This document is a proposed revision
of Section 8.2 of ebRS v1.1
Submitted by Len Gallagher, Sept 27, 2001

8. ~~Object(?)~~ Query Management Service

NOTE to EDITOR: Some changes will be needed to this introduction to reflect the current status of Section 8.1, Browse and Drill Down Query Support", and Section 8.3, "SQL Query Support". This proposal deals only with Section 8.2, "Filter Query Support".

8.1 Browse and Drill Down Query Support

NOTE to EDITOR: May be deleted in favor of Filter Query. That proposed action is independent of this proposal.

8.2 Filter Query Support

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

A client submits a FilterQuery, a ReturnRegistryEntry query, or a ReturnRepositoryItem query to the ObjectQueryManager as part of an AdhocQueryRequest. The ObjectQueryManager sends an AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResponse, ReturnRegistryEntryResponse, or ReturnRepositoryItemResponse specified herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specified in Section **Error! Reference source not found.**.

Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated with Z. The ebRIM Binding for C might be as in Figure 1.

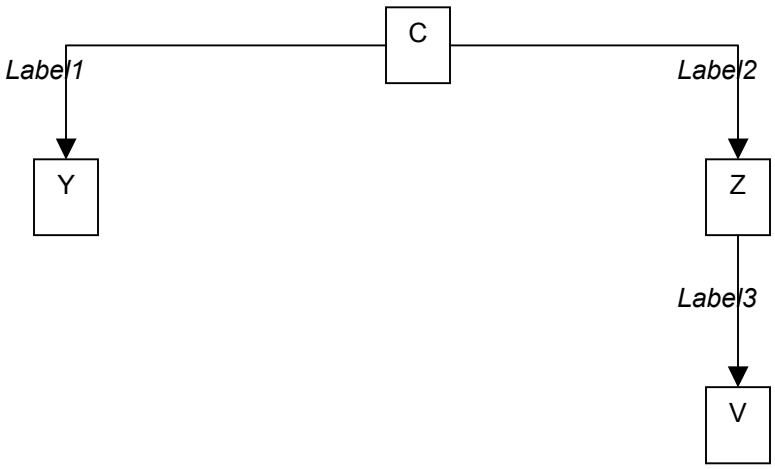


Figure 1: Example ebRIM Binding

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

Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a restricted *predicate clause* over the attributes of a single class. The supported class filters are specified in Section 8.2.9 and the supported predicate clauses are defined in Section **Error! Reference source not found.**.

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

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

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

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

Support for FilterQuery is required of every conforming ebXML Registry implementation, but other query options are possible. The Registry will hold a self-describing CPP that identifies all supported AdhocQuery options. This profile is described in Section **Error! Reference source not found.**.

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

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

8.2.1 FilterQuery

Purpose

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

Definition

```
<!ELEMENT FilterQuery
 (    RegistryEntryQuery
   |  AuditableEventQuery
   |  ClassificationNodeQuery
   |  RegistryPackageQuery
   |  OrganizationQuery        )>

<!ELEMENT FilterQueryResult
  (   RegistryEntryQueryResult
   |  AuditableEventQueryResult
   |  ClassificationNodeQueryResult
   |  RegistryPackageQueryResult
   |  OrganizationQueryResult   )>

<!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>

<!ELEMENT RegistryEntryView EMPTY >
<!ATTLIST RegistryEntryView
   id            CDATA     #REQUIRED
   name          CDATA     #REQUIRED
   contentURI    CDATA     #IMPLIED  >

<!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>

<!ELEMENT AuditableEventView EMPTY >
<!ATTLIST AuditableEventView
   id            CDATA     #REQUIRED
   name          CDATA     #REQUIRED
   timestamp     CDATA     #REQUIRED  >

<!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>

<!ELEMENT ClassificationNodeView EMPTY >
<!ATTLIST ClassificationNodeView
   id            CDATA     #REQUIRED
   name          CDATA     #REQUIRED
   code          CDATA     #REQUIRED  >

<!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>

<!ELEMENT RegistryPackageView EMPTY >
<!ATTLIST RegistryPackageView
   id            CDATA     #REQUIRED
   name          CDATA     #REQUIRED  >

<!ELEMENT OrganizationQueryResult ( OrganizationView* )>
```

```
<!ELEMENT OrganizationView EMPTY >
<!ATTLIST OrganizationView
   id            CDATA     #REQUIRED
   name          CDATA     #REQUIRED  >
```

Semantic Rules

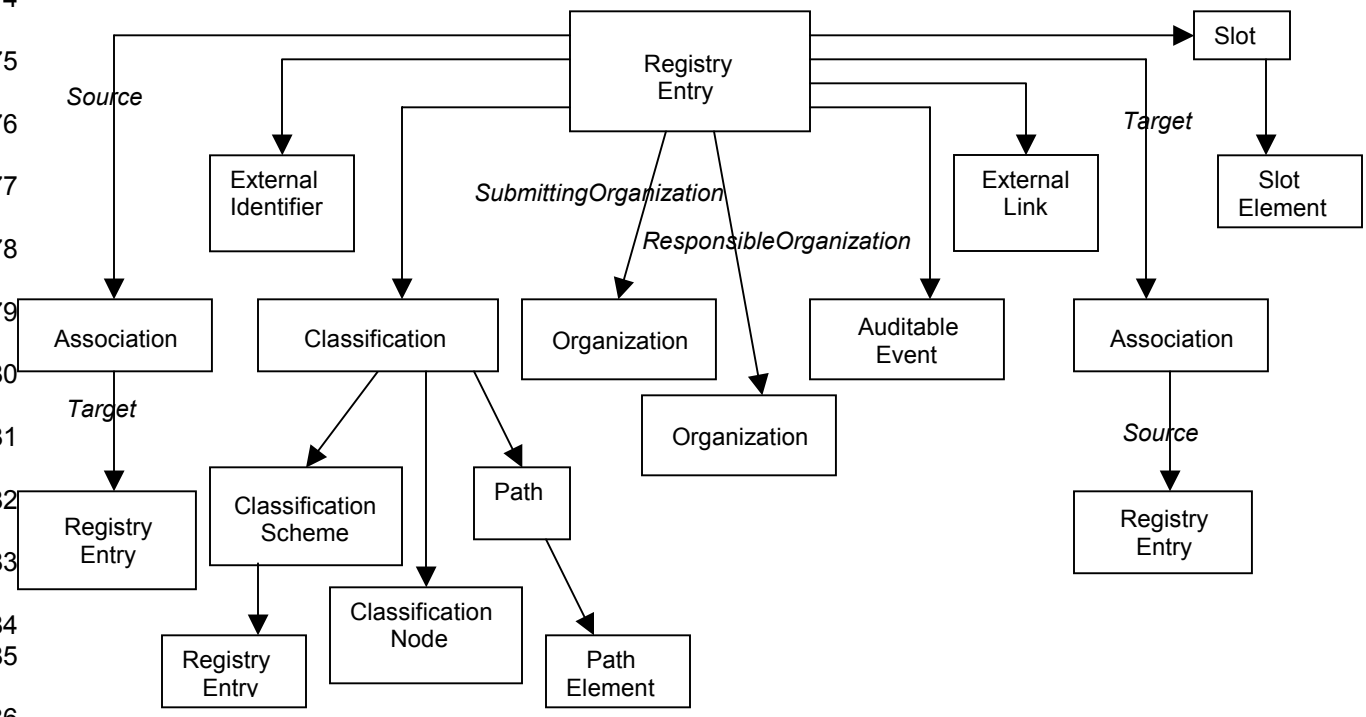
1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
2. Each FilterQueryResult is a set of XML elements to identify each instance of the result set. Each XML attribute carries a value derived from the value of an attribute specified in the Registry Information Model as follows:
   a) id carries the value of the id attribute of the RegistryObject class,
   b) name carries the value of the name attribute of the RegistryObjectClass,
   c) contentURI, if present, carries the value of the contentURI attribute of the ExtrinsicObject class,
   d) timestamp carries a character string literal value to represent the value of the timestamp attribute of the AuditableEvent class,
   e) code carries the value of the code attribute of the ClassificationNode 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”.

