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

```

Semantic Rules

1. Let A denote the set of all persistent Association instances in the Registry. The following steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
   a) If A is empty then continue to the next numbered rule.
   b) If an AssociationFilter element is not directly contained in the AssociationQuery element, then go to the next step; otherwise let x be an association instance in A. If x does not satisfy the AssociationFilter then remove x from A. If A is empty then continue to the next numbered rule.
   c) Let A be the set of remaining Association instances. Evaluate inherited RegistryObjectQuery over A as explained in Section 8.2.2.
2. If A is empty, then raise the warning: *association query result is empty*; otherwise, set A to be the result of the AssociationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Examples

A client application wishes to identify a set of associations that are ‘equivalentTo’ a set of other associations.

```

<AdhocQueryRequest">
  <ResponseOption returnType="LeafClass" />
  <FilterQuery>
    <AssociationQuery>
      <SourceAssociationBranch>
        <AssociationFilter>
          <Clause>
            <SimpleClause leftArgument="associationType">
              <StringClause stringPredicate="Equal">EquivalentTo</StringClause>
            </SimpleClause>
          </Clause>
        </AssociationFilter>
        <AssociationQuery>
          <AssociationFilter>
            <Clause>
              <SimpleClause leftArgument="associationType">
                <StringClause stringPredicate="StartsWith">Sin</StringClause>
              </SimpleClause>
            </Clause>
          </AssociationFilter>
        </AssociationQuery>
      </SourceAssociationBranch>
      <AssociationFilter>
```

```
        <Clause>
          <SimpleClause leftArgument="associationType">
            <StringClause stringPredicate="StartsWith">Son</StringClause>
          </SimpleClause>
        </Clause>
      </AssociationFilter>
    </AssociationQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.5 AuditableEventQuery

Purpose

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

ebRIM Binding

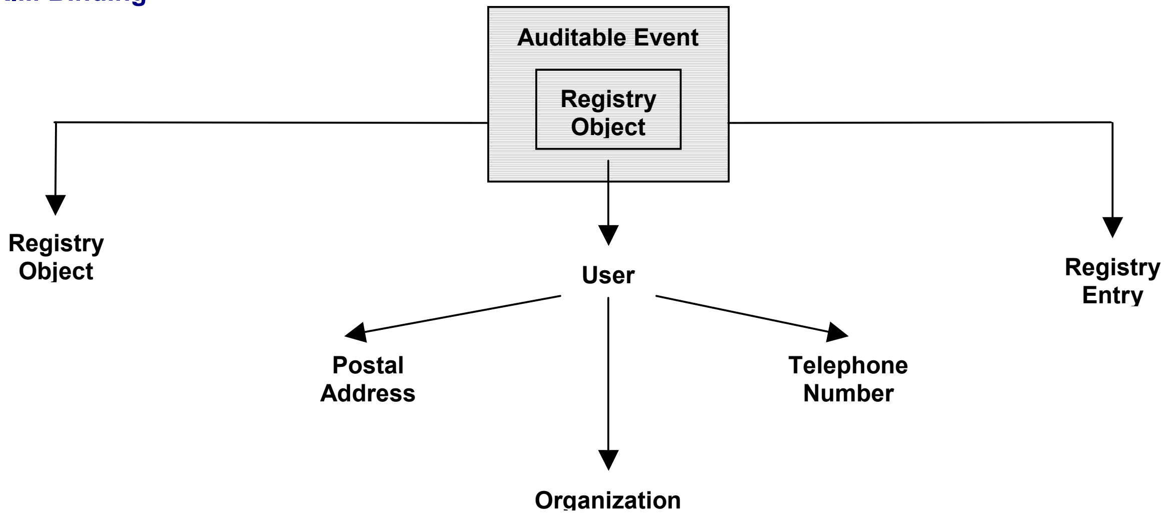

Figure 19: ebRIM Binding for AuditableEventQuery

Definition

```

<complexType name="AuditableEventQueryType">
  <complexContent>
    <extension base="tns:RegistryObjectQueryType">
      <sequence>
        <element ref="tns:AuditableEventFilter" minOccurs="0" />
        <element ref="tns:RegistryObjectQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:RegistryEntryQuery" minOccurs="0" maxOccurs="1" />
        <element ref="tns:UserBranch" minOccurs="0" maxOccurs="1" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="AuditableEventQuery" type="tns:AuditableEventQueryType" />

<element name="AuditableEventQueryResult">
  <complexType>
```

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

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 AE is empty then continue to the next numbered rule.

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

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

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

   e) If a UserBranch element is not specified then go to the next step; 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 UserBranch, and if u does not satisfy that filter, then remove x from AE. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove x from AE. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilters isn't satisfied by some of the email addresses of u then remove x from AE. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x from AE. If AE is empty then continue to the next numbered rule.

