2257              <Clause>
2258                <SimpleClause leftArgument="Age">
2259                  <RationalClause logicalPredicate="EL">
2260                    <IntClause e-dtype="int">45</IntClause>
2261                  </RationalClause>
2262                </SimpleClause>
2263              </Clause>
2264              <Clause>
2265                <SimpleClause leftArgument="American">
2266                  <BooleanClause booleanPredicate="True"/>
2267                </SimpleClause>
2268              </Clause>
```

```
2269              </CompoundClause>
2270            </Clause>
2271          </CompoundClause>
2272       </Clause>
```

8.3 SQL Query Support

2274 The Registry may optionally support an SQL based query capability that is designed for
2275 Registry clients that demand more complex query capability. The optional SQLQuery
2276 element in the AdhocQueryRequest allows a client to submit complex SQL queries
2277 using a declarative query language.

2278 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper
2279 subset of the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992,
2280 Database Language SQL [SQL], extended to include `<sql invoked routines>`
2281 (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-
2282 defined routines defined in template form in Appendix C.3. The exact syntax of the
2283 Registry query language is defined by the BNF grammar in C.1.

2284 Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement
2285 to use relational databases in a Registry implementation.

8.3.1 SQL Query Syntax Binding To [ebRIM]

2287 SQL Queries are defined based upon the query syntax in in Appendix C.1 and a fixed
2288 relational schema defined in Appendix C.3. The relational schema is an algorithmic
2289 binding to [ebRIM] as described in the following sections.

8.3.1.1   Interface and Class Binding

2291 A subset of the Interface and class names defined in [ebRIM] map to table names that
2292 may be queried by an SQL query. Appendix C.3 defines the names of the ebRIM
2293 interfaces and classes that may be queried by an SQL query.

2294 The algorithm used to define the binding of [ebRIM] classes to table definitions in
2295 Appendix C.3 is as follows:

- Only those classes and interfaces that have concrete instances are mapped to
  relational tables. This results in intermediate interfaces in the inheritance
  hierarchy, such as RegistryObject and IntrinsicObject, to not map to SQL tables.
  An exception to this rule is RegistryEntry, which is defined next.

- A special view called RegistryEntry is defined to allow SQL queries to be made
  against RegistryEntry instances. This is the only interface defined in [ebRIM] that
  does not have concrete instances but is queryable by SQL queries.

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

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

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

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

8.3.1.2   Accessor Method To Attribute Binding

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

8.3.1.3   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 interface 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].

8.3.1.4   Reference Attribute Binding

A few of the [ebRIM] interface methods return references to instances of interfaces or classes defined by [ebRIM]. For example, the getAccessControlPolicy method 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 8.3.1.3. The data type for the column is UUID as defined in Appendix C.3.

When a reference attribute value holds a null reference, it maps to a null value in the SQL binding and may be tested with the <null specification> as defined by [SQL].

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

8.3.1.5   Complex Attribute Binding

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

2342   The SQL query schema algorithmically maps such complex attributes as multiple
2343   primitive attributes within the parent table. The mapping simply flattens out the entity
2344   class attributes within the parent table. The attribute name for the flattened attributes
2345   are composed of a concatenation of attribute names in the refernce chain. For example
2346   Organization has a contact attribute of type Contact. Contact has an address attribute of
2347   type PostalAddress. PostalAddress has a String attribute named city. This city attribute
2348   will be named contact_address_city.

2349   ### 8.3.1.6   Collection Attribute Binding

2350   A few of the [ebRIM] interface methods return a collection of references to instances of
2351   interfaces or classes defined by [ebRIM]. For example, the getPackages method of the
2352   ManagedObject class returns a Collection of references to instances of Packages that
2353   the object is a member of.

2354   Such collection attributes in [ebRIM] classes have been mapped to stored procedures in
2355   Appendix C.3 such that these stored procedures return a collection of id attribute
2356   values. The returned value of these stored procedures can be treated as the result of a
2357   table sub-query in SQL.

2358   These stored procedures may be used as the right-hand-side of an SQL IN clause to
2359   test for membership of an object in such collections of references.

2360   ## 8.3.2   Semantic Constraints On Query Syntax

2361   This section defines simplifying constraints on the query syntax that cannot be
2362   expressed in the BNF for the query syntax. These constraints must be applied in the
2363   semantic analysis of the query.

2364   1. Class names and attribute names must be processed in a case insensitive manner.

2365   2. The syntax used for stored procedure invocation must be consistent with the syntax
2366   of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].

2367   3. For this version of the specification, the SQL select column list consists of exactly
2368   one column, and must always be t.id, where t is a table reference in the FROM
2369   clause.

2370   ## 8.3.3   SQL Query Results

2371   The results of an SQL query is always an ObjectRefList as defined by the
2372   AdHocQueryResponse in 8.4. This means the result of an SQL query is always a
2373   collection of references to instances of a sub-class of the RegistryObject interface in
2374   [ebRIM]. This is reflected in a semantic constraint that requires that the SQL select
2375   column specified must always be an id column in a table in Appendix C.3 for this
2376   version of the specification.

2377 ### 8.3.4 Simple Metadata Based Queries

2378 The simplest form of an SQL query is based upon metadata attributes specified for a
2379 single class within [ebRIM]. This section gives some examples of simple metadata
2380 based queries.

2381 For example, to get the collection of ExtrinsicObjects whose name contains the word
2382 'Acme' and that have a version greater than 1.3, the following query predicates must be
2383 supported:

2384
```
2385  SELECT id FROM ExtrinsicObject  WHERE name LIKE '%Acme%' AND
2386          majorVersion >= 1 AND
2387          (majorVersion >= 2 OR  minorVersion > 3);
```

2388 Note that the query syntax allows for conjugation of simpler predicates into more
2389 complex queries as shown in the simple example above.

2390 ### 8.3.5 RegistryEntry Queries

2391 Given the central role played by the RegistryEntry interface in ebRIM, the schema for
2392 the SQL query defines a special view called RegistryEntry that allows doing a
2393 polymorphic query against all RegistryEntry instances regardless of their actual
2394 concrete type or table name.

2395 The following example is the same as Section 8.3.4 except that it is applied against all
2396 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will
2397 include id for all qualifying RegistryEntry instances whose name contains the word
2398 'Acme' and that have a version greater than 1.3.
```
2399  SELECT id FROM RegistryEntry WHERE name LIKE '%Acme%' AND
2400          objectType = 'ExtrinsicObject' AND
2401          majorVersion >= 1 AND
2402          (majorVersion >= 2 OR  minorVersion > 3);
```

2403 ### 8.3.6 Classification Queries

2404 This section describes the various classification related queries that must be supported.

2405 #### 8.3.6.1 Identifying ClassificationNodes

2406 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they
2407 may also be identified as a path attribute that specifies an XPATH expression [XPT]
2408 from a root classification node to the specified classification node in the XML document
2409 that would represent the ClassificationNode tree including the said ClassificationNode.

2410 #### 8.3.6.2 Getting Root Classification Nodes

2411 To get the collection of root ClassificationNodes the following query predicate must be
2412 supported:
```
2413  SELECT cn.id FROM ClassificationNode cn WHERE parent IS NULL
```

2414 The above query returns all ClassificationNodes that have their parent attribute set to
2415 null. Note that the above query may also specify a predicate on the name if a specific
2416 root ClassificationNode is desired.

2417 **8.3.6.3   Getting Children of Specified ClassificationNode**

2418 To get the children of a ClassificationNode given the ID of that node the following style
2419 of query must be supported:
2420
```
SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>
```

2421 The above query returns all ClassificationNodes that have the node specified by <id> as
2422 their parent attribute.

2423 **8.3.6.4   Getting Objects Classified By a ClassificationNode**

2424 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the
2425 following style of query must be supported:

2426
```
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'))
```

2438 The above query gets the collection of ExtrinsicObjects that are classified by the
2439 Automotive Industry and the Japan Geography. Note that according to the semantics
2440 defined for GetClassifiedObjectsRequest, the query will also contain any objects that
2441 are classified by descendents of the specified ClassificationNodes.

2442 **8.3.6.5   Getting ClassificationNodes That Classify an Object**

2443 To get the collection of ClassificationNodes that classify a specified Object the following
2444 style of query must be supported:
2445
```
SELECT id FROM ClassificationNode
        WHERE id IN (RegistryEntry_classificationNodes(<id>))
```

2447 ## 8.3.7 Association Queries

2448 This section describes the various Association related queries that must be supported.

2449 **8.3.7.1   Getting All Association With Specified Object As Its Source**

2450 To get the collection of Associations that have the specified Object as its source, the
2451 following query must be supported:
2452
```
SELECT id FROM Association WHERE sourceObject = <id>
```

2453   **8.3.7.2   Getting All Association With Specified Object As Its Target**

2454   To get the collection of Associations that have the specified Object as its target, the
2455   following query must be supported:
2456   ```
SELECT id FROM Association WHERE targetObject = <id>
```

2457    **8.3.7.3    Getting Associated Objects Based On Association Attributes**

2458    To get the collection of Associations that have specified Association attributes, the
2459    following queries must be supported:

2460    Select Associations that have the specified name.
2461    ```
SELECT id FROM Association WHERE name = <name>
```

2462   Select Associations that have the specified source role name.
2463   ```
SELECT id FROM Association WHERE sourceRole = <roleName>
```

2464    Select Associations that have the specified target role name.
2465    ```
SELECT id FROM Association WHERE targetRole = <roleName>
```

2466   Select Associations that have the specified association type, where association type is a
2467   string containing the corresponding field name described in [ebRIM].
2468   ```
SELECT id FROM Association WHERE
2469   associationType = <associationType>
```