8.2.2 RegistryEntryQuery

Purpose

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

ebRIM Binding



Definition

```
<!ELEMENT RegistryEntryQuery
   (  RegistryEntryFilter?,
      SourceAssociationBranch*,
      TargetAssociationBranch*,
      HasClassificationBranch*,
      SubmittingOrganizationBranch?,
      ResponsibleOrganizationBranch?,
      ExternalIdentifierBranch*,
      ExternalLinkBranch*,
      HasSlotBranch*,
      HasAuditableEventBranch*         )>

<!ELEMENT SourceAssociationBranch
  (   AssociationFilter?,
      ( RegistryEntryFilter? | RegistryEntryQuery? ) )>

<!ELEMENT TargetAssociationBranch
  (   AssociationFilter?,
      ( RegistryEntryFilter? | RegistryEntryQuery? ) )>

<!ELEMENT HasClassificationBranch
  (   ClassificationFilter?,
```

```
          FromSchemeBranch?,
          HasPathBranch?,
          LocalNodeBranch?,
          SubmittingOrganizationBranch?    )>

    <!ELEMENT FromSchemeBranch
      (   ClassificationSchemeFilter | RegistryEntryQuery )>

    <!ELEMENT HasPathBranch
      (   PathFilter | XpathNodeExpression | PathElementFilter+ )>

    <!ELEMENT XpathNodeExpression  ( TO BE DETERMINED )>
```

Author'sNOTE: The HasPathBranch specifies 3 alternatives, each of which has flaws. PathFilter and PathElementFilter depend upon the definition of new methods for the ClassificationNode and Classification classes in ebRIM (cf Section 8.2.9 below), and the XpathNodeExpression depends upon a not yet existing specification for the getPath() method from the ebRIM ClassificationNode class and on specification of a subset of XPATH functionality that can be applied to the string returned from the getPath() method. It is possible that one or more of these alternatives will become superfluous with respect to the others, with likely deletion of the less useful alternative(s). The methods for Classification will depend upon a revised XML schema specification for Classification soon to emerge from the Registry external classification scheme team.

```
    <!ELEMENT LocalNodeBranch
      (   ClassificationNodeFilter? | ClassificationNodeQuery? )>

    <!ELEMENT SubmittingOrganizationBranch
      (   OrganizationFilter  |  OrganizationQuery   )>

    <!ELEMENT ResponsibleOrganizationBranch
      (   OrganizationFilter?  |  OrganizationQuery?   )>

    <!ELEMENT ExternalIdentifierBranch
      (   ExternalIdentifierFilter?,
          SubmittingOrganizationBranch?    )>

    <!ELEMENT ExternalLinkBranch
      (   ExternalLinkFilter               )>

    <!ELEMENT HasSlotBranch
      (   SlotFilter?,
          SlotElementFilter*               )>

    <!ELEMENT HasAuditableEventBranch
      (   AuditableEventFilter?  |  AuditableEventQuery? )>
```

Semantic Rules

1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
   a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below; otherwise, let x be a registry entry in RE. If x does not satisfy the RegistryEntryFilter as defined in Section **8.2.9**, then remove x from RE.
   b) If a SourceAssociationBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not the source object of some Association instance, then remove x from RE; otherwise, treat each SourceAssociationBranch element separately as follows:

If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be the set of all Association instances that have x as a source object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the source object. If AF is empty, then remove x from RE. If no RegistryEntryFilter or RegistryEntryQuery is specified within SourceAssociationBranch, then let RET be the set of all RegistryEntry instances that are the target object of some element of AF; otherwise, let RET be the set of RegistryEntry instances that satisfy the RegistryEntryFilter or RegistryEentryQuery and are the target object of some element of AF. If RET is empty, then remove x from RE.

c) If a TargetAssociationBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not the target object of some Association instance, then remove x from RE; otherwise, treat each TargetAssociationBranch element separately as follows:
If no AssociationFilter is specified within TargetAssociationBranch, then let AF be the set of all Association instances that have x as a target object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the target object. If AF is empty, then remove x from RE. If no RegistryEntryFilter or RegistryEntryQuery is specified within TargetAssociationBranch, then let RES be the set of all RegistryEntry instances that are the source object of some element of AF; otherwise, let RES be the set of RegistryEntry instances that satisfy the RegistryEntryFilter or RegistryEntryQuery and are the source object of some element of AF. If RES is empty, then remove x from RE.

d) If a HasClassificationBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not the classifiedObject of some Classification instance, then remove x from RE; otherwise, treat each HasClassificationBranch element separately as follows:
If no ClassificationFilter is specified within the HasClassificationBranch, then let CL be the set of all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then remove x from RE and continue below. Otherwise, if CL is not empty, and if a FromSchemeBranch is specified, then replace CL by the set of remaining Classification instances in CL whose defining classification scheme satisfies the ClassificationSchemeFilter or the RegistryEntryQuery immediately contained in the FromSchemeBranch. If the new CL is empty, then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a HasPathBranch is specified, then replace CL by the set of remaining Classification instances in CL that satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in the HasPathBranch. If the new CL is empty, then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a LocalNodeBranch is specified, then replace CL by the set of remaining Classification instances in CL for which a local node exists and for which that local node satisfies the ClassificationNodeFilter or the ClassificationNodeQuery immediately contained in the LocalNodeBranch. . If the new CL is empty, then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a SubmittingOrganizationBranch is specified, then replaceCL by the set of remaining Classification instances in CL for which the submitting organization of that classification satisfies the OrganizationFilter or OrganizationQuery immediately contained in the SubmittingOrganizationBranch. If the new CL is empty, then remove x from RE.

e) If a SubmittingOrganizationBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If the submitting organization for x does not satisfy the OrganizationFilter or OrganizationQuery immediately contained in the SubmittingOrganizationBranch, then remove x from RE.

f) If a ResponsibleOrganizationBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x does not have a responsible organization, then remove x from RE and continue below; otherwise, if an OrganizationFilter or OrganizationQuery is specified within the ResponsibleOrganizationBranch and if the responsible organization for x does not satisfy the OrganizationFilter or OrganizationQuery, then remove x from RE.