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

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

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

Examples

A Registry client has registered an item and it has been assigned a name "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 instances for all such events.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <AuditableEventQuery>
      <AuditableEventFilter>
        <Clause>
          <SimpleClause leftArgument = "timestamp">
            <RationalClause logicalPredicate = "GE">
              DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
            </RationalClause>
          </SimpleClause>
        </Clause>
      </AuditableEventFilter>
      <RegistryEntryQuery>
        <NameBranch>
          <LocalizedStringFilter>
            <Clause>
              <SimpleClause leftArgument = "value">
                <StringClause stringPredicate = "Equal">urn:path:myitem</StringClause>
              </SimpleClause>
            </Clause>
          </LocalizedStringFilter>
        </NameBranch>
      </RegistryEntryQuery>
    </AuditableEventQuery>
  </FilterQuery>
</AdhocQueryRequest
```

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”.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <AuditableEventQuery>
      <RegistryEntryQuery>
        <TargetAssociationBranch>
          <AssociationFilter>
            <Clause>
              <SimpleClause leftArgument = "associationType">
                <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
              </SimpleClause>
            </Clause>
          </AssociationFilter>
          <OrganizationQuery>
            <NameBranch>
              <LocalizedStringFilter>
```

```
                <Clause>
                 <SimpleClause leftArgument = "value">
                 <StringClause stringPredicate = "Equal">myorg</StringClause>
                 </SimpleClause>
                </Clause>
              </LocalizedStringFilter>
            </NameBranch>
          </OrganizationQuery>
        </TargetAssociationBranch>
      </RegistryEntryQuery>
      <UserBranch>
        <OrganizationQuery>
          <NameBranch>
            <LocalizedStringFilter>
              <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "-Equal">myorg</StringClause>
                </SimpleClause>
              </Clause>
            </LocalizedStringFilter>
          </NameBranch>
        </OrganizationQuery>
      </UserBranch>
    </AuditableEventQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

8.2.6 ClassificationQuery

Purpose

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

ebRIM Binding

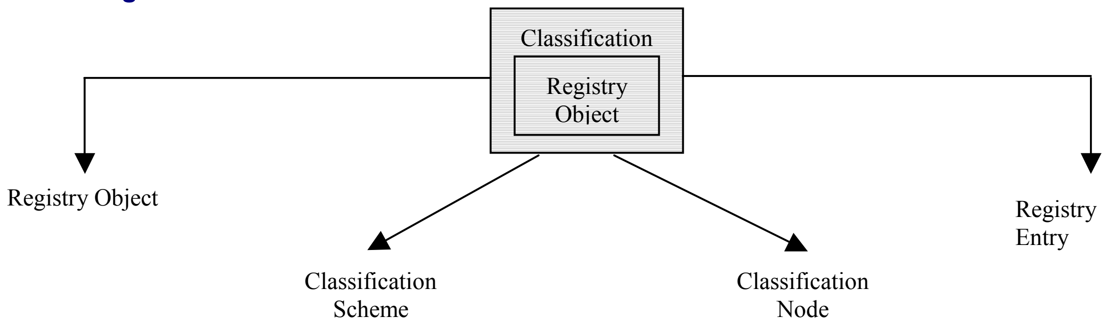

Figure 20: ebRIM Binding for ClassificationQuery

Definition

```
<complexType name = "ClassificationQueryType">
  <complexContent>
    <extension base = "tns:RegistryObjectQueryType">
      <sequence>
        <element ref = "tns:ClassificationFilter" minOccurs = "0" maxOccurs="1"/>
```

```
            <element ref = "tns:ClassificationSchemeQuery" minOccurs = "0" maxOccurs="1"/>
            <element ref = "tns:ClassificationNodeQuery" minOccurs = "0" maxOccurs="1"/>
            <element ref = "tns:RegistryObjectQuery" minOccurs = "0" maxOccurs="1"/>
            <element ref = "tns:RegistryEntryQuery" minOccurs = "0" maxOccurs="1"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
<element name = "ClassificationQuery" type = "tns:ClassificationQueryType"/>

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

Semantic Rules

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

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

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

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

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

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

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

2. If C is empty, then raise the warning: *classification query result is empty*; otherwise otherwise, set C to be the result of the ClassificationQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