2470    **8.3.7.4    Complex Association Queries**

2471    The various forms of Association queries may be combined into complex predicates.
2472    The following query selects Associations from an object with a specified id, that have
2473    the sourceRole "buysFrom" and targetRole "sellsTo":
2474    ```
SELECT id FROM Association WHERE
2475        sourceObject = <id> AND
2476        sourceRole = 'buysFrom' AND
2477        targetRole = 'sellsTo'
```

2478   ### 8.3.8 Package Queries

2479   To find all Packages that a specified ExtrinsicObject belongs to, the following query is
2480   specified:
2481   ```
SELECT id FROM Package WHERE id IN (RegistryEntry_packages(<id>))
```

2482    **8.3.8.1    Complex Package Queries**

2483    The following query gets all Packages that a specified object belongs to, that are not
2484    deprecated and where name contains "RosettaNet."
2485    ```
SELECT id FROM Package WHERE
2486        id IN (RegistryEntry_packages(<id>)) AND
2487        name LIKE '%RosettaNet%' AND
2488        status <> 'Deprecated'
```

2489   ### 8.3.9 ExternalLink Queries

2490   To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query
2491   is specified:
2492   ```
SELECT id From ExternalLink WHERE id IN (RegistryEntry_externalLinks(<id>))
```

2493  To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following
2494  query is specified:
2495  `SELECT id From ExtrinsicObject WHERE id IN (RegistryEntry_linkedObjects(<id>))`

#### 8.3.9.1    Complex ExternalLink Queries

2497  The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to,
2498  that contain the word 'legal' in their description and have a URL for their externalURI.
2499  `SELECT id FROM ExternalLink WHERE`
2500  `       id IN (RegistryEntry_externalLinks(<id>)) AND`
2501  `       description LIKE '%legal%' AND`
2502  `       externalURI LIKE '%http://%'`

### 8.3.10 Audit Trail Queries

2504  To get the complete collection of AuditableEvent objects for a specified ManagedObject,
2505  the following query is specified:

2506  SELECT id FROM AuditableEvent WHERE  registryEntry = <id>

## 8.4  Ad Hoc Query Request/Response

2508  A client submits an ad hoc query to the ObjectQueryManager by sending an
2509  AdhocQueryRequest. The AdhocQueryRequest contains a sub-element that defines a
2510  query in one of the supported Registry query mechanisms.

2511  The ObjectQueryManager sends an AdhocQueryResponse either synchronously or
2512  asynchronously back to the client. The AdhocQueryResponse returns a collection of
2513  objects whose element type is in the set of element types represented by the leaf nodes
2514  of the RegistryEntry hierarchy in [ebRIM].

2515



2517  **Figure 21: Submit Ad Hoc Query Sequence Diagram**

2518 For details on the schema for the business documents shown in this process refer to
2519 Appendix A.

## 2520  8.5  Content Retrieval

2521 A client retrieves content via the Registry by sending the GetContentRequest to the
2522 ObjectQueryManager. The GetContentRequest specifies a list of Object references for
2523 Objects that need to be retrieved. The ObjectQueryManager returns the specified
2524 content by sending a GetContentResponse message to the ObjectQueryManagerClient
2525 interface of the client. If there are no errors encountered, the GetContentResponse
2526 message includes the specified content as additional payloads within the message. In
2527 addition to the GetContentResponse payload, there is one additional payload for each
2528 content that was requested. If there are errors encountered, the RegistryResponse
2529 payload includes an error and there are no additional content specific payloads.

### 2530  8.5.1  Identification Of Content Payloads

2531 Since the GetContentResponse message may include several repository items as
2532 additional payloads, it is necessary to have a way to identify each payload in the
2533 message. To facilitate this identification, the Registry must do the following:

2534 • Use the ID for each RegistryEntry instance that describes the repository item as
2535    the DocumentLabel element in the DocumentReference for that object in the
2536    Manifest element of the ebXMLHeader.

### 2537  8.5.2  GetContentResponse Message Structure

2538 The following message fragment illustrates the structure of the GetContentResponse
2539 Message that is returning a Collection of CPPs as a result of a GetContentRequest that
2540 specified the IDs for the requested objects. Note that the ID for each object retrieved in
2541 the message as additional payloads is used as its DocumentLabel in the Manifest of the
2542 ebXMLHeader.

2543
2544
2545 …
2546
2547 --PartBoundary
2548 …
2549 <eb:MessageHeader SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2550 …
2551   <eb:Service eb:type="ebXMLRegistry">ObjectManager</eb:Service>
2552   <eb:Action>submitObjects</eb:Action>
2553 …
2554 </eb:MessageHeader>
2555 …
2556 <eb:Manifest SOAP-ENV:mustUnderstand="1" eb:version="1.0">
2557   <eb:Reference xlink:href="cid:registryentries@example.com" …>
2558     <eb:Description xml:lang="en-us">XML instances that are parameters         for the particular
2559 Registry Interface / Method. These are RIM structures that don't include repository items, just a
2560 reference – contentURI to them.</eb:Description>
2561   </eb:Reference>
2562   <eb:Reference xlink:href="cid:cpp1@example.com" …>

```
2563        <eb:Description xml:lang="en-us">XML instance of CPP 1. This is a repository
2564    item.</eb:Description>
2565      </eb:Reference>
2566      <eb:Reference xlink:href="cid:cpp2@example.com" …>
2567        <eb:Description xml:lang="en-us">XML instance of CPP 2. This is a repository
2568    item.</eb:Description>
2569      </eb:Reference>
2570    </eb:Manifest>
2571
2572    --PartBoundary
2573    Content-ID: registryentries@example.com
2574    Content-Type: text/xml
2575    …
2576    <?xml version="1.0" encoding="UTF-8"?>
2577    <RootElement>
2578    <SubmitObjectsRequest>
2579      <RegistryEntryList>
2580        <ExtrinsicObject … contentURI="cid:cpp1@example.com" …/>
2581        <ExtrinsicObject … contentURI="cid:cpp2@example.com" …/>
2582      </RegistryEntryList>
2583    </SubmitObjectsRequest>
2584    </RootElement>
2585    --PartBoundary
2586    Content-ID: cpp1@example.com
2587    Content-Type: text/xml
2588    …
2589    <CPP>
2590    …
2591    </CPP>
2592
2593    --PartBoundary
2594    Content-ID: cpp2@example.com
2595    Content-Type: text/xml
2596    …
2597    <CPP>
2598    …
2599    </CPP>
2600
2601    --PartBoundary--
2602
2603
```

8.6 Query And Retrieval: Typical Sequence

2605 The following diagram illustrates the use of both browse/drilldown and ad hoc queries
2606 followed by a retrieval of content that was selected by the queries.

Figure 23: Typical Query and Retrieval Sequence

9   Registry Security

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

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

9.1 Integrity of Registry Content

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

9.1.1 Message Payload Signature

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

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

- The content's veracity can be ascertained by its association with a specific submitting organization

9.2 Authentication

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

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

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

9.2.1 Message Header Signature

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

9.3 Confidentiality

9.3.1 On-the-wire Message Confidentiality

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

9.3.2 Confidentiality of Registry Content

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

9.4 Authorization

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

9.4.1 Pre-defined Roles For Registry Users

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

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

9.4.2 Default Access Control Policies

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

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

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

2675

2676   The following list summarizes the default role-based AccessControlPolicy:

2677   • The Registry must implement the default AccessControlPolicy and associate it
2678   with all Objects in the Registry

2679   • Anyone can publish content, but needs to be authenticated

2680   • Anyone can access the content without requiring authentication

2681   • The ContentOwner has access to all methods for Registry Objects owned by
2682   them

2683   • The RegistryAdministrator has access to all methods on all Registry Objects

2684   • Unauthenticated clients can access all read-only (getXXX) methods

2685   • At the time of content submission, the Registry must assign the default
2686   ContentOwner role to the Submitting Organization (SO) as authenticated by the
2687   credentials in the submission message. In the current version of this
2688   specification, it will be the DN as identified by the certificate

2689   • Clients that browse the Registry need not use certificates. The Registry must
2690   assign the default RegistryGuest role to such clients.