g) If an ExternalIdentifierBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some ExternalIdentifier instance, then remove x from RE; otherwise, treat each ExternalIdentifierBranch element

separately as follows: If an ExternalIdentifierFilter is not specified, then let EI be the set of ExternalIdentifier instances that are linked to x; otherwise, let EI be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RE and continue below. Otherwise, if EI remains not empty, and if a SubmittingOrganizationBranch is specified, replace EI by the set of remaining ExternalIdentifier instances in EI for which the OrganizationFilter or OrganizationQuery immediately contained in the SubmittingOrganizationBranch is valid. If the new EI is empty, then remove x from RE.

h) If an ExternalLinkBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some ExternalLink instance, then remove x from RE; otherwise, treat each ExternalLinkBranch element separately as follows: Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter directly contained in the ExternalLinkBranch and are linked to x. If EL is empty, then remove x from RE.

i) If a HasSlotBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some Slot instance, then remove x from RE and continue below; otherwise, treat each HasSlotBranch element separately as follows: If a SlotFilter is not specified within HasSlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty, then remove x from RE and continue below. Otherwise, if SL remains not empty, and if a SlotElementFilter is specified, replace SL by the set of remaining Slot instances in SL for which every specified SlotElementFilter is valid. If SL is empty, then remove x from RE.

j) If a HasAuditableEventBranch element is not specified, or if RE is empty, then continue below; otherwise, let x be a remaining registry entry in RE. If x is not linked to some AuditableEvent instance, then remove x from RE; otherwise, treat each HasAuditableEventBranch element separately as follows: If an AuditableEventFilter or AuditableEventQuery is not specified within HasAuditableEventBranch, then let AE be the set of all AuditableEvent instances for x; otherwise, let AE be the set of AuditableEvent instances that satisfy the AuditableEventFilter or AuditableEventQuery and are auditable events for x. If AE is empty, then remove x from RE.

2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return RE as the result of the RegistryEntryQuery.
3. Return any accumulated warnings or exceptions as the StatusResult associated with the RegistryEntryQuery.

Examples

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

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
         status EQUAL "Approved"        -- code by Clause, Section Error! Reference
                                           source not found.
   </RegistryEntryFilter>
   <SubmittingOrganizationBranch>
      <OrganizationFilter>
            name CONTAINS "XYZ"         -- code by Clause, Section Error! Reference
                                           source not found.
      </OrganizationFilter>
   </SubmittingOrganizationBranch>
</RegistryEntryquery>
```

A client is using the United Nations Standard Product and Services Classification (UNSPSC) scheme and wants to identify all companies that deal with products classified as "Integrated circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol

Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of registry entry instances for CPP's of companies that deal with integrated circuit components.

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
      objectType EQUAL "CPP" AND        -- code by Clause, Section Error! Reference
                                           source not found.
      status EQUAL "Approved"
   </RegistryEntryFilter>
   <HasClassificationBranch>
      <FromSchemeBranch>
         <ClassificationSchemeFilter>
         id EQUAL "urn:org:un:spsc:cs2001"   -- code by Clause, Section Error!
                                           Reference source not found.
         </ClassificationSchemeFilter>
      </FromSchemeBranch>
      <HasPathBranch>
         <PathFilter>
            code EQUAL "321118"
         </PathFilter>
      </HasPathBranch>
   <HasClassificationBranch>
</RegistryEntryQuery>
```

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

```
<RegistryEntryQuery>
   <HasClassificationBranch>
      <FromSchemeBranch>
         <ClassificationSchemeFilter>
            id EQUAL "urn:ebxml:cs:industry"      -- code by Clause, Section Error!
                                           Reference source not found.
         </ClassificationSchemeFilter>
      </FromSchemeBranch>
      <HasPathBranch>
         <XpathExpression>
            getPath = "//Automotive"
         </XpathExpression>
      </HasPathExpression>
   </HasClassificationBranch>
   <HasClassificationBranch>
      <FromSchemeBranch>
         <ClassificationSchemeFilter>
            id EQUAL "urn:ebxml:cs:geography"     -- code by Clause, Section Error!
                                           Reference source not found.
         </ClassificationSchemeFilter>
      </FromSchemeBranch>
      <HasPathBranch>
         <PathFilter>
            path STARTSWITH "/Asia/Japan"
         </PathFilter>
```

```
         </HasPathBranch>
      </HasClassificationBranch>
   </RegistryEntryQuery>
```

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

```
   <RegistryEntryQuery>
      <RegistryEntryFilter>
         objectType EQUAL "RegistryPackage"    -- code by Clause, Section Error!
                                                  Reference source not found.
      </RegistryEntryFilter>
      <SourceAssociationBranch>
         <AssociationFilter>
            associationType EQUAL "HasMember"  -- code by Clause, Section Error!
                                                  Reference source not found.
         </AssociationFilter>
         <RegistryEntryFilter>
            id EQUAL "urn:path:myitem"     -- code by Clause, Section Error! Reference
                                              source not found.
         </RegistryEntryFilter>
      </SourceAssociationBranch>
   </RegistryEntryQuery>
```

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

```
   <RegistryEntryQuery>
      <HasClassificationBranch>
         <LocalNodeBranch>
            <ClassificationNodeFilter>

               name CONTAINS "transistor" OR   -- code by Clause, Section Error!
                                                  Reference source not found.
               description CONTAINS "transistor"
            </ClassificationNodeFilter>
         </LocalNodeBranch>
      </HasClassificationBranch>
   </RegistryEntryQuery>
```

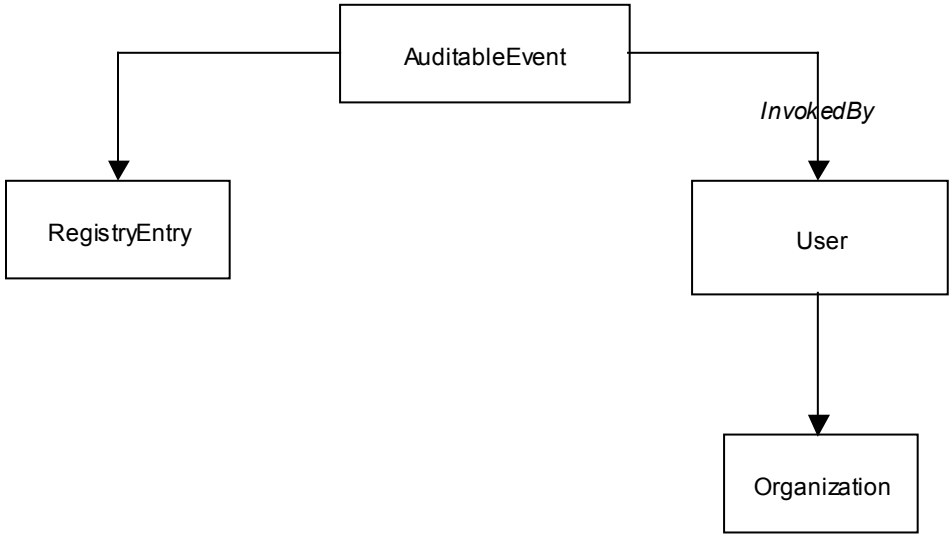
8.2.3 AuditableEventQuery

Purpose

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

ebRIM Binding

Definition




```
<!ELEMENT AuditableEventQuery
  (   AuditableEventFilter?,
      RegistryEntryQuery*,
      InvokedByBranch?     )>

<!ELEMENT InvokedByBranch
  (   UserFilter?,
      OrganizationQuery?   )>
```

Semantic Rules

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

   a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below; otherwise, let x be an auditable event in AE. If x does not satisfy the AuditableEventFilter as defined in Section 8.2.9, then remove x from AE.
   b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue below; otherwise, let x be a remaining auditable event in AE. Treat each RegistryEntryQuery element separately as follows:
   Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If x is not an auditable event for some registry entry in RE, then remove x from AE.
   c) If an InvokedByBranch element is not specified, or if AE is empty, then continue below; otherwise, let x be a remaining auditable event in AE.

