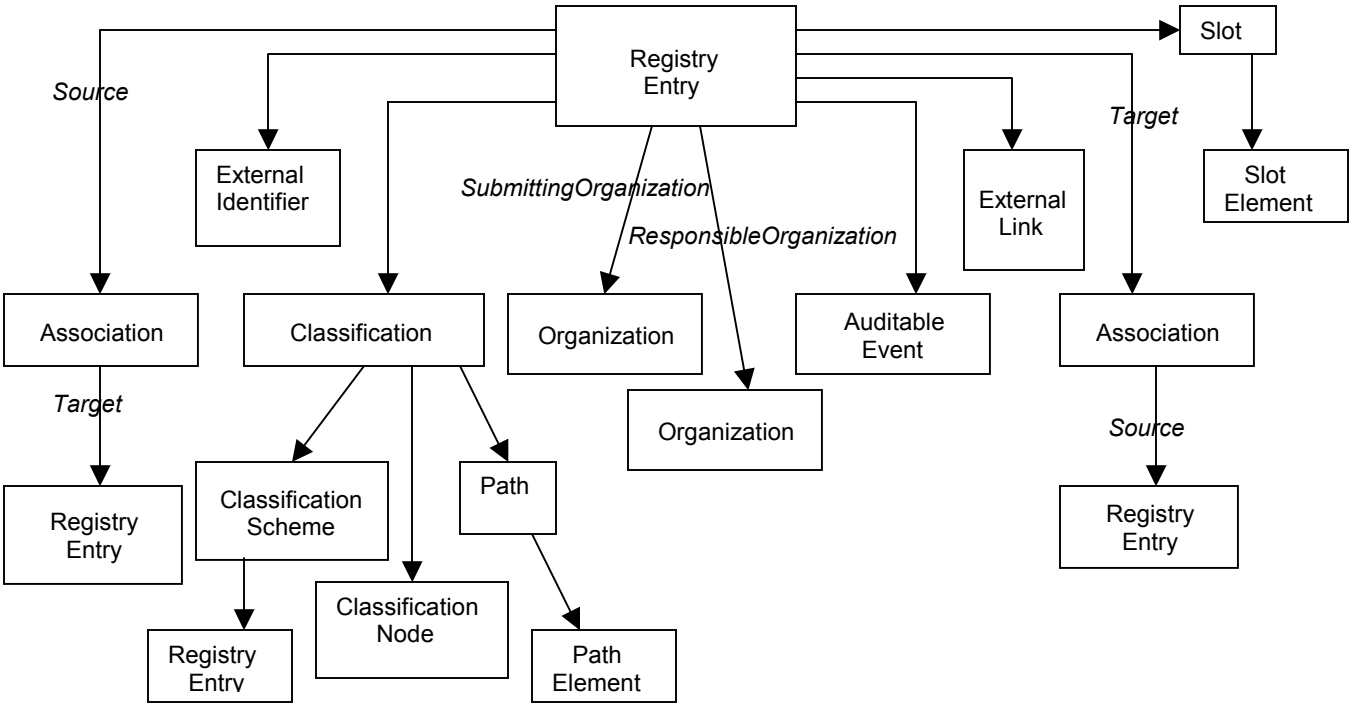
2. RegistryEntryQuery

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*,
       ExternalLinkFilter*,
       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 )>

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

<!ELEMENT SubmittingOrganizationBranch
  (   OrganizationFilter  |  OrganizationQuery   )>

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

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

<!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 **Error! Reference source not found.**, 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 ExternalLinkFilter 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 ExternalLinkFilter element separately as follows: Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter 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>
      <HasPathFilter>
         path EQUAL "321118"
      </HasPathFilter>
   <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>
         <PathFilter>
            path STARTSWITH "Automotive"
         </PathFilter>
      </HasPathBranch>
   </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>
```