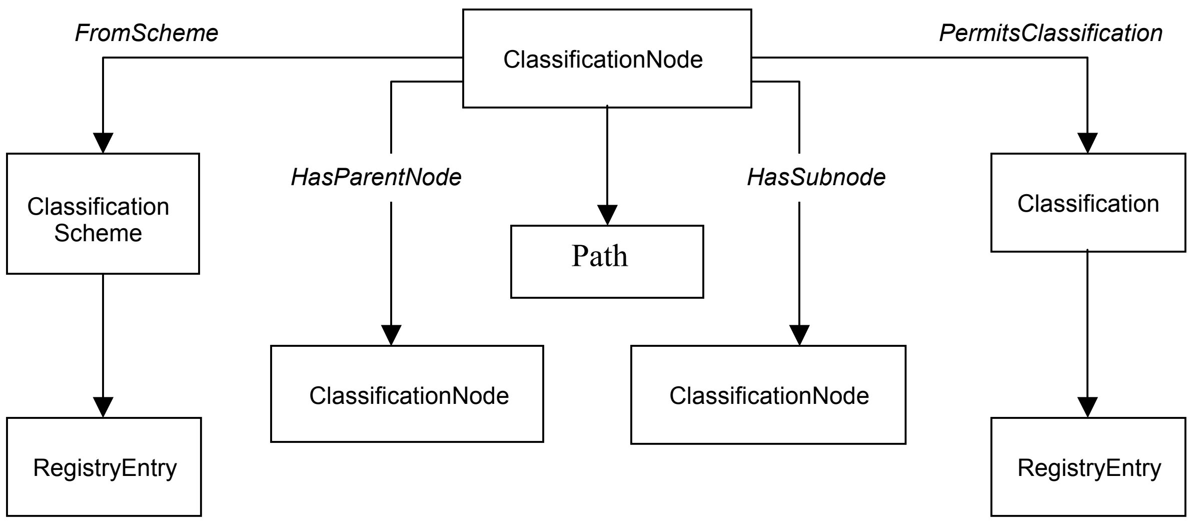
1.1.1 ClassificationNodeQuery

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*,
      PermitsClassificationBranch?  )>

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

<!ELEMENT HasSubnodeBranch
   (  ClassificationNodeFilter?,
      HasPathBranch?,
      HasSubnodeBranch*               )>

<!ELEMENT PermitsClassificationBranch
   (  ClassificationFilter?,
      RegistryEntryQuery?       )>
```

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 **Error! Reference source not found.**, 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 <u>every</u> 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 <u>every</u> 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 <u>every</u> 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.
   f) If a PermitsClassificationBranch 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 target node of some Classification instance, then remove x from CN; otherwise, treat each PermitsClassificationBranch element separately as follows:If no ClassificationFilter is specified within the PermitsClassificationBranch element, then let CL be the set of all Classification instances that have x as the target node; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the target node. If CL is empty, then remove x from CN. If no RegistryEntryQuery is specified within the PermitsClassificationBranch element, then let RES be the set of all RegistryEntry instances that are the classified object of some classification instance in CL; otherwise, let RE be the result set of the RegistryEntryQuery as defined in Section **Error! Reference source not found.** and let RES be the set of all instances in RE that are the classified object of some classification in CL. If RES is empty, then remove x from CN.
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>
```