

## 4 8. Object(?) Query Management Service

5  
6 NOTE to EDITOR: Some changes will be needed to this introduction to reflect the current status of  
7 Section 8.1, "Browse and Drill Down Query Support", and Section 8.3, "SQL Query Support". This  
8 proposal deals only with Section 8.2, "Filter Query Support".  
9

### 10 8.1 Browse and Drill Down Query Support

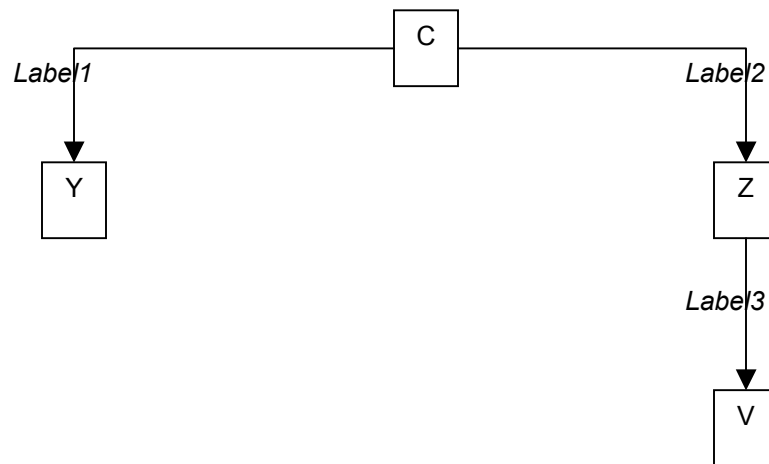
11  
12 NOTE to EDITOR: May be deleted in favor of Filter Query. That proposed action is independent of this  
13 proposal.

### 14 8.2 Filter Query Support

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

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

26 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes  
27 derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a  
28 class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a  
29 class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated  
30 with Z. The ebRIM Binding for C might be as in Figure 1.  
31



32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
Figure 1: Example ebRIM Binding

49  
50 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and Label3  
51 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to which ebRIM  
52 association is intended. The name of the query is determined by the root class, i.e. this is an ebRIM  
53 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  
54 association identified by Label1. Similarly, the Z and V nodes are limited to instances that are linked to  
55 their parent node by the identified association.

56 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a restricted  
57 *predicate clause* over the attributes of a single class. The supported class filters are specified in Section  
58 8.2.9 and the supported predicate clauses are defined in Section **Error! Reference source not found.**  
59 A FilterQuery will be composed of elements that traverse the tree to determine which branches satisfy the  
60 designated class filters, and the query result will be the set of root node instances that support such a  
61 branch.

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

69 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  
70 element on Y is allowed. However, if the association is one-to-many, then multiple filter or branch  
71 elements are allowed. This allows one to specify that an instance of C must have associations with  
72 multiple instances of Y before the instance of C is said to satisfy the branch element.

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

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

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

82 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a way to structure an  
83 XML document as an expansion of the result of a RegistryEntryQuery. The ReturnRegistryEntry element  
84 specified in Section 8.2.7 allows one to specify what metadata one wants returned with each registry  
85 entry identified in the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section  
86 8.2.8 allows one to specify what repository items one wants returned based on their relationships to the  
87 registry entries identified by the result of a RegistryEntryQuery.  
88