Appendix A   ebXML Registry DTD Definition

2692   The following is the definition for the various ebXML Message payloads described in
2693   this document.

```
2694
2695    <?xml version="1.0" encoding="UTF-8"?>
2696    <!-- Begin information model mapping. -->
2697
2698    <!--
2699    ObjectAttributes are attributes from the RegistryObject interface in ebRIM.
2700
2701    id may be empty. If specified it may be in urn:uuid format or be in some
2702    arbitrary format. If id is empty registry must generate globally unique id.
```

```
2703
2704    If id is provided and in proper UUID syntax (starts with urn:uuid:)
2705    registry will honour it.
2706
2707    If id is provided and is not in proper UUID syntax then it is used for
2708    linkage within document and is ignored by the registry. In this case the
2709    registry generates a UUID for id attribute.
2710
2711    id must not be null when object is being retrieved from the registry.
2712    -->
2713    <!ENTITY % ObjectAttributes "
2714       id            ID      #IMPLIED
2715       name          CDATA   #IMPLIED
2716       description CDATA     #IMPLIED
2717    ">
2718
2719    <!--
2720    Use as a proxy for an Object that is in the registry already.
2721    Specifies the id attribute of the object in the registry as its id attribute.
2722    id attribute in ObjectAttributes is exactly the same syntax and semantics as
2723    id attribute in RegistryObject.
2724    -->
2725    <!ELEMENT ObjectRef EMPTY>
2726    <!ATTLIST ObjectRef
2727        id ID #IMPLIED
2728    >
2729
2730    <!ELEMENT ObjectRefList (ObjectRef)*>
2731
2732    <!--
2733    RegistryEntryAttributes are attributes from the RegistryEntry interface
2734    in ebRIM.
2735    It inherits ObjectAttributes
2736    -->
2737    <!ENTITY % RegistryEntryAttributes " %ObjectAttributes;
2738       majorVersion      CDATA   '1'
2739       minorVersion      CDATA   '0'
2740       status            CDATA   #IMPLIED
2741       userVersion       CDATA   #IMPLIED
2742       stability         CDATA   'Dynamic'
2743       expirationDate    CDATA   #IMPLIED">
2744
2745    <!ELEMENT RegistryEntry (SlotList?)>
2746    <!ATTLIST RegistryEntry
2747           %RegistryEntryAttributes; >
2748    <!ELEMENT Value (#PCDATA)>
2749    <!ELEMENT ValueList (Value*)>
2750    <!ELEMENT Slot (ValueList?)>
2751    <!ATTLIST Slot
2752           name CDATA #REQUIRED
2753           slotType CDATA #IMPLIED
2754    >
2755    <!ELEMENT SlotList (Slot*)>
2756
2757    <!--
2758    ExtrinsicObject are attributes from the ExtrinsicObject interface in ebRIM.
```

```
2759    It inherits RegistryEntryAttributes
2760    -->
2761
2762
2763    <!ELEMENT ExtrinsicObject EMPTY >
2764    <!ATTLIST ExtrinsicObject
2765            %RegistryEntryAttributes;
2766            contentURI CDATA #REQUIRED
2767            mimeType CDATA #IMPLIED
2768            objectType CDATA #REQUIRED
2769            opaque (true | false) "false"
2770    >
2771
2772
2773    <!ENTITY % IntrinsicObjectAttributes " %RegistryEntryAttributes;">
2774
2775    <!-- Leaf classes that reflect the concrete classes in ebRIM -->
2776    <!ELEMENT RegistryEntryList
2777    (Association | Classification | ClassificationNode | Package |
2778     ExternalLink | ExternalIdentifier | Organization |
2779     ExtrinsicObject | ObjectRef)*>
2780
2781    <!--
2782    An ExternalLink specifies a link from a RegistryEntry and an external URI
2783    -->
2784    <!ELEMENT ExternalLink EMPTY>
2785    <!ATTLIST ExternalLink
2786            %IntrinsicObjectAttributes;
2787            externalURI CDATA #IMPLIED
2788    >
2789
2790    <!--
2791    An ExternalIdentifier provides an identifier for a RegistryEntry
2792
2793    The value is the value of the identifier (e.g. the social security number)
2794    -->
2795    <!ELEMENT ExternalIdentifier EMPTY>
2796    <!ATTLIST ExternalIdentifier
2797            %IntrinsicObjectAttributes;
2798            value CDATA #REQUIRED
2799    >
2800
2801    <!--
2802    An Association specifies references to two previously submitted
2803    registry entrys.
2804
2805    The sourceObject is id of the sourceObject in association
2806    The targetObject is id of the targetObject in association
2807    -->
2808    <!ELEMENT Association EMPTY>
2809    <!ATTLIST Association
2810            %IntrinsicObjectAttributes;
2811            sourceRole CDATA #IMPLIED
2812            targetRole CDATA #IMPLIED
2813            associationType CDATA #REQUIRED
2814            bidirection (true | false) "false"
```

```
2815            sourceObject IDREF #REQUIRED
2816            targetObject IDREF #REQUIRED
2817    >
2818
2819    <!--
2820    A Classification specifies references to two registry entrys.
2821
2822    The classifiedObject is id of the Object being classified.
2823    The classificationNode is id of the ClassificationNode classying the object
2824    -->
2825    <!ELEMENT Classification EMPTY>
2826    <!ATTLIST Classification
2827            %IntrinsicObjectAttributes;
2828            classifiedObject IDREF #REQUIRED
2829            classificationNode IDREF #REQUIRED
2830    >
2831
2832    <!--
2833    A Package is a named collection of objects.
2834    -->
2835    <!ELEMENT Package EMPTY>
2836    <!ATTLIST Package
2837            %IntrinsicObjectAttributes;
2838    >
2839
2840    <!-- Attributes inherited by various types of telephone number elements -->
2841    <!ENTITY % TelephoneNumberAttributes " areaCode   CDATA  #REQUIRED
2842      contryCode CDATA  #REQUIRED
2843      extension  CDATA  #IMPLIED
2844      number     CDATA  #REQUIRED
2845      url        CDATA  #IMPLIED">
2846    <!ELEMENT TelephoneNumber EMPTY>
2847    <!ATTLIST TelephoneNumber
2848            %TelephoneNumberAttributes;
2849    >
2850    <!ELEMENT FaxNumber EMPTY>
2851    <!ATTLIST FaxNumber
2852            %TelephoneNumberAttributes;
2853    >
2854
2855    <!ELEMENT PagerNumber EMPTY>
2856    <!ATTLIST PagerNumber
2857            %TelephoneNumberAttributes;
2858    >
2859
2860    <!ELEMENT MobileTelephoneNumber EMPTY>
2861    <!ATTLIST MobileTelephoneNumber
2862            %TelephoneNumberAttributes;
2863    >
2864    <!-- PostalAddress -->
2865    <!ELEMENT PostalAddress EMPTY>
2866    <!ATTLIST PostalAddress
2867            city CDATA #REQUIRED
2868            country CDATA #REQUIRED
2869            postalCode CDATA #REQUIRED
2870            state CDATA #IMPLIED
```

```
2871                 street CDATA #REQUIRED
2872    >
2873    <!-- PersonName -->
2874    <!ELEMENT PersonName EMPTY>
2875    <!ATTLIST PersonName
2876            firstName CDATA #REQUIRED
2877            middleName CDATA #IMPLIED
2878            lastName CDATA #REQUIRED
2879    >
2880
2881    <!-- Organization -->
2882    <!ELEMENT Organization (PostalAddress, FaxNumber?, TelephoneNumber)>
2883    <!ATTLIST Organization
2884            %IntrinsicObjectAttributes;
2885            parent IDREF #IMPLIED
2886            primaryContact IDREF #REQUIRED
2887    >
2888
2889    <!ELEMENT User (PersonName, PostalAddress, TelephoneNumber,
2890                                            MobileTelephoneNumber?,
2891                                            FaxNumber?, PagerNumber?)>
2892    <!ATTLIST User
2893            %ObjectAttributes;
2894            organization IDREF #IMPLIED
2895            email CDATA #IMPLIED
2896            url CDATA #IMPLIED
2897    >
2898
2899    <!ELEMENT AuditableEvent EMPTY>
2900    <!ATTLIST AuditableEvent
2901            %ObjectAttributes;
2902            eventType CDATA #REQUIRED
2903            registryEntry IDREF #REQUIRED
2904            timestamp CDATA #REQUIRED
2905            user IDREF #REQUIRED
2906    >
2907
2908    <!--
2909    ClassificationNode is used to submit a Classification tree to the Registry.
2910
2911    parent is the id to the parent node. code is an optional code value for a
2912                                            ClassificationNode
2913    often defined by an external taxonomy (e.g. NAICS)
2914    -->
2915    <!ELEMENT ClassificationNode EMPTY>
2916    <!ATTLIST ClassificationNode
2917            %IntrinsicObjectAttributes;
2918            parent IDREF #IMPLIED
2919            code CDATA #IMPLIED
2920    >
2921
2922    <!--
2923    End information model mapping.
2924
2925    Begin Registry Services Interface
2926
```

```
2927  <!ELEMENT RequestAcceptedResponse EMPTY>
2928  <!ATTLIST RequestAcceptedResponse
2929         xml:lang NMTOKEN #REQUIRED
2930  >
2931  <!--
2932
2933  The SubmitObjectsRequest allows one to submit a list of RegistryEntry
2934  elements. Each RegistryEntry element provides metadata for a single submitted
2935  object.  Note that the repository item being submitted is in a separate
2936  document that is not in this DTD. The ebXML Messaging Services Specfication
2937  defines packaging, for submission, of the metadata of a repository item with
2938  the repository item itself. The value of the contentURI attribute of the
2939  ExtrinsicObject element must be the same as the xlink:href attribute within
2940  the Reference element within the Manifest element of the MessageHeader.
2941  -->
2942  <!ELEMENT SubmitObjectsRequest (RegistryEntryList)>
2943  <!ELEMENT AddSlotsRequest (ObjectRef, SlotList)+>
2944  <!-- Only need name in Slot within SlotList -->
2945  <!ELEMENT RemoveSlotsRequest (ObjectRef, SlotList)+>
2946  <!--
2947  The ObjectRefList is the list of
2948  refs to the registry entrys being approved.
2949  -->
2950  <!ELEMENT ApproveObjectsRequest (ObjectRefList)>
2951  <!--
2952  The ObjectRefList is the list of
2953  refs to the registry entrys being deprecated.
2954  -->
2955  <!ELEMENT DeprecateObjectsRequest (ObjectRefList)>
2956  <!--
2957  The ObjectRefList is the list of
2958  refs to the registry entrys being removed
2959  -->
2960  <!ELEMENT RemoveObjectsRequest (ObjectRefList)>
2961  <!ATTLIST RemoveObjectsRequest
2962         deletionScope (DeleteAll | DeleteRepositoryItemOnly) "DeleteAll"
2963  >
2964  <!ELEMENT GetRootClassificationNodesRequest EMPTY>
2965  <!--
2966  The namePattern follows SQL-92 syntax for the pattern specified in
2967  LIKE clause. It allows for selecting only those root nodes that match
2968  the namePattern. The default value of '*' matches all root nodes.
2969  -->
2970  <!ATTLIST GetRootClassificationNodesRequest
2971         namePattern CDATA "*"
2972  >
2973  <!--
2974  The response includes one or more ClassificationNodes
2975  -->
2976  <!ELEMENT GetRootClassificationNodesResponse ( ClassificationNode+ )>
2977  <!--
2978  Get the classification tree under the ClassificationNode specified parentRef.
2979
2980  If depth is 1 just fetch immediate child
2981  nodes, otherwise fetch the descendant tree upto the specified depth level.
2982  If depth is 0 that implies fetch entire sub-tree
```

```
2983  -->
2984  <!ELEMENT GetClassificationTreeRequest EMPTY>
2985  <!ATTLIST GetClassificationTreeRequest
2986          parent CDATA #REQUIRED
2987          depth CDATA "1"
2988  >
2989  <!--
2990  The response includes one or more ClassificationNodes which includes only
2991  immediate ClassificationNode children nodes if depth attribute in
2992  GetClassificationTreeRequest was 1, otherwise the decendent nodes
2993  upto specified depth level are returned.
2994  -->
2995  <!ELEMENT GetClassificationTreeResponse ( ClassificationNode+ )>
2996  <!--
2997  Get refs to all registry entrys that are classified by all the
2998  ClassificationNodes specified by ObjectRefList.
2999  Note this is an implicit logical AND operation
3000  -->
3001  <!ELEMENT GetClassifiedObjectsRequest (ObjectRefList)>
3002  <!--
3003  objectType attribute can specify the type of objects that the registry
3004  client is interested in, that is classified by this ClassificationNode.
3005  It is a String that matches a choice in the type attribute of
3006                                                  ExtrinsicObject.
3007  The default value of '*' implies that client is interested in all types
3008  of registry entrys that are classified by the specified ClassificationNode.
3009  -->
3010  <!--
3011  The response includes a RegistryEntryList which has zero or more
3012  RegistryEntrys that are classified by the ClassificationNodes
3013  specified in the ObjectRefList in GetClassifiedObjectsRequest.
3014  -->
3015  <!ELEMENT GetClassifiedObjectsResponse ( RegistryEntryList )>
3016  <!--
3017  An Ad hoc query request specifies a query string as defined by [RS] in the
3018                                                  queryString attribute
3019  -->
3020  <!ELEMENT AdhocQueryRequest (FilterQuery | ReturnRegistryEntry |
3021                                                  ReturnRepositoryItem | SQLQuery)>
3022  <!ELEMENT SQLQuery (#PCDATA)>
3023  <!--
3024  The response includes a RegistryEntryList which has zero or more
3025  RegistryEntrys that match the query specified in AdhocQueryRequest.
3026  -->
3027  <!ELEMENT AdhocQueryResponse
3028   ( RegistryEntryList |
3029     FilterQueryResult |
3030     ReturnRegistryEntryResult |
3031     ReturnRepositoryItemResult )>
3032  <!--
3033  Gets the actual content (not metadata) specified by the ObjectRefList
3034  -->
3035  <!ELEMENT GetContentRequest (ObjectRefList)>
3036  <!--
3037  The GetObjectsResponse will have no sub-elements if there were no errors.
3038  The actual contents will be in the other payloads of the message.
```