8.2.7 ClassificationNodeQuery

Purpose

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

ebRIM Binding

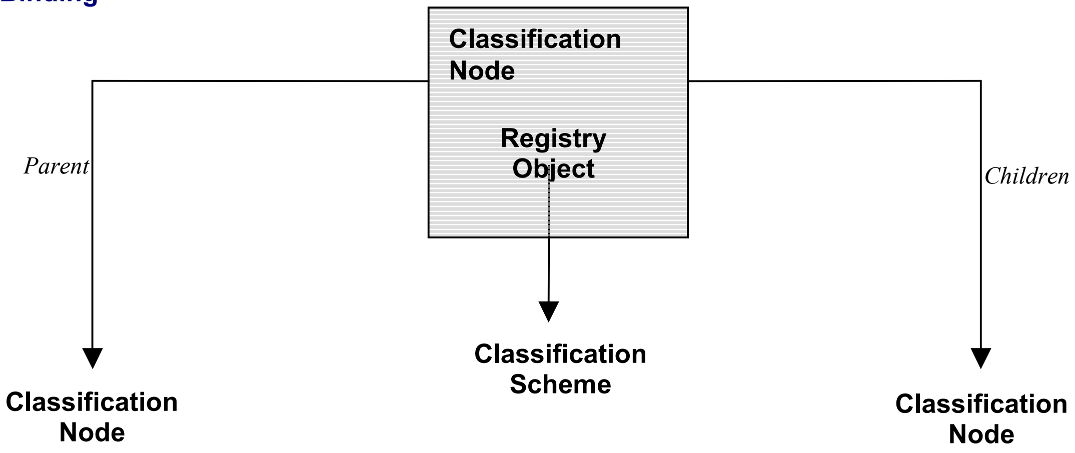

Figure 21: ebRIM Binding for ClassificationNodeQuery

Definition

```

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

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

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 CN is empty then continue to the next numbered rule.

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

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

   d) If a ClassificationNodeParentBranch element is not specified, then go to the next step; 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 go to the next step; otherwise, let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in the ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter, then remove x from CN. If CN is empty then continue to the next numbered rule. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeParentBranch and if defining classification scheme of p does not satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue to the next numbered rule.

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

   e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next numbered rule; 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 and if CN is empty continue to the next numbered rule; otherwise, treat each ClassificationNodeChildrenBranch element separately and execute the following paragraph with n = x.

Let n be a classification node instance. If a ClassificationNodeFilter element is not specified within the ClassificationNodeChildrenBranch 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 and if CN is empty continue to the next numbered rule; otherwise, let c be any member of CNC. If a ClassificationSchemeQuery element is directly contained in the ClassificationNodeChildrenBranch and if the defining classification scheme of c does not satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then remove x from CN. If CN is empty then continue to the next numbered rule; otherwise, let y be an element of CNC and continue with the next paragraph.

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

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

2. If CN is empty, then raise the warning: ***classification node query result is empty***; otherwise set CN to be the result of the ClassificationNodeQuery.
3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

Path Filter Expression usage in ClassificationNodeFilter

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

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

The path filter expression syntax includes support for matching multiple nodes by using wild card syntax as follows:

- Use of '*' as a wildcard in place of any path element in the pathFilter
- Use of '//' syntax to denote any descendent of a node in the pathFilter

It is defined by the following BNF grammar:

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

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

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

Use Cases and Examples of Path Filter Expressions

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

Table 10: Path Filter Expressions for Use Cases

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

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 name of the underlying classification scheme is “urn:ebxml:cs:myscheme”. The following query identifies all nodes at the first three levels.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ClassificationNodeQuery>
      <ClassificationNodeFilter>
        <Clause>
          <SimpleClause leftArgument = "levelNumber">
            <RationalClause logicalPredicate = "LE">
              <IntClause>3</IntClause>
            </RationalClause>
          </SimpleClause>
        </Clause>
      </ClassificationNodeFilter>
      <ClassificationSchemeQuery>
        <NameBranch>
          <LocalizedStringFilter>
            <Clause>
                <SimpleClause leftArgument = "value">
                  <StringClause stringPredicate = "Equal">urn:ebxml:cs:myscheme</StringClause>
                </SimpleClause>
            </Clause>
          </LocalizedStringFilter>
        </NameBranch>
      </ClassificationSchemeQuery>
    </ClassificationNodeQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

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

The following query finds all children nodes of a first level node whose code is NorthAmerica.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass"/>
  <FilterQuery>
    <ClassificationNodeQuery>
      <ClassificationNodeFilter>
       <Clause>
        <SimpleClause leftArgument = "path">
          <StringClause stringPredicate = "Equal">/Geography-id/NorthAmerica/*</StringClause>
        </SimpleClause>
       </Clause>
      </ClassificationNodeFilter>
    </ClassificationNodeQuery>
  </FilterQuery>
</AdhocQueryRequest>
```

The following query finds all third level nodes with code of Tokyo.

```
<AdhocQueryRequest>
  <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
  <FilterQuery>
```