## 88 8.2.1 FilterQuery

### 89 Purpose

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

### 94 Definition

```
95
96 <!ELEMENT FilterQuery
97   ( RegistryEntryQuery
98     | AuditableEventQuery
99     | ClassificationNodeQuery
100    | RegistryPackageQuery
101    | OrganizationQuery )>
102
103 <!ELEMENT FilterQueryResult
104   ( RegistryEntryQueryResult
105     | AuditableEventQueryResult
106     | ClassificationNodeQueryResult
107     | RegistryPackageQueryResult
108     | OrganizationQueryResult )>
109
110 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
111
112 <!ELEMENT RegistryEntryView EMPTY >
113 <!ATTLIST RegistryEntryView
114   id          CDATA      #REQUIRED
115   name        CDATA      #REQUIRED
116   contentURI  CDATA      #IMPLIED >
117
118 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
119
120 <!ELEMENT AuditableEventView EMPTY >
121 <!ATTLIST AuditableEventView
122   id          CDATA      #REQUIRED
123   name        CDATA      #REQUIRED
124   timestamp   CDATA      #REQUIRED >
125
126 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
127
128 <!ELEMENT ClassificationNodeView EMPTY >
129 <!ATTLIST ClassificationNodeView
130   id          CDATA      #REQUIRED
131   name        CDATA      #REQUIRED
132   code        CDATA      #REQUIRED >
133
134 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
135
136 <!ELEMENT RegistryPackageView EMPTY >
137 <!ATTLIST RegistryPackageView
138   id          CDATA      #REQUIRED
139   name        CDATA      #REQUIRED >
140
141 <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
```

```
142
143 <!ELEMENT OrganizationView EMPTY >
144 <!ATTLIST OrganizationView
145     id          CDATA          #REQUIRED
146     name        CDATA          #REQUIRED >
147
148
```

#### 149 **Semantic Rules**

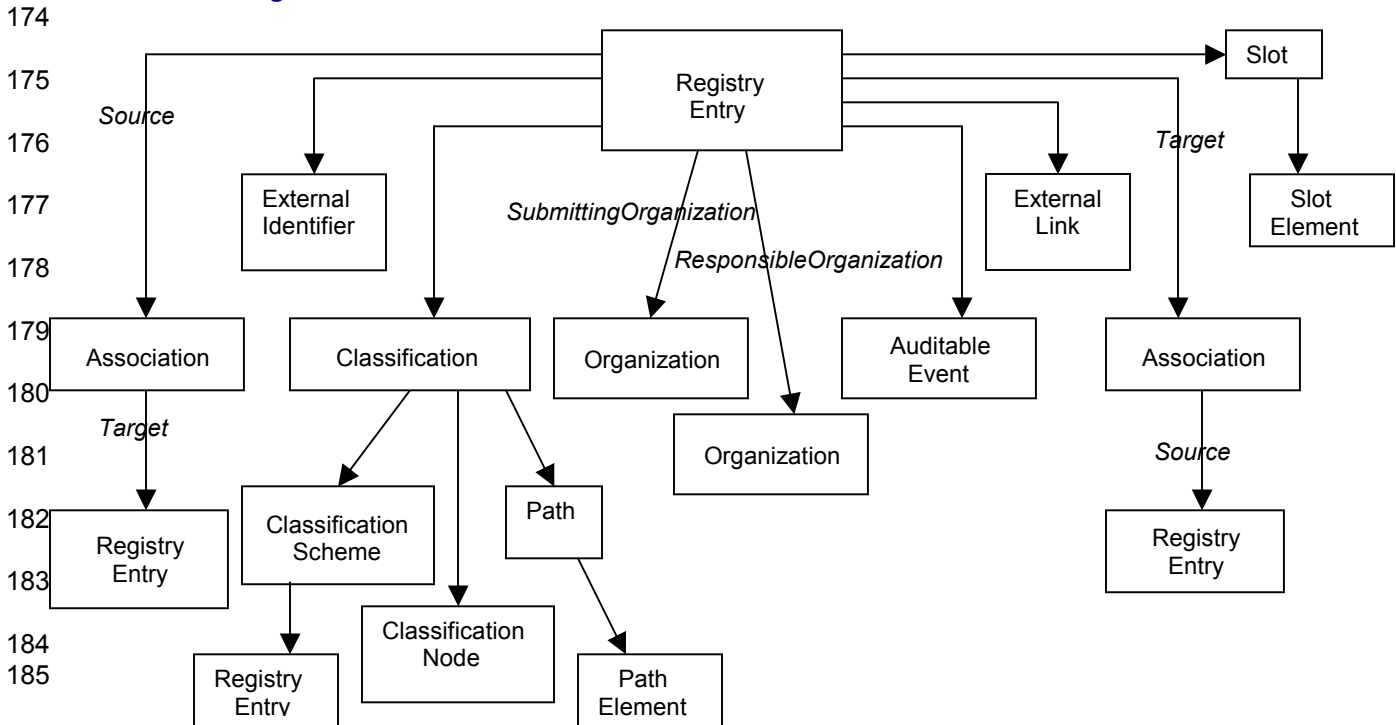
- 150 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 151 2. Each FilterQueryResult is a set of XML elements to identify each instance of the result set. Each XML  
152 attribute carries a value derived from the value of an attribute specified in the Registry Information  
153 Model as follows:
  - 154 a) id carries the value of the id attribute of the RegistryObject class,
  - 155 b) name carries the value of the name attribute of the RegistryObjectClass,
  - 156 c) contentURI, if present, carries the value of the contentURI attribute of the ExtrinsicObject class,
  - 157 d) timestamp carries a character string literal value to represent the value of the timestamp attribute  
158 of the AuditableEvent class,
  - 159 e) code carries the value of the code attribute of the ClassificationNode class.
- 160 3. If an error condition is raised during any part of the execution of a FilterQuery, then the status  
161 attribute of the XML RegistryResult is set to “failure” and no query result element is returned; instead,  
162 a RegistryErrorList element must be returned with its highestSeverity element set to “error”. At least  
163 one of the RegistryError elements in the RegistryErrorList will have its severity attribute set to “error”.
- 164 4. If no error conditions are raised during execution of a FilterQuery, then the status attribute of the XML  
165 RegistryResult is set to “success” and an appropriate query result element must be included. If a  
166 RegistryErrorList is also returned, then the highestSeverity attribute of the RegistryErrorList is set to  
167 “warning” and the serverity attribute of each RegistryError is set to “warning”.  
168  
169  
170