ebXML Registry Services Specification   Page 87

```
3039    -->
3040    <!ELEMENT GetContentResponse EMPTY >
3041    <!--
3042    Describes the capability profile for the registry and what optional features
3043    are supported
3044    -->
3045    <!ELEMENT RegistryProfile (OptionalFeaturesSupported)>
3046    <!ATTLIST RegistryProfile
3047           version CDATA #REQUIRED
3048    >
3049
3050    <!ELEMENT OptionalFeaturesSupported EMPTY>
3051    <!ATTLIST OptionalFeaturesSupported
3052           sqlQuery (true | false) "false"
3053           xQuery (true | false) "false"
3054    >
3055    <!-- Begin FilterQuery DTD -->
3056    <!ELEMENT FilterQuery (RegistryEntryQuery | AuditableEventQuery |
3057                                          ClassificationNodeQuery |
3058                                          RegistryPackageQuery |
3059                                          OrganizationQuery)>
3060    <!ELEMENT FilterQueryResult (RegistryEntryQueryResult |
3061                                          AuditableEventQueryResult |
3062                                          ClassificationNodeQueryResult |
3063                                          RegistryPackageQueryResult |
3064                                          OrganizationQueryResult)>
3065    <!ELEMENT RegistryEntryQueryResult (RegistryEntryView*)>
3066    <!ELEMENT RegistryEntryView EMPTY>
3067    <!ATTLIST RegistryEntryView
3068           objectURN CDATA #REQUIRED
3069           contentURI CDATA #IMPLIED
3070           objectID CDATA #IMPLIED
3071    >
3072    <!ELEMENT AuditableEventQueryResult (AuditableEventView*)>
3073    <!ELEMENT AuditableEventView EMPTY>
3074    <!ATTLIST AuditableEventView
3075           objectID CDATA #REQUIRED
3076           timestamp CDATA #REQUIRED
3077    >
3078    <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
3079    <!ELEMENT ClassificationNodeView EMPTY>
3080    <!ATTLIST ClassificationNodeView
3081           objectURN CDATA #REQUIRED
3082           contentURI CDATA #IMPLIED
3083           objectID CDATA #IMPLIED
3084    >
3085    <!ELEMENT RegistryPackageQueryResult (RegistryPackageView*)>
3086    <!ELEMENT RegistryPackageView EMPTY>
3087    <!ATTLIST RegistryPackageView
3088           objectURN CDATA #REQUIRED
3089           contentURI CDATA #IMPLIED
3090           objectID CDATA #IMPLIED
3091    >
3092    <!ELEMENT OrganizationQueryResult (OrganizationView*)>
3093    <!ELEMENT OrganizationView EMPTY>
3094    <!ATTLIST OrganizationView
```