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

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

2. If AE is empty, then raise the warning: *auditable event query result is empty*.
3. Return AE as the result of the AuditableEventQuery.

Examples

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

```
<AuditableEventquery>
   <AuditableEventFilter>
      timestamp GE "2001-01-01" AND          -- code by Clause, Section Error!
                                             Reference source not found.
      registryEntry EQUAL "urn:path:myitem"
   </AuditableEventFilter>
</AuditableEventQuery>
```

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

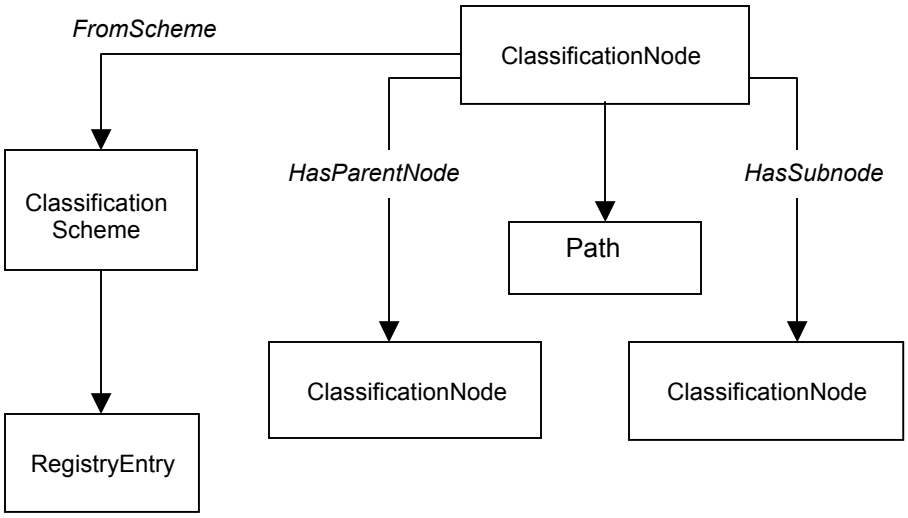
```
<AuditableEventQuery>
   <RegistryEntryQuery>
      <SubmittingOrganizationBranch>
         <OrganizationFilter>
            id EQUAL "urn:somepath:myorg"      -- code by Clause, Section Error!
                                               Reference source not found.
         </OrganizationFilter>
      </SubmittingOrganizationBranch>
      <ResponsibleOrganizationBranch>
         <OrganizationFilter>
            id EQUAL "urn:somepath:myorg"      -- code by Clause, Section Error!
                                               Reference source not found.
         </OrganizationFilter>
      </ResponsibleOrganizationBranch>
   </RegistryEntryQuery>
   <InvokedByBranch>
      <OrganizationQuery>
         <OrganizationFilter>
            id -EQUAL "urn:somepath:myorg"     -- code by Clause, Section Error!
                                               Reference source not found.
         </OrganizationFilter>
      </OrganizationQuery>
   </InvokedByBranch>
</AuditableEventQuery>
```

8.2.4 ClassificationNodeQuery

Purpose

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

ebRIM Binding



Definition

```
<!ELEMENT ClassificationNodeQuery
   (  ClassificationNodeFilter?,
      FromSchemeBranch?,
      HasPathBranch?,
      HasParentNodeBranch?,
      HasSubnodeBranch*          )>

<!ELEMENT HasParentNodeBranch
   (  ClassificationNodeFilter?,
      HasPathBranch?,
      HasParentNodeBranch?            )>

<!ELEMENT HasSubnodeBranch
   (  ClassificationNodeFilter?,
      HasPathBranch?,
      HasSubnodeBranch*
```

Semantic Rules

1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The following steps will eliminate instances in CN that do not satisfy the conditions of the specified filters.
   a) If a ClassificationNodeFilter is not specified, or if CN is empty, then continue below; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter as defined in Section **8.2.9**, then remove x from AE.

b) If a FromSchemeBranch is not specified, or if CN is empty, then continue below; otherwise, let x be a remaining classification node in CN. If the defining classification scheme of x does not satisfy the ClassificationSchemeFilter or the RegistryEntryQuery immediately contained in the FromSchemeBranch, then remove x from CN.

c) If a HasPathBranch is not specified, or if CN is empty, then continue below; otherwise, let x be a remaining classification node in CN. If the path derived from x does not satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in the HasPathBranch, then remove x from CN.

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

Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base level node), then remove x from CN and continue below; otherwise, let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in the HasParentNodeBranch and if p does not satisfy the ClassificationNodeFilter, then remove x from CN. If a HasPathBranch element is directly contained in HasParentNodeBranch and if the path derived from p does not satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in the HasPathBranch, then remove x from CN.

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

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

Let n be a classification node instance. If a ClassificationNodeFilter is not specified within the HasSubnodeBranch element then let CNC be the set of all classification nodes that have n as their parent node; otherwise, let CNC be the set of all classification nodes that satisfy the ClassificationNodeFilter and have n as their parent node. If CNC is empty, then remove x from CN; otherwise, let c be any member of CNC. If a HasPathBranch element is directly contained in the HasSubodeBranch and if the path derived from c does not satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in the HasPathBranch, then remove x from CN. If CNC is empty then remove x from CN; otherwise, let y be an element of CNC and continue with the next paragraph.

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

2. If CN is empty, then raise the warning: *classification node query result is empty*.
3. Return CN as the result of the ClassificationNodeQuery.