170 **8.2.2 RegistryEntryQuery**

171 **Purpose**

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

173 **ebRIM Binding**



187 **Definition**

```

188
189 <!ELEMENT RegistryEntryQuery
190   ( RegistryEntryFilter?,
191     SourceAssociationBranch*,
192     TargetAssociationBranch*,
193     HasClassificationBranch*,
194     SubmittingOrganizationBranch?,
195     ResponsibleOrganizationBranch?,
196     ExternalIdentifierBranch*,
197     ExternalLinkBranch*,
198     HasSlotBranch*,
199     HasAuditableEventBranch*      )>
200
201 <!ELEMENT SourceAssociationBranch
202   ( AssociationFilter?,
203     ( RegistryEntryFilter? | RegistryEntryQuery? ) )>
204
205 <!ELEMENT TargetAssociationBranch
206   ( AssociationFilter?,
207     ( RegistryEntryFilter? | RegistryEntryQuery? ) )>
208
209 <!ELEMENT HasClassificationBranch
210   ( ClassificationFilter?,
  
```

```

211         FromSchemeBranch?,
212         HasPathBranch?,
213         LocalNodeBranch?,
214         SubmittingOrganizationBranch?     )>
215
216     <!ELEMENT FromSchemeBranch
217     ( ClassificationSchemeFilter | RegistryEntryQuery )>
218
219     <!ELEMENT HasPathBranch
220     ( PathFilter | XpathNodeExpression | PathElementFilter+ )>
221
222     <!ELEMENT XpathNodeExpression ( TO BE DETERMINED )>
223

```

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

```

233
234     <!ELEMENT LocalNodeBranch
235     ( ClassificationNodeFilter? | ClassificationNodeQuery? )>
236
237     <!ELEMENT SubmittingOrganizationBranch
238     ( OrganizationFilter | OrganizationQuery )>
239
240     <!ELEMENT ResponsibleOrganizationBranch
241     ( OrganizationFilter? | OrganizationQuery? )>
242
243     <!ELEMENT ExternalIdentifierBranch
244     ( ExternalIdentifierFilter?,
245       SubmittingOrganizationBranch? )>
246
247     <!ELEMENT ExternalLinkBranch
248     ( ExternalLinkFilter )>
249
250     <!ELEMENT HasSlotBranch
251     ( SlotFilter?,
252       SlotElementFilter* )>
253
254     <!ELEMENT HasAuditableEventBranch
255     ( AuditableEventFilter? | AuditableEventQuery? )>
256
257