ebXML Registry Services Specification   Page 88

```
3095          orgURN CDATA #REQUIRED
3096          objectID CDATA #IMPLIED
3097  >
3098
3099  <!ELEMENT RegistryEntryQuery
3100     (  RegistryEntryFilter?,
3101        SourceAssociationBranch*,
3102        TargetAssociationBranch*,
3103        HasClassificationBranch*,
3104        SubmittingOrganizationBranch?,
3105        ResponsibleOrganizationBranch?,
3106        ExternalIdentifierFilter*,
3107        ExternalLinkFilter*,
3108        SlotFilter*,
3109        HasAuditableEventBranch*            )>
3110
3111  <!ELEMENT SourceAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3112  <!ELEMENT TargetAssociationBranch (AssociationFilter?, RegistryEntryFilter?)>
3113  <!ELEMENT HasClassificationBranch (ClassificationFilter?,
3114                                              ClassificationNodeFilter?)>
3115  <!ELEMENT SubmittingOrganizationBranch (OrganizationFilter?, ContactFilter?)>
3116  <!ELEMENT ResponsibleOrganizationBranch (OrganizationFilter?,
3117                                              ContactFilter?)>
3118  <!ELEMENT HasAuditableEventBranch (AuditableEventFilter?, UserFilter?,
3119                                              OrganizationFilter?)>
3120  <!ELEMENT AuditableEventQuery
3121    (AuditableEventFilter?, RegistryEntryQuery*, InvokedByBranch? )>
3122
3123  <!ELEMENT InvokedByBranch
3124    ( UserFilter?, OrganizationQuery? )>
3125
3126  <!ELEMENT ClassificationNodeQuery (ClassificationNodeFilter?,
3127                                              PermitsClassificationBranch*,
3128                                              HasParentNode?, HasSubnode*)>
3129  <!ELEMENT PermitsClassificationBranch (ClassificationFilter?,
3130                                              RegistryEntryQuery?)>
3131  <!ELEMENT HasParentNode (ClassificationNodeFilter?, HasParentNode?)>
3132  <!ELEMENT HasSubnode (ClassificationNodeFilter?, HasSubnode*)>
3133  <!ELEMENT RegistryPackageQuery (PackageFilter?, HasMemberBranch*)>
3134  <!ELEMENT HasMemberBranch (RegistryEntryQuery?)>
3135  <!ELEMENT OrganizationQuery (OrganizationFilter?, SubmitsRegistryEntry*,
3136                                              HasParentOrganization?,
3137                                              InvokesEventBranch*,
3138                                              ContactFilter*)>
3139  <!ELEMENT SubmitsRegistryEntry (RegistryEntryQuery?)>
3140  <!ELEMENT HasParentOrganization (OrganizationFilter?,
3141                                              HasParentOrganization?)>
3142  <!ELEMENT InvokesEventBranch (UserFilter?, AuditableEventFilter?,
3143                                              RegistryEntryQuery?)>
3144  <!ELEMENT ReturnRegistryEntry (RegistryEntryQuery, WithClassifications?,
3145                                              WithSourceAssociations?,
3146                                              WithTargetAssociations?,
3147                                              WithAuditableEvents?,
3148                                              WithExternalLinks?)>
3149  <!ELEMENT WithClassifications (ClassificationFilter?)>
3150  <!ELEMENT WithSourceAssociations (AssociationFilter?)>
```

```
3151    <!ELEMENT WithTargetAssociations (AssociationFilter?)>
3152    <!ELEMENT WithAuditableEvents (AuditableEventFilter?)>
3153    <!ELEMENT WithExternalLinks (ExternalLinkFilter?)>
3154    <!ELEMENT ReturnRegistryEntryResult (RegistryEntryMetadata*)>
3155    <!ELEMENT RegistryEntryMetadata (RegistryEntry, Classification*,
3156                                           SourceAssociations?,
3157                                           TargetAssociations?,
3158                                           AuditableEvent*, ExternalLink*)>
3159    <!ELEMENT SourceAssociations (Association*)>
3160    <!ELEMENT TargetAssociations (Association*)>
3161    <!ELEMENT ReturnRepositoryItem (RegistryEntryQuery,
3162                                           RecursiveAssociationOption?,
3163                                           WithDescription?)>
3164    <!ELEMENT RecursiveAssociationOption (AssociationType+)>
3165    <!ATTLIST RecursiveAssociationOption
3166          depthLimit CDATA #IMPLIED
3167    >
3168    <!ELEMENT AssociationType EMPTY>
3169    <!ATTLIST AssociationType
3170          role CDATA #REQUIRED
3171    >
3172    <!ELEMENT WithDescription EMPTY>
3173    <!ELEMENT ReturnRepositoryItemResult (RepositoryItem*)>
3174    <!ELEMENT RepositoryItem (RegistryPackage | ExtrinsicObject | WithdrawnObject
3175                                           | ExternalLink)>
3176    <!ATTLIST RepositoryItem
3177          identifier CDATA #REQUIRED
3178          name CDATA #REQUIRED
3179          contentURI CDATA #REQUIRED
3180          objectType CDATA #REQUIRED
3181          status CDATA #REQUIRED
3182          stability CDATA #REQUIRED
3183          description CDATA #IMPLIED
3184    >
3185    <!ELEMENT RegistryPackage EMPTY>
3186    <!ELEMENT WithdrawnObject EMPTY>
3187    <!ELEMENT ExternalLinkItem EMPTY>
3188    <!ELEMENT ObjectFilter (Clause)>
3189    <!ELEMENT RegistryEntryFilter (Clause)>
3190    <!ELEMENT IntrinsicObjectFilter (Clause)>
3191    <!ELEMENT ExtrinsicObjectFilter (Clause)>
3192    <!ELEMENT PackageFilter (Clause)>
3193    <!ELEMENT OrganizationFilter (Clause)>
3194    <!ELEMENT ContactFilter (Clause)>
3195    <!ELEMENT ClassificationNodeFilter (Clause)>
3196    <!ELEMENT AssociationFilter (Clause)>
3197    <!ELEMENT ClassificationFilter (Clause)>
3198    <!ELEMENT ExternalLinkFilter (Clause)>
3199    <!ELEMENT SlotFilter (Clause)>
3200    <!ELEMENT ExternalIdentifierFilter (Clause)>
3201    <!ELEMENT AuditableEventFilter (Clause)>
3202    <!ELEMENT UserFilter (Clause)>
3203
3204    <!--
3205    The following lines define the XML syntax for Clause.
3206    -->
```

```
3207    <!ELEMENT Clause (SimpleClause | CompoundClause)>
3208    <!ELEMENT SimpleClause (BooleanClause | RationalClause | StringClause)>
3209    <!ATTLIST SimpleClause
3210          leftArgument CDATA #REQUIRED
3211    >
3212    <!ELEMENT CompoundClause (Clause, Clause+)>
3213    <!ATTLIST CompoundClause
3214          connectivePredicate (And | Or) #REQUIRED
3215    >
3216    <!ELEMENT BooleanClause EMPTY>
3217    <!ATTLIST BooleanClause
3218          booleanPredicate (true | false) #REQUIRED
3219    >
3220    <!ELEMENT RationalClause (IntClause | FloatClause)>
3221    <!ATTLIST RationalClause
3222          logicalPredicate (LE | LT | GE | GT | EQ | NE) #REQUIRED
3223    >
3224    <!ELEMENT IntClause (#PCDATA)>
3225    <!ATTLIST IntClause
3226          e-dtype NMTOKEN #FIXED "int"
3227    >
3228    <!ELEMENT FloatClause (#PCDATA)>
3229    <!ATTLIST FloatClause
3230          e-dtype NMTOKEN #FIXED "float"
3231    >
3232    <!ELEMENT StringClause (#PCDATA)>
3233    <!ATTLIST StringClause
3234          stringPredicate
3235        (contains | -contains |
3236           startswith | -startswith |
3237           equal | -equal |
3238           endswith | -endswith) #REQUIRED
3239    >
3240    <!-- End FilterQuery DTD -->
3241
3242    <!-- Begin RegistryError definition -->
3243    <!-- The RegistryErrorList is derived from the ErrorList element from the
3244     ebXML Message Service Specification -->
3245    <!ELEMENT RegistryErrorList ( RegistryError+ )>
3246    <!ATTLIST RegistryErrorList
3247      highestSeverity  ( Warning | Error ) 'Warning' >
3248
3249    <!ELEMENT RegistryError  (#PCDATA) >
3250    <!ATTLIST RegistryError
3251      codeContext   CDATA   #REQUIRED
3252      errorCode     CDATA   #REQUIRED
3253      severity  ( Warning | Error ) 'Warning'
3254      location      CDATA   #IMPLIED
3255      xml:lang      NMTOKEN  #IMPLIED>
3256
3257    <!ELEMENT RegistryResponse
3258     (( AdhocQueryResponse     |
3259        GetContentResponse     |
3260        GetClassificationTreeResponse  |
3261        GetClassifiedObjectsResponse   |
3262        GetRootClassificationNodesResponse  )?,
```

```
3263        RegistryErrorList?  )>
3264  <!ATTLIST RegistryResponse
3265     status  (success | failure )  #REQUIRED >
3266
3267  <!-- The contrived root node -->
3268
3269  <!ELEMENT RootElement
3270     ( SubmitObjectsRequest |
3271       ApproveObjectsRequest |
3272       DeprecateObjectsRequest |
3273       RemoveObjectsRequest |
3274       GetRootClassificationNodesRequest |
3275       GetClassificationTreeRequest |
3276       GetClassifiedObjectsRequest |
3277       AdhocQueryRequest |
3278       GetContentRequest |
3279       AddSlotsRequest |
3280       RemoveSlotsRequest |
3281       RegistryResponse |
3282       RegistryProfile) >
3283
3284  <!ELEMENT Href  (#PCDATA )>
3285
3286  <!ELEMENT XMLDocumentErrorLocn  (DocumentId , Xpath )>
3287
3288  <!ELEMENT DocumentId  (#PCDATA )>
3289
3290  <!ELEMENT Xpath (#PCDATA)>
```

3291 Appendix B
3292   Interpretation of UML Diagrams

3293 This section describes in *abstract terms* the conventions used to define ebXML
3294 business process description in UML.

3295 B.1 UML Class Diagram

3296 A UML class diagram is used to describe the Service Interfaces (as defined by [ebCPP])
3297 required to implement an ebXML Registry Services and clients. See Figure 2 on page
3298 14 for an example. The UML class diagram contains:

3299

3300   1. A collection of UML interfaces where each interface represents a Service
3301   Interface for a Registry service.

3302   2. Tabular description of methods on each interface where each method represents
3303   an Action (as defined by [ebCPP]) within the Service Interface representing the
3304   UML interface.

3305   3. Each method within a UML interface specifies one or more parameters, where
3306   the type of each method argument represents the ebXML message type that is
3307   exchanged as part of the Action corresponding to the method. Multiple
3308   arguments imply multiple payload documents within the body of the
3309   corresponding ebXML message.

B.2 UML Sequence Diagram

3311 A UML sequence diagram is used to specify the business protocol representing the
3312 interactions between the UML interfaces for a Registry specific ebXML business
3313 process. A UML sequence diagram provides the necessary information to determine the
3314 sequencing of messages, request to response association as well as request to error
3315 response association as described by [ebCPP].

3316 Each sequence diagram shows the sequence for a specific conversation protocol as
3317 method calls from the requestor to the responder. Method invocation may be
3318 synchronous or asynchronous based on the UML notation used on the arrow-head for
3319 the link. A half arrow-head represents asynchronous communication. A full arrow-head
3320 represents synchronous communication.

3321 Each method invocation may be followed by a response method invocation from the
3322 responder to the requestor to indicate the ResponseName for the previous Request.
3323 Possible error response is indicated by a conditional response method invocation from
3324 the responder to the requestor. See on page 20 for an example.

Appendix C   SQL Query

C.1 SQL Query Syntax Specification

3327   This section specifies the rules that define the SQL Query syntax as a subset of
3328   SQL-92. The terms enclosed in angle brackets are defined in [SQL] or in
3329   [SQL/PSM]. The SQL query syntax conforms to the <query specification>, modulo
3330   the restrictions identified below:

3331   1. A <select list> may contain at most one <select sublist>.

3332   2. In a <select list> must be is a single column whose data type is UUID, from the
3333   table in the <from clause>.

3334   3. A <derived column> may not have an <as clause>.

3335   4. <table expression> does not contain the optional <group by clause> and <having
3336   clause> clauses.

3337   5. A <table reference> can only consist of <table name> and <correlation name>.

3338   6. A <table reference> does not have the optional AS between <table name> and
3339   <correlation name>.

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

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

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

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

C.2 Non-Normative BNF for Query Syntax Grammar

The following BNF exemplifies the grammar for the registry query syntax. It is provided here as an aid to implementors. Since this BNF is not based on [SQL] it is provided as non-normative syntax. For the normative syntax rules see Appendix C.1.