Examples

A client application wishes to identify all of the classification nodes in the first three levels of a classification scheme hierarchy. The client knows that the URN identifier for the underlying classification scheme is “urn:ebxml:cs:myscheme”. The following query identifies all nodes at the first three levels.

```
<ClassificationNodeQuery>
   <FromSchemeBranch>
      <ClassificationSchemeFilter>
          id EQUAL "urn:ebxml:cs:myscheme"   -- code by Clause, Section Error!
                                                Reference source not found.
      </ClassificationSchemeFilter>
   </FromSchemeBranch>
   <HasPathBranch>
      <PathFilter>
```

```
                pathDepth LE “3”
            </PathFilter>
        </HasPathBranch>
    </ClassificationNodeQuery>
```

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

By assuming that the "path" of a node is known, and the URN of the classification scheme it comes from, one could get all nodes at the next level below that node as follows:

```
 <ClassificationNodeQuery>
     <FromSchemeBranch>
         <ClassificationSchemeFilter>
             id EQUAL "urn:some:known:scheme"
         </ClassificationSchemeFilter>
     </FromSchemeBranch>
     <HasParentBranch>
         <HasPathBranch>
             <PathFilter>
                 path EQUAL "KnownPathOfGivenNode"
             </PathFilter>
         </HasPathBranch>
     </HasParentBranch>
  </ClassificationNodeQuery>
```

If instead, one wanted ALL nodes in the subtree beneath the given node, then the following query could be used:

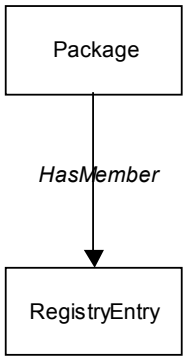
```
  <ClassificationNodeQuery>
     <FromSchemeBranch>
         <ClassificationSchemeFilter>
             id EQUAL "urn:some:known:scheme"
         </ClassificationSchemeFilter>
     </FromSchemeBranch>
     <HasParentBranch>
         <HasPathBranch>
             <PathFilter>
                 path STARTSWITH "KnownPathOfGivenNode"
             </PathFilter>
         </HasPathBranch>
     </HasParentBranch>
   </ClassificationNodeQuery>
```

8.2.5 RegistryPackageQuery

Purpose

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

ebRIM Binding



Definition

```
<!ELEMENT RegistryPackageQuery
  (   PackageFilter?,
      HasMemberBranch*     )>

<!ELEMENT HasMemberBranch
  (   RegistryEntryQuery?   )>
```

Semantic Rules

1. Let RP denote the set of all persistent Package instances in the Registry. The following steps will eliminate instances in RP that do not satisfy the conditions of the specified filters.
   a) If a PackageFilter is not specified, or if RP is empty, then continue below; otherwise, let x be a package instance in RP. If x does not satisfy the PackageFilter as defined in Section 8.2.9, then remove x from RP.
   b) If a HasMemberBranch element is not directly contained in the RegistryPackageQuery, or if RP is empty, then continue below; otherwise, let x be a remaining package instance in RP. If x is an empty package, then remove x from RP; otherwise, treat each HasMemberBranch element separately as follows:

      If a RegistryEntryQuery element is not directly contained in the HasMemberBranch element, then let PM be the set of all RegistryEntry instances that are members of the package x; otherwise, let RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in Section 8.2.2 and let PM be the subset of RE that are members of the package x. If PM is empty, then remove x from RP.
2. If RP is empty, then raise the warning: *registry package query result is empty*.
3. Return RP as the result of the RegistryPackageQuery.

Examples

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

```
<RegistryPackageQuery>
```

```
      <HasMemberBranch>
         <RegistryEntryQuery>
            <RegistryEntryFilter>
               objectType EQ "Invoice"      -- code by Clause, Section Error! Reference
                                             source not found.
            </RegistryEntryFilter>
         </RegistryEntryQuery>
      </HasMemberBranch>
   </RegistryPackageQuery>
```

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

```
   <RegistryEntryQuery>
      <HasMemberBranch/>
   </RegistryEntryQuery>
```

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

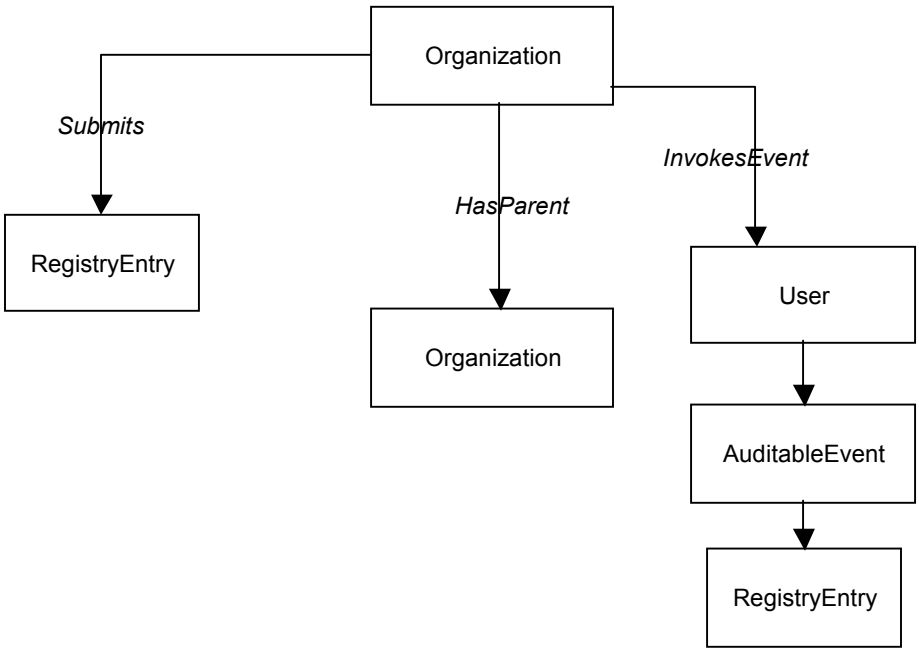
Note: A registry package is an intrinsic RegistryEntry instance that is completely determined by its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an equivalent RegistryEntryQuery using appropriate “Source” and “Target” associations. However, the equivalent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

8.2.6 OrganizationQuery

Purpose

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

ebRIM Binding



Definition

```
<!ELEMENT OrganizationQuery
  (   OrganizationFilter?,
      SubmitsRegistryEntry*,
      HasParentOrganization?,
      InvokesEventBranch*   )>

<!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>

<!ELEMENT HasParentOrganization
  (   OrganizationFilter?,
      HasParentOrganization?   )>

<!ELEMENT InvokesEventBranch
  (   UserFilter?,
      AuditableEventFilter?,
      RegistryEntryQuery?      )>
```

Semantic Rules

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

a) If an OrganizationFilter element is not directly contained in the OrganizationQuery element, or if ORG is empty, then continue below; otherwise, let x be an organization instance in ORG. If x does not satisfy the OrganizationFilter as defined in Section 8.2.9, then remove x from RP.
b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, consider each SubmitsRegistryEntry element separately as follows:

   If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element, then let RES be the set of all RegistryEntry instances that have been submitted to the Registry by organization x; otherwise, let RE be the result of the RegistryEntryQuery as defined in Section 8.2.2 and let RES be the set of all instances in RE that have been submitted to the Registry by organization x. If RES is empty, then remove x from ORG.
c) If a HasParentOrganization element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, execute the following paragraph with o = x:

   Let o be an organization instance. If an OrganizationFilter is not specified within the HasParentOrganization and if o has no parent (i.e. if o is a root organization in the Organization hierarchy), then remove x from ORG; otherwise, let p be the parent organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG.
   If another HasParentOrganization element is directly contained within this HasParentOrganization element, then repeat the previous paragraph with o = p.
d) If an InvokesEventBranch element is not specified within the OrganizationQuery, or if ORG is empty, then continue below; otherwise, consider each InvokesEventBranch element separately as follows:
   If an UserFilter is not specified, and if x is not the submitting organization of some AuditableEvent instance, then remove x from ORG. If an AuditableEventFilter is not specified, then let AE be the set of all AuditableEvent instances that have x as the submitting organization; otherwise, let AE be the set of AuditableEvent instances that satisfy the AuditableEventFilter and have x as the submitting organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is not specified in the InvokesEventBranch element, then let RES be the set of all RegistryEntry instances associated with an event in AE; otherwise, let RE be the result set of the RegistryEntryQuery, as specified in Section 8.2.2, and let RES be the subset of RE of entries submitted by x. If RES is empty, then remove x from ORG.