```

## 258 Semantic Rules

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

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

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

288 d) If a HasClassificationBranch element is not specified, or if RE is empty, then continue below;  
289 otherwise, let x be a remaining registry entry in RE. If x is not the classifiedObject of some  
290 Classification instance, then remove x from RE; otherwise, treat each HasClassificationBranch  
291 element separately as follows:  
292 If no ClassificationFilter is specified within the HasClassificationBranch, then let CL be the set of  
293 all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of  
294 Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If  
295 CL is empty, then remove x from RE and continue below. Otherwise, if CL is not empty, and if a  
296 FromSchemeBranch is specified, then replace CL by the set of remaining Classification instances  
297 in CL whose defining classification scheme satisfies the ClassificationSchemeFilter or the  
298 RegistryEntryQuery immediately contained in the FromSchemeBranch. If the new CL is empty,  
299 then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a  
300 HasPathBranch is specified, then replace CL by the set of remaining Classification instances in  
301 CL that satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter  
302 elements immediately contained in the HasPathBranch. If the new CL is empty, then remove x  
303 from RE and continue below. Otherwise, if CL remains not empty, and if a LocalNodeBranch is  
304 specified, then replace CL by the set of remaining Classification instances in CL for which a local  
305 node exists and for which that local node satisfies the ClassificationNodeFilter or the  
306 ClassificationNodeQuery immediately contained in the LocalNodeBranch. . If the new CL is  
307 empty, then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a  
308 SubmittingOrganizationBranch is specified, then replace CL by the set of remaining Classification  
309 instances in CL for which the submitting organization of that classification satisfies the  
310 OrganizationFilter or OrganizationQuery immediately contained in the  
311 SubmittingOrganizationBranch. If the new CL is empty, then remove x from RE.

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

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

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

325 separately as follows: If an ExternalIdentifierFilter is not specified, then let EI be the set of  
326 ExternalIdentifier instances that are linked to x; otherwise, let EI be the set of ExternalIdentifier  
327 instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x  
328 from RE and continue below. Otherwise, if EI remains not empty, and if a  
329 SubmittingOrganizationBranch is specified, replace EI by the set of remaining ExternalIdentifier  
330 instances in EI for which the OrganizationFilter or OrganizationQuery immediately contained in  
331 the SubmittingOrganizationBranch is valid. If the new EI is empty, then remove x from RE.  
332 h) If an ExternalLinkBranch element is not specified, or if RE is empty, then continue below;  
333 otherwise, let x be a remaining registry entry in RE. If x is not linked to some ExternalLink  
334 instance, then remove x from RE; otherwise, treat each ExternalLinkBranch element separately  
335 as follows: Let EL be the set of ExternalLink instances that satisfy the ExternalLinkFilter directly  
336 contained in the ExternalLinkBranch and are linked to x. If EL is empty, then remove x from RE.  
337 i) If a HasSlotBranch element is not specified, or if RE is empty, then continue below; otherwise, let  
338 x be a remaining registry entry in RE. If x is not linked to some Slot instance, then remove x from  
339 RE and continue below; otherwise, treat each HasSlotBranch element separately as follows: If a  
340 SlotFilter is not specified within HasSlotBranch, then let SL be the set of all Slot instances for x;  
341 otherwise, let SL be the set of Slot instances that satisfy the SlotFilter and are Slot instances for  
342 x. If SL is empty, then remove x from RE and continue below. Otherwise, if SL remains not  
343 empty, and if a SlotElementFilter is specified, replace SL by the set of remaining Slot instances in  
344 SL for which every specified SlotElementFilter is valid. If SL is empty, then remove x from RE.  
345 j) If a HasAuditableEventBranch element is not specified, or if RE is empty, then continue below;  
346 otherwise, let x be a remaining registry entry in RE. If x is not linked to some AuditableEvent  
347 instance, then remove x from RE; otherwise, treat each HasAuditableEventBranch element  
348 separately as follows: If an AuditableEventFilter or AuditableEventQuery is not specified within  
349 HasAuditableEventBranch, then let AE be the set of all AuditableEvent instances for x; otherwise,  
350 let AE be the set of AuditableEvent instances that satisfy the AuditableEventFilter or  
351 AuditableEventQuery and are auditable events for x. If AE is empty, then remove x from RE.  
352 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return RE as  
353 the result of the RegistryEntryQuery.  
354 3. Return any accumulated warnings or exceptions as the StatusResult associated with the  
355 RegistryEntryQuery.