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

RegistryQuery = SQLSelect [";"]

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

SQLSelectCols =  ID

SQLTableList = SQLTableRef

SQLTableRef = ID

SQLWhere = "WHERE" SQLOrExpr

SQLOrExpr = SQLAndExpr ( "OR" SQLAndExpr)*

SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)*

SQLNotExpr = [ "NOT" ] SQLCompareExpr

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


SQLCompareExprRight =
    SQLLikeClause
  | SQLInClause
  | SQLCompareOp SQLSumExpr

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

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

```
3396    SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
3397
3398    SQLLValueElement = "NULL" | SQLSelect
3399
3400    SQLIsClause =  SQLColRef "IS" [ "NOT" ] "NULL"
3401
3402    SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
3403
3404    SQLPattern = STRING_LITERAL
3405
3406    SQLLiteral =
3407        STRING_LITERAL
3408      | INTEGER_LITERAL
3409      | FLOATING_POINT_LITERAL
3410
3411    SQLColRef = SQLLvalue
3412
3413    SQLLvalue =  SQLLvalueTerm
3414
3415    SQLLvalueTerm = ID ( "."  ID )*
3416
3417    SQLSumExpr = SQLProductExpr (( "+" | "-" ) SQLProductExpr )*
3418
3419    SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*
3420
3421    SQLUnaryExpr = [ ( "+" | "-") ] SQLTerm
3422
3423    SQLTerm = "(" SQLOrExpr ")"
3424      | SQLColRef
3425      | SQLLiteral
3426
3427    INTEGER_LITERAL = (["0"-"9"])+
3428
3429    FLOATING_POINT_LITERAL =
3430            (["0"-"9"])+ "." (["0"-"9"])+ (EXPONENT)?
3431          | "." (["0"-"9"])+ (EXPONENT)?
3432          | (["0"-"9"])+ EXPONENT
3433          | (["0"-"9"])+ (EXPONENT)?
3434
3435    EXPONENT = ["e","E"] (["+","-"])? (["0"-"9"])+
3436
3437    STRING_LITERAL: "'" (~["'"])* ( "''" (~["'"])* )* "'"
3438
3439    ID = ( <LETTER> )+ ( "_" | "$" | "#" | <DIGIT> | <LETTER> )*
3440    LETTER = ["A"-"Z", "a"-"z"]
3441    DIGIT = ["0"-"9"]
```

C.3 Relational Schema For SQL Queries

```
3444    --SQL Load file for creating the ebXML Registry tables
3445
3446
3447    --Minimal use of SQL-99 features in DDL is illustrative and may be easily mapped to SQL-92
3448
3449
3450    CREATE TYPE ShortName AS VARCHAR(64) NOT FINAL;
3451    CREATE TYPE LongName AS VARCHAR(128) NOT FINAL;
3452    CREATE TYPE FreeFormText AS VARCHAR(256) NOT FINAL;
3453
3454    CREATE TYPE UUID UNDER ShortName FINAL;
3455    CREATE TYPE URI UNDER LongName FINAL;
3456
3457    CREATE TABLE ExtrinsicObject (
3458
3459    --RegistryObject Attributes
3460      id                                                UUID PRIMARY KEY NOT NULL,
3461      name                                              LongName,
```

```
3462      description                                   FreeFormText,
3463      accessControlPolicy            UUID NOT NULL,
3464
3465    --Versionable attributes
3466      majorVersion                              INT DEFAULT 0 NOT NULL,
3467      minorVersion                              INT DEFAULT 1 NOT NULL,
3468
3469    --RegistryEntry attributes
3470      status                                          INT DEFAULT 0 NOT NULL,
3471      userVersion                          ShortName,
3472      stability                                   INT    DEFAULT 0 NOT NULL,
3473      expirationDate               TIMESTAMP,
3474
3475    --ExtrinsicObject attributes
3476      contentURI                          URI,
3477      mimeType                                  ShortName,
3478      objectType                          INT DEFAULT 0 NOT NULL,
3479      opaque                                    BOOLEAN DEFAULT false NOT NULL
3480
3481    );
3482
3483    CREATE PROCEDURE RegistryEntry_associatedObjects(registryEntryId) {
3484    --Must return a collection of UUIDs for related RegistryEntry instances
3485    }
3486
3487    CREATE PROCEDURE RegistryEntry_auditTrail(registryEntryId) {
3488    --Must return an collection of UUIDs for AuditableEvents related to the RegistryEntry.
3489    --Collection must be in ascending order by timestamp
3490    }
3491
3492    CREATE PROCEDURE RegistryEntry_externalLinks(registryEntryId) {
3493    --Must return a collection of UUIDs for ExternalLinks annotating this RegistryEntry.
3494    }
3495
3496    CREATE PROCEDURE RegistryEntry_externalIdentifiers(registryEntryId) {
3497    --Must return a collection of UUIDs for ExternalIdentifiers for this RegistryEntry.
3498    }
3499
3500    CREATE PROCEDURE RegistryEntry_classificationNodes(registryEntryId) {
3501    --Must return a collection of UUIDs for ClassificationNodes classifying this RegistryEntry.
3502    }
3503
3504    CREATE PROCEDURE RegistryEntry_packages(registryEntryId) {
3505    --Must return a collection of UUIDs for Packages that this RegistryEntry belongs to.
3506    }
3507
3508    CREATE TABLE Package (
3509
3510    --RegistryObject Attributes
3511      id                                        UUID PRIMARY KEY NOT NULL,
3512      name                                      LongName,
3513      description                          FreeFormText,
3514      accessControlPolicy            UUID NOT NULL,
3515
3516    --Versionable attributes
3517      majorVersion                              INT DEFAULT 0 NOT NULL,
3518      minorVersion                              INT DEFAULT 1 NOT NULL,
3519
3520    --RegistryEntry attributes
3521      status                                          INT DEFAULT 0 NOT NULL,
3522      userVersion                          ShortName,
3523      stability                                   INT    DEFAULT 0 NOT NULL,
3524      expirationDate               TIMESTAMP,
3525
3526    --Package attributes
3527    );
3528
3529    CREATE PROCEDURE Package_memberbjects(packageId) {
3530    --Must return a collection of UUIDs for RegistryEntrys that are memebers of this Package.
3531    }
```

```
3532
3533    CREATE TABLE ExternalLink (
3534
3535    --RegistryObject Attributes
3536      id                                              UUID PRIMARY KEY NOT NULL,
3537      name                                            LongName,
3538      description                            FreeFormText,
3539      accessControlPolicy            UUID NOT NULL,
3540
3541    --Versionable attributes
3542      majorVersion                                INT DEFAULT 0 NOT NULL,
3543      minorVersion                                INT DEFAULT 1 NOT NULL,
3544
3545    --RegistryEntry attributes
3546      status                                          INT DEFAULT 0 NOT NULL,
3547      userVersion                            ShortName,
3548      stability                                     INT    DEFAULT 0 NOT NULL,
3549      expirationDate                    TIMESTAMP,
3550
3551    --ExternalLink attributes
3552      externalURI                            URI NOT NULL
3553    );
3554
3555    CREATE PROCEDURE ExternalLink_linkedObjects(registryEntryId) {
3556    --Must return a collection of UUIDs for objects in this relationship
3557    }
3558
3559    CREATE TABLE ExternalIdentifier (
3560
3561    --RegistryObject Attributes
3562      id                                              UUID PRIMARY KEY NOT NULL,
3563      name                                            LongName,
3564      description                            FreeFormText,
3565      accessControlPolicy            UUID NOT NULL,
3566
3567    --Versionable attributes
3568      majorVersion                                INT DEFAULT 0 NOT NULL,
3569      minorVersion                                INT DEFAULT 1 NOT NULL,
3570
3571    --RegistryEntry attributes
3572      status                                          INT DEFAULT 0 NOT NULL,
3573      userVersion                    ShortName,
3574      stability                                     INT    DEFAULT 0 NOT NULL,
3575      expirationDate                    TIMESTAMP,
3576
3577    --ExternalIdentifier attributes
3578      value                                           ShortName NOT NULL
3579    );
3580
3581
3582    --A SlotValue row represents one value of one slot in some
3583    --RegistryEntry
3584    CREATE TABLE SlotValue (
3585
3586    --RegistryObject Attributes
3587      registryEntry                    UUID    PRIMARY KEY NOT NULL,
3588
3589    --Slot attributes
3590      name                                            LongName NOT NULL PRIMARY KEY NOT NULL,
3591      value                                           ShortName NOT NULL
3592    );
3593
3594    CREATE TABLE Association (
3595    --RegistryObject Attributes
3596      id                                              UUID PRIMARY KEY NOT NULL,
3597      name                                            LongName,
3598      description                            FreeFormText,
3599      accessControlPolicy            UUID NOT NULL,
3600
3601    --Versionable attributes
```