2. If ORG is empty, then raise the warning: *organization query result is empty*.
3. Return ORG as the result of the OrganizationQuery.

Examples

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

```
<OrganizationQuery>
   <OrganizationFilter>
      country EQUAL "France"          -- code by Clause, Section Error! Reference
                                         source not found.
   </OrganizationFilter>
   <SubmitsRegistryEntry>
      <RegistryEntryQuery>
         <RegistryEntryFilter>
            objectType EQUAL "CPP"    -- code by Clause, Section Error! Reference
                                         source not found.
         </RegistryEntryFilter>
         <HasAuditableEventBranch>
            <AuditableEventFilter>
               timestamp GE "2001-01-01"   -- code by Clause, Section Error!
                                              Reference source not found.
            </AuditableEventFilter>
         </HasAuditableEventBranch>
```

```
            </RegistryEntryQuery>
         </SubmitsRegistryEntry>
      </OrganizationQuery>
```

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

```
   <OrganizationQuery>
      <HasParentOrganization>
         <OrganizationFilter>
            id EQUAL "urn:ebxml:org:xyz"      -- code by Clause, Section Error! Reference
                                                 source not found.
         </OrganizationFilter>
      </HasParentOrganization>
   </OrganizationQuery>
```

8.2.7 ReturnRegistryEntry

Purpose

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

Definition

```
<!ELEMENT ReturnRegistryEntry
  (   RegistryEntryQuery,
      WithClassifications?,
      WithSourceAssociations?,
      WithTargetAssociations?,
      WithAuditableEvents?,
      WithExternalLinks?              )>

<!ELEMENT WithClassifications  ( ClassificationFilter? )>
<!ELEMENT WithSourceAssociations  ( AssociationFilter? )>
<!ELEMENT WithTargetAssociations  ( AssociationFilter? )>
<!ELEMENT WithAuditableEvents  ( AuditableEventFilter? )>
<!ELEMENT WithExternalLinks  ( ExternalLinkFilter? )>

<!ELEMENT ReturnRegistryEntryResult
   (  RegistryEntryMetadata*)>

<!ELEMENT RegistryEntryMetadata
    ( RegistryEntry,
      Classification*,
      SourceAssociations?,
      TargetAssociations?,
      AuditableEvent*,
      ExternalLink*                        )>

<!ELEMENT SourceAssociations ( Association* )>
<!ELEMENT TargetAssociations ( Association* )>
```

Semantic Rules

1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink elements contained in the ReturnRegistryEntryResult are defined by the ebXML Registry schema specified in Appendix A.
2. Execute the RegistryEntryQuery according to the Semantic Rules specified in Section 8.2.2, and let R be the result set of registry entry instances. Let S be the set of warnings and errors returned. If any element in S is an error condition, then stop execution and return the same set of warnings and errors along with the ReturnRegistryEntryResult.
3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*. Add this warning to the error list returned by the RegistryEntryQuery and return this enhanced error list with the ReturnRegistryEntryResult.
4. For each registry entry E referenced by an element of R, use the attributes of E to create a new RegistryEntry element as defined in Appendix A. Then create a new RegistryEntryMetadata element as defined above to be the parent element of that RegistryEntry element.
5. If no With option is specified, then the resulting RegistryEntryMetadata element has no Classification, SourceAssociations, TargetAssociations, AuditableEvent, or ExternalData subelements. The set of RegistryEntryMetadata elements, with the Error list from the RegistryEntryQuery, is returned as the ReturnRegistryEntryResult.
6. If WithClassifications is specified, then for each E in R do the following: If a ClassificationFilter is not present, then let C be any classification instance linked to E; otherwise, let C be a classification

instance linked to E that satisfies the ClassificationFilter (Section 8.2.9). For each such C, create a new Classification element as defined in Appendix A. Add these Classification elements to their parent RegistryEntryMetadata element.

7. If WithSourceAssociations is specified, then for each E in R do the following: If an AssociationFilter is not present, then let A be any association instance whose source object is E; otherwise, let A be an association instance that satisfies the AssociationFilter (Section 8.2.9) and whose source object is E. For each such A, create a new Association element as defined in Appendix A. Add these Association elements as subelements of the WithSourceAssociations and add that element to its parent RegistryEntryMetadata element.
8. If WithTargetAssociations is specified, then for each E in R do the following: If an AssociationFilter is not present, then let A be any association instance whose target object is E; otherwise, let A be an association instance that satisfies the AssociationFilter (Section 8.2.9) and whose target object is E. For each such A, create a new Association element as defined in Appendix A. Add these Association elements as subelements of the WithTargetAssociations and add that element to its parent RegistryEntryMetadata element.
9. If WithAuditableEvents is specified, then for each E in R do the following: If an AuditableEventFilter is not present, then let A be any auditable event instance linked to E; otherwise, let A be any auditable event instance linked to E that satisfies the AuditableEventFilter (Section 8.2.9). For each such A, create a new AuditableEvent element as defined in Appendix A. Add these AuditableEvent elements to their parent RegistryEntryMetadata element.
10. If WithExternalLinks is specified, then for each E in R do the following: If an ExternalLinkFilter is not present, then let L be any external link instance linked to E; otherwise, let L be any external link instance linked to E that satisfies the ExternalLinkFilter (Section 8.2.9). For each such D, create a new ExternalLink element as defined in Appendix A. Add these ExternalLink elements to their parent RegistryEntryMetadata element.
11. If any warning or error condition results, then add the code and the message to the RegistryResponse element that includes the RegistryEntryQueryResult.
12. Return the set of RegistryEntryMetadata elements as the content of the ReturnRegistryEntryResult.