356

### 357 **Examples**

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

```

363
364 <RegistryEntryQuery>
365   <RegistryEntryFilter>
366     status EQUAL "Approved"           -- code by Clause, Section Error! Reference
367                                     source not found.
368   </RegistryEntryFilter>
369   <SubmittingOrganizationBranch>
370     <OrganizationFilter>
371       name CONTAINS "XYZ"           -- code by Clause, Section Error! Reference
372                                     source not found.
373     </OrganizationFilter>
374   </SubmittingOrganizationBranch>
375 </RegistryEntryQuery>
376
```

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



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

```

384
385 <RegistryEntryQuery>
386   <RegistryEntryFilter>
387     objectType EQUAL "CPP" AND -- code by Clause, Section Error! Reference
388                               source not found.
389     status EQUAL "Approved"
390   </RegistryEntryFilter>
391   <HasClassificationBranch>
392     <FromSchemeBranch>
393       <ClassificationSchemeFilter>
394         id EQUAL "urn:org:un:spsc:cs2001" -- code by Clause, Section Error!
395                                           Reference source not found.
396       </ClassificationSchemeFilter>
397     </FromSchemeBranch>
398     <HasPathBranch>
399       <PathFilter>
400         code EQUAL "321118"
401       </PathFilter>
402     </HasPathBranch>
403   </HasClassificationBranch>
404 </RegistryEntryQuery>

```

405  
 406 A client application needs all items that are classified by two different classification schemes, one based  
 407 on "Industry" and another based on "Geography". Both schemes have been defined by ebXML and are  
 408 registered as "urn:ebxml:cs:industry" and "urn:ebxml:cs:geography", respectively. The following query  
 409 identifies registry entries for all registered items that are classified by Industry as any subnode of  
 410 "Automotive" and by Geography as any subnode of "Asia/Japan".

```

411
412 <RegistryEntryQuery>
413   <HasClassificationBranch>
414     <FromSchemeBranch>
415       <ClassificationSchemeFilter>
416         id EQUAL "urn:ebxml:cs:industry" -- code by Clause, Section Error!
417                                           Reference source not found.
418       </ClassificationSchemeFilter>
419     </FromSchemeBranch>
420     <HasPathBranch>
421       <XpathExpression>
422         getPath = "//Automotive"
423       </XpathExpression>
424     </HasPathExpression>
425   </HasClassificationBranch>
426   <HasClassificationBranch>
427     <FromSchemeBranch>
428       <ClassificationSchemeFilter>
429         id EQUAL "urn:ebxml:cs:geography" -- code by Clause, Section Error!
430                                           Reference source not found.
431       </ClassificationSchemeFilter>
432     </FromSchemeBranch>
433     <HasPathBranch>
434       <PathFilter>
435         path STARTSWITH "/Asia/Japan"
436       </PathFilter>

```

```

437     </HasPathBranch>
438     </HasClassificationBranch>
439 </RegistryEntryQuery>