ebXML Registry Services Specification   Page 97

```
3602      majorVersion                                       INT DEFAULT 0 NOT NULL,
3603      minorVersion                                       INT DEFAULT 1 NOT NULL,
3604
3605    --RegistryEntry attributes
3606      status                                             INT DEFAULT 0 NOT NULL,
3607      userVersion                            ShortName,
3608      stability                                  INT    DEFAULT 0 NOT NULL,
3609      expirationDate                 TIMESTAMP,
3610
3611    --Association attributes
3612      associationType                INT NOT NULL,
3613      bidirectional                          BOOLEAN DEFAULT false NOT NULL,
3614      sourceObject                           UUID NOT NULL,
3615      sourceRole                             ShortName,
3616      label                                      ShortName,
3617      targetObject                           UUID NOT NULL,
3618      targetRole                             ShortName
3619    );
3620
3621    --Classification is currently identical to Association
3622    CREATE TABLE Classification (
3623    --RegistryObject Attributes
3624      id                                           UUID PRIMARY KEY NOT NULL,
3625      name                                         LongName,
3626      description                        FreeFormText,
3627      accessControlPolicy        UUID NOT NULL,
3628
3629    --Versionable attributes
3630      majorVersion                               INT DEFAULT 0 NOT NULL,
3631      minorVersion                               INT DEFAULT 1 NOT NULL,
3632
3633    --RegistryEntry attributes
3634      status                                       INT DEFAULT 0 NOT NULL,
3635      userVersion                        ShortName,
3636      stability                                  INT    DEFAULT 0 NOT NULL,
3637      expirationDate                 TIMESTAMP,
3638
3639    --Classification attributes. Assumes not derived from Association
3640      sourceObject                           UUID NOT NULL,
3641      targetObject                           UUID NOT NULL,
3642    );
3643
3644
3645    CREATE TABLE ClassificationNode (
3646    --RegistryObject Attributes
3647      id                                           UUID PRIMARY KEY NOT NULL,
3648      name                                         LongName,
3649      description                        FreeFormText,
3650      accessControlPolicy        UUID NOT NULL,
3651
3652    --Versionable attributes
3653      majorVersion                               INT DEFAULT 0 NOT NULL,
3654      minorVersion                               INT DEFAULT 1 NOT NULL,
3655
3656    --RegistryEntry attributes
3657      status                                       INT DEFAULT 0 NOT NULL,
3658      userVersion                        ShortName,
3659      stability                                  INT    DEFAULT 0 NOT NULL,
3660      expirationDate                 TIMESTAMP,
3661
3662    --ClassificationNode attributes
3663      parent                                     UUID,
3664      path                                       VARCHAR(512) NOT NULL,
3665      code                                       ShortName
3666    );
3667
3668    CREATE PROCEDURE ClassificationNode_classifiedObjects(classificationNodeId) {
3669    --Must return a collection of UUIDs for RegistryEntries classified by this ClassificationNode
3670    }
3671
```

```
3672   --Begin Registry Audit Trail tables
3673
3674   CREATE TABLE AuditableEvent (
3675   --RegistryObject Attributes
3676     id                                            UUID PRIMARY KEY NOT NULL,
3677     name                                          LongName,
3678     description                       FreeFormText,
3679     accessControlPolicy           UUID NOT NULL,
3680
3681   --AuditableEvent attributes
3682     user                              UUID,
3683     eventType                                     INT DEFAULT 0 NOT NULL,
3684     registryEntry                         UUID NOT NULL,
3685     timestamp                                     TIMESTAMP NOT NULL,
3686   );
3687
3688
3689
3690   CREATE TABLE User (
3691   --RegistryObject Attributes
3692     id                                            UUID PRIMARY KEY NOT NULL,
3693     name                                          LongName,
3694     description                       FreeFormText,
3695     accessControlPolicy           UUID NOT NULL,
3696
3697   --User attributes
3698     organization                      UUID NOT NULL
3699
3700   --address attributes flattened
3701     address_city                      ShortName,
3702     address_country                 ShortName,
3703     address_postalCode              ShortName,
3704     address_state                     ShortName,
3705     address_street                  ShortName,
3706
3707     email                                 ShortName,
3708
3709   --fax attribute flattened
3710     fax_areaCode                      VARCHAR(4) NOT NULL,
3711     fax_countryCode                 VARCHAR(4),
3712     fax_extension                   VARCHAR(8),
3713     fax_umber                             VARCHAR(8) NOT NULL,
3714     fax_url                           URI
3715
3716    --mobilePhone attribute flattened
3717     mobilePhone_areaCode            VARCHAR(4) NOT NULL,
3718     mobilePhone_countryCode         VARCHAR(4),
3719     mobilePhone_extension           VARCHAR(8),
3720     mobilePhone_umber                     VARCHAR(8) NOT NULL,
3721     mobilePhone_url                   URI
3722
3723   --name attribute flattened
3724     name_firstName                  ShortName,
3725     name_middleName                 ShortName,
3726     name_lastName                     ShortName,
3727
3728   --pager attribute flattened
3729     pager_areaCode                  VARCHAR(4) NOT NULL,
3730     pager_countryCode               VARCHAR(4),
3731     pager_extension                 VARCHAR(8),
3732     pager_umber                     VARCHAR(8) NOT NULL,
3733     pager_url                                 URI
3734
3735   --telephone attribute flattened
3736     telephone_areaCode              VARCHAR(4) NOT NULL,
3737     telephone_countryCode           VARCHAR(4),
3738     telephone_extension             VARCHAR(8),
3739     telephone_umber                       VARCHAR(8) NOT NULL,
3740     telephone_url                             URI,
3741
```

```
3742      url                                                        URI,
3743
3744   );
3745
3746   CREATE TABLE Organization (
3747   --RegistryObject Attributes
3748      id                                                UUID PRIMARY KEY NOT NULL,
3749      name                                              LongName,
3750      description                           FreeFormText,
3751      accessControlPolicy           UUID NOT NULL,
3752
3753   --Versionable attributes
3754      majorVersion                             INT DEFAULT 0 NOT NULL,
3755      minorVersion                             INT DEFAULT 1 NOT NULL,
3756
3757   --RegistryEntry attributes
3758      status                                        INT DEFAULT 0 NOT NULL,
3759      userVersion                          ShortName,
3760      stability                                    INT    DEFAULT 0 NOT NULL,
3761      expirationDate                  TIMESTAMP,
3762
3763   --Organization attributes
3764
3765   --Organization.address attribute flattened
3766      address_city                          ShortName,
3767      address_country               ShortName,
3768      address_postalCode         ShortName,
3769      address_state                      ShortName,
3770      address_street                  ShortName,
3771
3772   --primary contact for Organization, points to a User.
3773   --Note many Users may belong to the same Organization
3774      contact                                          UUID NOT NULL,
3775
3776   --Organization.fax attribute falttened
3777      fax_areaCode                     VARCHAR(4) NOT NULL,
3778      fax_countryCode              VARCHAR(4),
3779      fax_extension                 VARCHAR(8),
3780      fax_umber                            VARCHAR(8) NOT NULL,
3781      fax_url                               URI,
3782
3783   --Organization.parent attribute
3784      parent                                   UUID,
3785
3786   --Organization.telephone attribute falttened
3787      telephone_areaCode              VARCHAR(4) NOT NULL,
3788      telephone_countryCode           VARCHAR(4),
3789      telephone_extension             VARCHAR(8),
3790      telephone_umber                      VARCHAR(8) NOT NULL,
3791      telephone_url                           URI
3792   );
3793
3794
3795   --Note that the ebRIM security view is not visible through the public query mechanism
3796   --in the current release
3797
3798
3799   --The RegistryEntry View allows polymorphic queries over all ebRIM classes derived
3800   --from RegistryEntry
3801
3802   CREATE VIEW RegistryEntry (
3803   --RegistryObject Attributes
3804      id,
3805      name,
3806      description,
3807      accessControlPolicy,
3808
3809   --Versionable attributes
3810      majorVersion,
3811      minorVersion,
```

```
3812
3813    --RegistryEntry attributes
3814      status,
3815      userVersion,
3816      stability,
3817      expirationDate
3818
3819    ) AS
3820     SELECT
3821    --RegistryObject Attributes
3822      id,
3823      name,
3824      description,
3825      accessControlPolicy,
3826
3827    --Versionable attributes
3828      majorVersion,
3829      minorVersion,
3830
3831    --RegistryEntry attributes
3832      status,
3833      userVersion,
3834      stability,
3835      expirationDate
3836
3837     FROM ExtrinsicObject
3838     UNION
3839
3840     SELECT
3841    --RegistryObject Attributes
3842      id,
3843      name,
3844      description,
3845      accessControlPolicy,
3846
3847    --Versionable attributes
3848      majorVersion,
3849      minorVersion,
3850
3851    --RegistryEntry attributes
3852      status,
3853      userVersion,
3854      stability,
3855      expirationDate
3856     FROM (Registry)Package
3857     UNION
3858
3859     SELECT
3860    --RegistryObject Attributes
3861      id,
3862      name,
3863      description,
3864      accessControlPolicy,
3865
3866    --Versionable attributes
3867      majorVersion,
3868      minorVersion,
3869
3870    --RegistryEntry attributes
3871      status,
3872      userVersion,
3873      stability,
3874      expirationDate
3875    FROM ClassificationNode;
```

3876

Appendix D   Non-normative Content Based Ad Hoc Queries

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

D.1.1 Automatic Classification of XML Content

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

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

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

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

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

D.1.2 Index Definition

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

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

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

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

3912   2. contentIdentifier: This attribute identifies a specific data element within the
3913   document instances of the schema using an XPATH expression as defined by
3914   [XPT].

3915   D.1.3   Example Of Index Definition

3916   To define an index that automatically classifies a CPP based upon the roles defined
3917   within its RoleName elements, the following index must be defined on the CPP schema
3918   or DTD:

```
3919     <ClassificationIndex
3920             classificationNode='id-for-role-classification-scheme'
3921             contentIdentifier='/Role//RoleName'
3922     />
```

3923   D.1.4   Proposed XML Definition

```
3924     <!--
3925     A ClassificationIndexList is specified on ExtrinsicObjects of objectType
3926     'Schema' to define an automatic Classification of instance objects of the
3927     schema using the specified classificationNode as parent and a
3928     ClassificationNode created or selected by the object content as selected by
3929     the contentIdentifier
3930     -->
3931     <!ELEMENT ClassificationIndex EMPTY>
3932     <!ATTLIST ClassificationIndex
3933             %ObjectAttributes;
3934             classificationNode IDREF #REQUIRED
3935             contentIdentifier CDATA #REQUIRED
3936     >

3938     <!-- ClassificationIndexList contains new ClassificationIndexes  -->
3939     <!ELEMENT ClassificationIndexList (ClassificationIndex)*>
```

3940   D.1.5   Example of Automatic Classification

3941   Assume that a CPP is submitted that defines two roles as "seller" and "buyer." When the
3942   CPP is submitted it will automatically be classified by two ClassificationNodes named
3943   "buyer" and "seller" that are both children of the ClassificationNode (e.g. a node named
3944   Role) specified in the classificationNode attribute of the ClassificationIndex. Note that if
3945   either of the two ClassificationNodes named "buyer" and "seller" did not previously exist,
3946   the ObjectManager would automatically create these ClassificationNodes.