Examples

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

```
<ReturnRegistryEntry>
   <RegistryEntryQuery>
      <RegistryEntryFilter>
         id EQUAL "urn:com:xyz:po:325"      -- code by Clause, Section Error!
                                            Reference source not found.
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <WithClassifications/>
   <WithSourceAssociations>
      <AssociationFilter>                   -- code by Clause, Section Error!
                                            Reference source not found.
         associationType EQUAL "SupersededBy" OR
         associationType EQUAL "ReplacedBy"
      </AssociationFilter>
   </WithSourceAssociations>
</ReturnRegistryEntry>
```

A client of the Registry registered an XML DTD several years ago and is now thinking of replacing it with a revised version. The identifier for the existing DTD is "urn:xyz:dtd:po97". The proposed revision is not

completely upward compatible with the existing DTD. The client desires a list of all registered items that use the existing DTD so they can assess the impact of an incompatible change. The following query returns an XML document that is a list of all RegistryEntry elements that represent registered items that use, contain, or extend the given DTD. The document also links each RegistryEntry element in the list to an element for the identified association.

```
<ReturnRegistryEntry>
   <RegistryEntryQuery>
      <SourceAssociationBranch>
         <AssociationFilter>            -- code by Clause, Section Error! Reference
                                        source not found.
            associationType EQUAL "Contains" OR
            associationType EQUAL "Uses" OR
            associationType EQUAL "Extends"
         </AssociationFilter>
         <RegistryEntryFilter>          -- code by Clause, Section Error! Reference
                                        source not found.
            id EQUAL "urn:xyz:dtd:po97"
         </RegistryEntryFilter>
      </SourceAssociationBranch>
   </RegistryEntryQuery>
   <WithSourceAssociations>
      <AssociationFilter>               -- code by Clause, Section Error!
                                        Reference source not found.
         associationType EQUAL "Contains" OR
         associationType EQUAL "Uses" OR
         associationType EQUAL "Extends"
      </AssociationFilter>
   </WithSourceAssociations>
</ReturnRegistryEntry>
```

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

```
<ReturnRegistryEntry>
   <RegistryEntryQuery>
      <TargetAssociationBranch>
         <AssociationFilter>          -- code by Clause, Section Error! Reference
                                      source not found.
             associationType EQUAL "HasMember"
         </AssociationFilter>
         <RegistryEntryFilter>        -- code by Clause, Section Error! Reference
                                      source not found.
            id EQUAL "urn:com:cc:pkg:ccstuff"
         </RegistryEntryFilter>
      </TargetAssociationBranch>
   </RegistryEntryQuery>
   <WithSourceAssociations>
      <AssociationFilter>             -- code by Clause, Section Error! Reference
                                      source not found.
         associationType EQUAL "HasMember" OR
         associationType EQUAL "Uses"
      </AssociationFilter>
```

```
            </WithSourceAssociations>
        </ReturnRegistryEntry>
```

8.2.8 ReturnRepositoryItem

Purpose

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

Definition

```
<!ELEMENT ReturnRepositoryItem
(  RegistryEntryQuery,
   RecursiveAssociationOption?,
   WithDescription?             )>

<!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
<!ATTLIST RecursiveAssociationOption
      depthLimit   CDATA   #IMPLIED   >

<!ELEMENT AssociationType EMPTY >
<!ATTLIST AssociationType
      role   CDATA   #REQUIRED  >

<!ELEMENT WithDescription EMPTY >

<!ELEMENT ReturnRepositoryItemResult
   ( RepositoryItem*)>

<!ELEMENT RepositoryItem
   (    ClassificationSchemeRepresentation
      | RegistryPackageElements
      | ExtrinsicObjectFile
      | WithdrawnObject
      | ExternalRegistryItem      )>
<!ATTLIST RepositoryItem
   id              CDATA   #REQUIRED
   name            CDATA   #REQUIRED
   objectType      CDATA   #REQUIRED
   status          CDATA   #REQUIRED
   stability       CDATA   #REQUIRED
   contentURI      CDATA   #IMPLIED
   description     CDATA   #IMPLIED   >

<!ELEMENT ClassificationSchemeRepresentation
(  ClassificationNode+                    )>

<!ELEMENT RegistryPackageElements
   (  RegistryObject*                     )>

<!ELEMENT ExtrinsicObjectFile  EMPTY  >
<!ATTLIST ExtrinsicObjectFile
   contentURI    CDATA    #REQUIRED   >    -- REF to attached file

<!ELEMENT WithdrawnObject EMPTY >

<!ELEMENT ExternalRegistryItem EMPTY >
```

Semantic Rules

1. If the RecursiveOption element is not present , then set Limit=0. If the RecursiveOption element is present, interpret its depthLimit attribute as an integer literal. If the depthLimit attribute is not present, then set Limit = -1. A Limit of 0 means that no recursion occurs. A Limit of -1 means that recursion occurs indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive integer, then stop execution and raise the exception: *invalid depth limit*; otherwise, set Limit=N, where N is that positive integer. A Limit of N means that exactly N recursive steps will be executed unless the process terminates prior to that limit.
2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned as part of the ReturnRepositoryItemResult. Initially Result is empty. Semantic rules 4 through 10 determine the content of Result.
3. If the WithDescription element is present, then set WSD="yes"; otherwise, set WSD="no".
4. Execute the RegistryEntryQuery according to the Semantic Rules specified in Section 8.2.2, and let R be the result set of registry entry instances. Let S be the set of warnings and errors returned. If any element in S is an error condition, then stop execution and return the same set of warnings and errors along with the ReturnRepositoryItemResult.
5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from R. After execution of these rules, if Depth is now equal to Limit, then return the content of Result as the set of RepositoryItem elements in the ReturnRepositoryItemResult element; otherwise, continue with Semantic Rule 8.
6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the following:
   a) If E references a repository item in this registry, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7.
      1) If E.objectType="ClassificationScheme", then put the classification scheme nodes described by E as a ClassificationSchemeRepresentation subelement of this RepositoryItem.
      2) If E.objectType="Package", then put the package members described by E as a RegistryPackageElements subelement of this RepositoryItem.
      3) Otherwise, i.e., if the repository item referenced by E has an unknown internal structure, then attach the file that represents that structure to the ReturnRepositoryItemResult. Create a new ExtrrinsicObjectFile as the subelement of this RepositoryItem and set the contentURI attribute to reference the attached file.
   b) If E references a registered object in some other registry, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7, and create a new ExternalRegistryItem element as the subelement of this RepositoryItem.
   c) If E describes a repository item that has since been withdrawn, then create a new RepositoryItem element, with values for its attributes derived as specified in Semantic Rule 7, and create a new WithdrawnObject element as the subelement of this RepositoryItem.
7. Let E be a registry entry and let RO be the RepositoryItem element created in Semantic Rule 6. Set the attributes of RO to the values derived from the corresponding attributes of E. If WSD="yes", include the value of the description attribute; otherwise, do not include it. Insert this new RepositoryItem element into the Result set.
8. Let R be defined as in Semantic Rule **Error! Reference source not found.**. Execute Semantic Rule 9 with Y as the set of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.
9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty set of RegistryEntry instances. For each registry entry E in Y, and for each AssociationType of the RecursiveAssociationOption, do the following:
   a) Let Z be the set of target items E' linked to E under Association instances having E as the source object, E' as the target object, and with the associationType of the association equal to the value of the role attribute of that AssociationType.
   b) Add the elements of Z to NextLevel.
10. Let X be the set of new registry entries that are in NextLevel but are not yet represented in the Result set.
    Case:
    a) If X is empty, then return the content of Result as the set of RepositoryItem elements in the ReturnRepositoryItemResult element.