```

440  
441 A client application wishes to identify all registry Package instances that have a given registry entry as a  
442 member of the package. The following query identifies all registry packages that contain the registry entry  
443 identified by URN "urn:path:myitem" as a member:

```

444 <RegistryEntryQuery>
445   <RegistryEntryFilter>
446     objectType EQUAL "RegistryPackage"      -- code by Clause, Section Error!
447                                           Reference source not found.
448   </RegistryEntryFilter>
449   <SourceAssociationBranch>
450     <AssociationFilter>
451       associationType EQUAL "HasMember"    -- code by Clause, Section Error!
452                                           Reference source not found.
453     </AssociationFilter>
454     <RegistryEntryFilter>
455       id EQUAL "urn:path:myitem"         -- code by Clause, Section Error! Reference
456                                           source not found.
457     </RegistryEntryFilter>
458   </SourceAssociationBranch>
459 </RegistryEntryQuery>

```

460  
461 A client application wishes to identify all RegistryEntry instances that are classified by some internal  
462 classification scheme and have some given keyword as part of the name or description of one of the  
463 classification nodes of that classification scheme. The following query identifies all such RegistryEntry  
464 instances. The query takes advantage of the knowledge that the classification scheme is internal, and  
465 thus that all of its nodes are fully described as ClassificationNode instances.

```

466 <RegistryEntryQuery>
467   <HasClassificationBranch>
468     <LocalNodeBranch>
469       <ClassificationNodeFilter>
470         name CONTAINS "transistor" OR    -- code by Clause, Section Error!
471                                           Reference source not found.
472         description CONTAINS "transistor"
473       </ClassificationNodeFilter>
474     </LocalNodeBranch>
475   </HasClassificationBranch>
476 </RegistryEntryQuery>

```

```

480
481

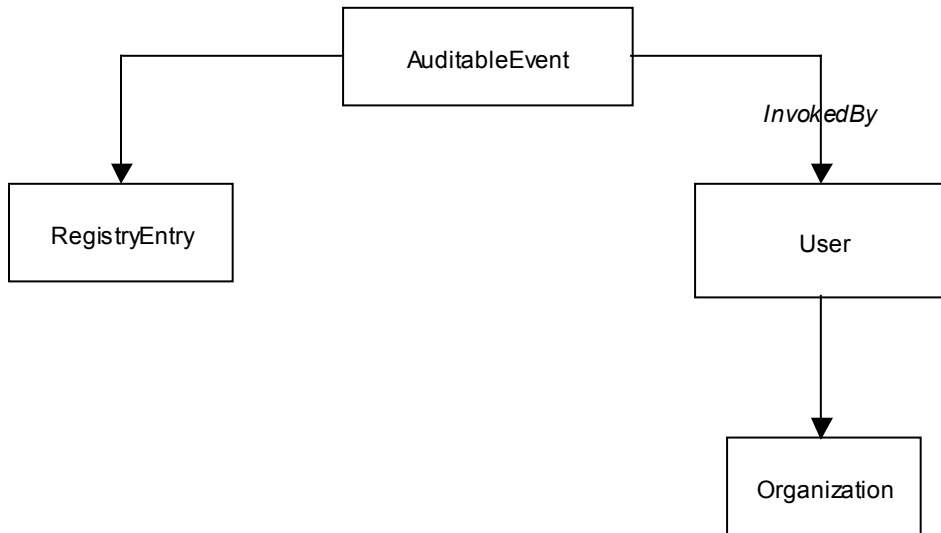
```

481 **8.2.3 AuditableEventQuery**

482 **Purpose**

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

485 **Definition**



```
486
487 <!ELEMENT AuditableEventQuery
488   ( AuditableEventFilter?,
489     RegistryEntryQuery*,
490     InvokedByBranch? )>
491
492 <!ELEMENT InvokedByBranch
493   ( UserFilter?,
494     OrganizationQuery? )>
495
```

496 **Semantic Rules**

- 497 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The following steps  
498 will eliminate instances in AE that do not satisfy the conditions of the specified filters.  
499
- 500 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below; otherwise, let x  
501 be an auditable event in AE. If x does not satisfy the AuditableEventFilter as defined in Section  
502 8.2.9, then remove x from AE.
  - 503 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue below;  
504 otherwise, let x be a remaining auditable event in AE. Treat each RegistryEntryQuery element  
505 separately as follows:  
506 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If x is not an  
507 auditable event for some registry entry in RE, then remove x from AE.
  - 508 c) If an InvokedByBranch element is not specified, or if AE is empty, then continue below; otherwise,  
509 let x be a remaining auditable event in AE.
- 510 Let u be the user instance that invokes x. If a UserFilter element is specified within the InvokedByBranch,  
511 and if u does not satisfy that filter, then remove x from AE; otherwise, continue below.