3947   Appendix E   Security Implementation Guideline

3948   This section provides a suggested blueprint for how security processing may be
3949   implemented in the Registry. It is meant to be illustrative not prescriptive. Registries
3950   may choose to have different implementations as long as they support the default
3951   security roles and authorization rules described in this document.

E.1 Authentication

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

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

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

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

E.2 Authorization

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

E.3 Registry Bootstrap

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

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

E.4 Content Submission – Client Responsibility

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

E.5 Content Submission – Registry Responsibility

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

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

3. The RegistryEntry is assigned the singleton default AccessControlPolicy.

3982   4. If a principal with the identity of the SO is not available, an identity object with the
3983   SO's DN is created

3984   5. A principal with this identity is created

E.6 Content Delete/Deprecate – Client Responsibility

3986   The Registry client has to sign the payload (not entire message) before submission, for
3987   authentication purposes; otherwise, the request will be rejected

E.7 Content Delete/Deprecate – Registry Responsibility

3989   1. Like any other request, the client will be first authenticated. In this case, the Principal
3990   object will get the DN from the certificate. As there will be a principal with this identity
3991   in the Registry, the Principal object will get all the roles from that object

3992   2. As per the request in the message (delete or deprecate), the appropriate method in
3993   the RegistryObject class will be accessed.

3994   3. The access controller performs the authorization by iterating through the Permission
3995   objects associated with this object via the singleton default AccessControlPolicy.

3996   4. If authorization succeeds then the action will be permitted. Otherwise an error
3997   response is sent back with a suitable AuthorizationException error message.

Appendix F   Native Language Support (NLS)

F.1 Definitions

4000   Although this section discusses only character set and language, the following terms
4001   have to be defined clearly.
4002

F.1.1 Coded Character Set (CCS):

4004   CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130].
4005   Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.
4006

F.1.2 Character Encoding Scheme (CES):

4008   CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of
4009   CES are ISO-2022, UTF-8.

F.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].

F.2 NLS And Request / Response Messages

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

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

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

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

F.3 NLS And Storing of RegistryEntry

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

F.3.1   Character Set of *RegistryEntry*

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

F.3.2   Language Information of *RegistryEntry*

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

F.4 NLS And Storing of Repository Items

4044

4045 This section provides NLS guidelines on how a registry should store repository items.

F.4.1 Character Set of Repository Items

4046

4047 Unlike the character set of *RegistryEntry*, the charset of a repository item must be
4048 preserved as it is originally specified in the transaction. The registry may use   a special
4049 Slot with name *repositoryItemCharset,* and sloType of *nls* for the *RegistryEntry* for
4050 storing the charset of the corresponding repository item. Value must be the one defined
4051 in [RFC 2277], [RFC 2278]. The *repositoryItemCharset* is optional because not all
4052 repository items require it.

F.4.2 Language information of repository item

4053

4054 Specifying only character set is not enough to tell which language is used in the
4055 repository item. A registry may use a special Slot with name *repositoryItemLang,* and
4056 sloType of *nls* to store that information. This attribute is optional because not all
4057 repository items require it. Value must be compliant to [RFC 1766]
4058
4059 This document currently specifies only the method of sending the information of
4060 character set and language, and how it is stored in a registry. However, the language
4061 information may be used as one of the query criteria, such as retrieving only DTD
4062 written in French. Furthermore, a language negotiation procedure, like registry client is
4063 asking a favorite language for messages from registry services, could be another
4064 functionality for the future revision of this document.

Appendix G   Terminology Mapping

4065

4066 While every attempt has been made to use the same terminology used in other works
4067 there are some terminology differences.

4068 The following table shows the terminology mapping between this specification and that
4069 used in other specifications and working groups.

4070

This Document	OASIS	ISO 11179
"repository item"	RegisteredObject	
RegistryEntry	RegistryEntry	Administered Component
ExternalLink	RelatedData	N/A
Object.id	regEntryId, orgId, etc.	
ExtrinsicObject.uri	objectURL	
ExtrinsicObject.objectType	defnSource, objectType	

RegistryEntry.name	commonName	
Object.description	shortDescription, Description	
ExtrinsicObject.mimeType	objectType="mime" fileType="<mime type>"	
Versionable.majorVersion	userVersion only	
Versionable.minorVersion	userVersion only	
RegistryEntry.status	registrationStatus	

4071   **Table 1: Terminology Mapping Table**

4072

References

4072

4073 [Bra97] Keywords for use in RFCs to Indicate Requirement Levels.

4074 [GLS] ebXML Glossary, http://www.ebxml.org/documents/199909/terms_of_reference.htm

4075 [TA]   ebXML Technical Architecture

4076   http://www.ebxml.org/specdrafts/ebXML_TA_v1.0.pdf

4077 [OAS] OASIS Information Model

4078   http://www.nist.gov/itl/div897/ctg/regrep/oasis-work.html

4079 [ISO]   ISO 11179 Information Model

4080 http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba785256621005419d7/b83fc
4081   7816a6064c68525690e0065f913?OpenDocument

4082 [ebRIM] ebXML Registry Information Model

4083   http://www.ebxml.org/project_teams/registry/private/registryInfoModelv0.54.pdf

4084 [ebBPM]   ebXML Business Process Specification Schema

4085   http://www.ebxml.org/specdrafts/Busv2-0.pdf

4086 [ebCPP]   ebXML Collaboration-Protocol Profile and Agreement Specification

4087   http://www.ebxml.org/project_teams/trade_partner/private/

4088 [ebXML-UDDI] Using UDDI to Find ebXML Reg/Reps

4089   http://lists.ebxml.org/archives/ebxml-regrep/200104/msg00104.html

4090 [CTB] Context table informal document from Core Components

4091 [ebMS]   ebXML Messaging Service Specification, Version 0.21

4092 http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf

4093 [ERR] ebXML TRP Error Handling Specification

4094 http://www.ebxml.org/project_teams/transport/ebXML_Message_Service_Specification_v-0.8_001110.pdf

4095 [SEC] ebXML Risk Assessment Technical Report, Version 3.6

4096   http://lists.ebxml.org/archives/ebxml-ta-security/200012/msg00072.html

4097 [XPT] XML Path Language (XPath) Version 1.0

4098   http://www.w3.org/TR/xpath

4099 [SQL] Structured Query Language (FIPS PUB 127-2)

4100   http://www.itl.nist.gov/fipspubs/fip127-2.htm

4101

4102 [SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
4103   (SQL/PSM) [ISO/IEC 9075-4:1996]

4104

4105   [IANA]   IANA (Internet Assigned Numbers Authority).
4106   Official Names for Character Sets, ed. Keld Simonsen et al.
4107   ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets

4108

4109   [RFC 1766]   IETF (Internet Engineering Task Force). RFC 1766:
4110   Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
4111   http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html

4112

4113   [RFC 2277]   IETF (Internet Engineering Task Force). RFC 2277:
4114   IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
4115   http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html

4116

4117   [RFC 2278]   IETF (Internet Engineering Task Force). RFC 2278:
4118   IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
4119   http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html

4120

4121   [RFC 3023]   IETF (Internet Engineering Task Force). RFC 3023:
4122   XML Media Types, ed. M. Murata. 2001.
4123   ftp://ftp.isi.edu/in-notes/rfc3023.txt

4124

4125   [REC-XML]   W3C Recommendation. Extensible Markup language(XML)1.0(Second
4126   Edition)
4127   http://www.w3.org/TR/REC-xml

4128

4129   [UUID] DCE 128 bit Universal Unique Identifier
4130   http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
4131   http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

4132   Disclaimer

4133   The views and specification expressed in this document are those of the authors and
4134   are not necessarily those of their employers. The authors and their employers
4135   specifically disclaim responsibility for any problems arising from correct or incorrect
4136   implementation or use of this design.

4137

Contact Information

4138   Team Leader

4139   Name: Scott Nieman

4140   Company: Norstan Consulting

4141   Street: 5101 Shady Oak Road

4142   City, State, Postal Code: Minnetonka, MN 55343

4143   Country: USA

4144   Phone: 952.352.5889

4145   Email: Scott.Nieman@Norstan

4146

4147   Vice Team Lead

4148   Name: Yutaka Yoshida

4149   Company: Sun Microsystems

4150   Street: 901 San Antonio Road, MS UMPK17-102

4151   City, State, Postal Code: Palo Alto, CA 94303

4152   Country: USA

4153   Phone: 650.786.5488

4154   Email: Yutaka.Yoshida@eng.sun.com

4155

4156   Editor

4157   Name: Farrukh S. Najmi

4158   Company: Sun Microsystems

4159   Street: 1 Network Dr., MS BUR02-302

4160   City, State, Postal Code: Burlington, MA, 01803-0902

4161   Country: USA

4162   Phone: 781.442.0703

4163   Email: najmi@east.sun.com

4164

4165

Copyright Statement

4165

4166   Copyright © UN/CEFACT and OASIS, 2001. All Rights Reserved.

4167

4168   This document and translations of it may be copied and furnished to others, and
4169   derivative works that comment on or otherwise explain it or assist in its implementation
4170   MAY be prepared, copied, published and distributed, in whole or in part, without
4171   restriction of any kind, provided that the above copyright notice and this paragraph are
4172   included on all such copies and derivative works. However, this document itself MAY
4173   not be modified in any way, such as by removing the copyright notice or references to
4174   ebXML, UN/CEFACT, or OASIS, except as required to translate it into languages other
4175   than English.

4176

4177   The limited permissions granted above are perpetual and will not be revoked by ebXML
4178   or its successors or assigns.

4179

4180   This document and the information contained herein is provided on an

4181   "AS IS" basis and ebXML DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED,
4182   INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
4183   INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
4184   WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
4185   PURPOSE.