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

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

Examples

A registry client has found a registry entry for a core-component item. The item's URN identity is "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many other registered items. The client desires the collection of all items needed for a complete implementation of "goodthing". The following query returns an XML document that is a collection of all needed items. The query follows all “Uses” and “ValidatesTo” association types through an arbitrary number of recursive steps to return every repository item in this registry that is needed by “goodthing”.

```
<ReturnRepositoryItem>
   <RegistryEntryQuery>
      <RegistryEntryFilter>                    -- code by Clause, Section Error! Reference
                                               source not found.
         id EQUAL "urn:ebxml:cc:goodthing"
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <RecursiveAssociationOption>
      <AssociationType  role="Uses"  />
      <AssociationType  role="ValidatesTo"  />
   </RecursiveAssociationOption>
</ReturnRepositoryItem>
```

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

```
<ReturnRepositoryItem>
   <RegistryEntryQuery>
      <RegistryEntryFilter>                    -- code by Clause, Section Error! Reference
                                               source not found.
         id EQUAL "urn:ebxml:cc:rtn:nice87"
      </RegistryEntryFilter>
   </RegistryEntryQuery>
   <RecursiveAssociationOption   depthLimit="1" >
      <AssociationType  role="ValidatesTo"  />
   </RecursiveAssociationOption>
</ReturnRepositoryItem>
```

A user has been told that the 1997 version of the North American Industry Classification System (NAICS) is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The following query would retrieve the complete classification scheme, with all 1810 nodes, as an XML document that contains all of the ClassificationNode instances for the ClassificationScheme instance identified by that URN.

```
<ReturnRepositoryItem>
   <RegistryEntryQuery>
      <RegistryEntryFilter>                    -- code by Clause, Section Error!
                                               Reference source not found.
         id EQUAL "urn:nist:cs:naics-1997"
      </RegistryEntryFilter>
```

```
      </RegistryEntryQuery>
    </ReturnRepositoryItem>
```

8.2.9 Registry Filters

Purpose

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

Definition

```
<!ELEMENT RegistryObjectFilter ( Clause )>

<!ELEMENT RegistryEntryFilter ( Clause )>

<!ELEMENT ExtrinsicObjectFilter ( Clause )>

<!ELEMENT PackageFilter ( Clause )>

<!ELEMENT OrganizationFilter ( Clause

<!ELEMENT ClassificationNodeFilter ( Clause )>

<!ELEMENT AssociationFilter ( Clause )>

<!ELEMENT ClassificationFilter ( Clause )>

<!ELEMENT ExternalLinkFilter ( Clause )>

<!ELEMENT ExternalIdentifierFilter ( Clause )>

<!ELEMENT SlotFilter ( Clause )>

<!ELEMENT AuditableEventFilter ( Clause )>

<!ELEMENT UserFilter ( Clause )>

<!ELEMENT PathFilter ( Clause )>

<!ELEMENT PathElementFilter ( Clause )>

<!ELEMENT SlotElementFilter ( Clause )>
```

Semantic Rules

1. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryObject UML class defined in [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the Clause predicate.
3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True* for the Clause predicate.
4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause predicate.

5. For every PackageFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Package UML class defined in [ebRIM]. If not, raise exception: *package attribute error*. The PackageFilter returns a set of identifiers for Package instances whose attribute values evaluate to *True* for the Clause predicate.
6. For every OrganizationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Organization or PostalAddress UML classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The OrganizationFilter returns a set of identifiers for Organization instances whose attribute values evaluate to *True* for the Clause predicate.
7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. The ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose attribute values evaluate to *True* for the Clause predicate.
8. For every AssociationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Association UML class defined in [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a set of identifiers for Association instances whose attribute values evaluate to *True* for the Clause predicate.
9. For every ClassificationFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Classification UML class defined in [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter returns a set of identifiers for Classification instances whose attribute values evaluate to *True* for the Clause predicate.
10. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalLink UML class defined in [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the Clause predicate.
11. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in [ebRIM]. If not, raise exception: *external identifier attribute error*. The ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose attribute values evaluate to *True* for the Clause predicate.
12. For every SlotFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot instances whose attribute values evaluate to *True* for the Clause predicate.
13. For every AuditableEventFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [ebRIM]. If not, raise exception: *auditable event attribute error*. The AuditableEventFilter returns a set of identifiers for AuditableEvent instances whose attribute values evaluate to *True* for the Clause predicate.
14. For every UserFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If not, raise exception: *auditable identity attribute error*. The UserFilter returns a set of identifiers for User instances whose attribute values evaluate to *True* for the Clause predicate.
15. Path is a derived, non-persistent class based on the ClassificationNode and Classification classes from ebRIM. The visible attributes of the Path class are “path”, “code”, and “pathDepth”. Each is derived from the corresponding method defined in ebRIM for a ClassificationNode or Classification instance. The getPath() method acts on a ClassificationNode or Classification instance to produce a character string, i.e. path, that can be queried by the predicates of a StringClause element. The getCode() method on a Classification instance returns a string value, i.e. code: (i) if an internal classification, returns the code attribute of the referenced ClassificationNode, and (ii) if an external classification, returns the classification value submitted by the classifier (ebRIM definitions needed!). The getPathDepth() method acts on a ClassificationNode or Classification instance to produce an integer that identifies the level of the referenced node and that can be queried by the predicates of an IntClause element. For an external Classification instance, getPathDepth() may return void since the depth of the node referenced by that classification may not be known if it wasn’t supplied by the classifier. For every PathFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the Path class just defined. If not, raise exception:

path attribute error. The PathFilter returns a set of Path instances whose attribute values evaluate to *True* for the Clause predicate.

16. PathElement is a derived, non-persistent class based on the ClassificationNode and Classification classes from ebRIM. The visible attributes of PathElement are “level” and “value”. Each is a character string. The dynamic instances of PathElement are derived from the getPathElements() method defined in ebRIM for a ClassificationNode or Classification instance. This method returns a set of level/value pairs for each ClassificationNode or Classification instance. For an external Classification instance, getPathElements() may return void since the explicit structure of the node referenced by that classification may not be known if it wasn’t supplied by the classifier. For every PathElementFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the PathElement class just defined. If not, raise exception: *path element attribute error*. The PathElementFilter returns a set of PathElement instances whose attribute values evaluate to *True* for the Clause predicate.
17. SlotElement is a derived, non-persistent class based on the Slot class from ebRIM. The visible attribute of PathElement is“value”. It is a character string. The dynamic instances of SlotElement are derived from the “values” attribute defined in ebRIM for a Slot instance. For every SlotElementFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify the “value” attribute of the SlotElement class just defined. If not, raise exception: *slot element attribute error*. The SlotElementFilter returns a set of Slot instances whose “value” attribute evaluates to *True* for the Clause predicate.

Example

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

```
<RegistryEntryQuery>
   <RegistryEntryFilter>
       <Clause>
          <CompoundClause   connectivePredicate="And" >
             <Clause>
                <SimpleClause  leftArgument="objectType" >
            <StringClause  stringPredicate="equal" >CPP</StringClause>
                </SimpleClause>
             </Clause>
             <Clause>
                <SimpleClause  leftArgument="status" >
            <StringClause  stringPredicate="equal" >Approved</StringClause>
                </SimpleClause>
             </Clause>
          </CompoundClause>
       </Clause>
   </RegistryEntryFilter>
</RegistryEntryQuery>
```

8.2.10 XML Clause Constraint Representation

NOTE to Editor: This proposal makes no changes to Section 8.2.10, so it remains as currently specified in ebRS v1.1.