512 If an OrganizationQuery element is not specified within the InvokedByBranch, then continue  
 513 below; otherwise, let OG be the set of Organization instances that are identified by the  
 514 organization attribute of u and are in the result set of the OrganizationQuery. If OG is empty, then  
 515 remove x from AE.  
 516 2. If AE is empty, then raise the warning: *auditable event query result is empty*.  
 517 3. Return AE as the result of the AuditableEventQuery.  
 518

519 **Examples**

520 A Registry client has registered an item and it has been assigned a URN identifier "urn:path:myitem".  
 521 The client is now interested in all events since the beginning of the year that have impacted that item. The  
 522 following query will return a set of AuditableEvent identifiers for all such events.  
 523

```
524 <AuditableEventquery>
525   <AuditableEventFilter>
526     timestamp GE "2001-01-01" AND
527     registryEntry EQUAL "urn:path:myitem"
528   </AuditableEventFilter>
529 </AuditableEventQuery>
```

-- code by Clause, Section **Error!**  
**Reference source not found.**

531 A client company has many registered objects in the Registry. The Registry allows events submitted by  
 532 other organizations to have an impact on your registered items, e.g. new classifications and new  
 533 associations. The following query will return a set of identifiers for all auditable events, invoked by some  
 534 other party, that had an impact on an item submitted by "myorg" and for which "myorg" is the responsible  
 535 organization.  
 536  
 537

```
538 <AuditableEventQuery>
539   <RegistryEntryQuery>
540     <SubmittingOrganizationBranch>
541       <OrganizationFilter>
542         id EQUAL "urn:somepath:myorg"
543       </OrganizationFilter>
544     </SubmittingOrganizationBranch>
545     <ResponsibleOrganizationBranch>
546       <OrganizationFilter>
547         id EQUAL "urn:somepath:myorg"
548       </OrganizationFilter>
549     </ResponsibleOrganizationBranch>
550   </RegistryEntryQuery>
551 <InvokedByBranch>
552   <OrganizationQuery>
553     <OrganizationFilter>
554       id -EQUAL "urn:somepath:myorg"
555     </OrganizationFilter>
556   </OrganizationQuery>
557 </InvokedByBranch>
558 </AuditableEventQuery>
```

-- code by Clause, Section **Error!**  
**Reference source not found.**

-- code by Clause, Section **Error!**  
**Reference source not found.**

-- code by Clause, Section **Error!**  
**Reference source not found.**

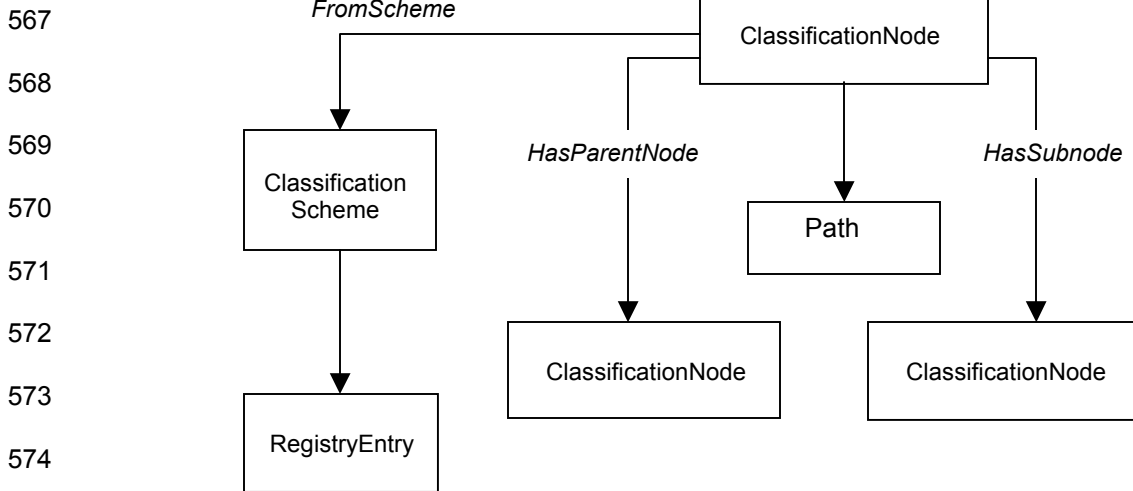
562 **8.2.4 ClassificationNodeQuery**

563 **Purpose**

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

565 **ebRIM Binding**

566



575

576

577 **Definition**

578

```
579 <!ELEMENT ClassificationNodeQuery
580 ( ClassificationNodeFilter?,
581 FromSchemeBranch?,
582 HasPathBranch?,
583 HasParentNodeBranch?,
584 HasSubnodeBranch* )>
```

585

```
586 <!ELEMENT HasParentNodeBranch
587 ( ClassificationNodeFilter?,
588 HasPathBranch?,
589 HasParentNodeBranch? )>
```

590

```
591 <!ELEMENT HasSubnodeBranch
592 ( ClassificationNodeFilter?,
593 HasPathBranch?,
594 HasSubnodeBranch* )>
```

595

596 **Semantic Rules**

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

- 602 b) If a FromSchemeBranch is not specified, or if CN is empty, then continue below; otherwise, let x  
603 be a remaining classification node in CN. If the defining classification scheme of x does not  
604 satisfy the ClassificationSchemeFilter or the RegistryEntryQuery immediately contained in the  
605 FromSchemeBranch, then remove x from CN.
- 606 c) If a HasPathBranch is not specified, or if CN is empty, then continue below; otherwise, let x be a  
607 remaining classification node in CN. If the path derived from x does not satisfy the PathFilter, the  
608 XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in  
609 the HasPathBranch, then remove x from CN.
- 610 d) If a HasParentNodeBranch element is not specified, or if CN is empty, then continue below;  
611 otherwise, let x be a remaining classification node in CN and execute the following paragraph  
612 with n=x.

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

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

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

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

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

- 638 2. If CN is empty, then raise the warning: *classification node query result is empty*.  
639 3. Return CN as the result of the ClassificationNodeQuery.  
640

## 641 Examples

642  
643 A client application wishes to identify all of the classification nodes in the first three levels of a  
644 classification scheme hierarchy. The client knows that the URN identifier for the underlying classification  
645 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three levels.  
646

```

647 <ClassificationNodeQuery>
648   <FromSchemeBranch>
649     <ClassificationSchemeFilter>
650       id EQUAL "urn:ebxml:cs:myscheme" -- code by Clause, Section Error!
651                                     Reference source not found.
652     </ClassificationSchemeFilter>
653   </FromSchemeBranch>
654   <HasPathBranch>
655     <PathFilter>
```

```
656         pathDepth LE "3"
657     </PathFilter>
658 </HasPathBranch>
659 </ClassificationNodeQuery>
```

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

662  
663 By assuming that the "path" of a node is known, and the URN of the classification scheme it comes from,  
664 one could get all nodes at the next level below that node as follows:

```
665  
666 <ClassificationNodeQuery>
667     <FromSchemeBranch>
668         <ClassificationSchemeFilter>
669             id EQUAL "urn:some:known:scheme"
670         </ClassificationSchemeFilter>
671     </FromSchemeBranch>
672     <HasParentBranch>
673         <HasPathBranch>
674             <PathFilter>
675                 path EQUAL "KnownPathOfGivenNode"
676             </PathFilter>
677         </HasPathBranch>
678     </HasParentBranch>
679 </ClassificationNodeQuery>
```

680  
681 If instead, one wanted ALL nodes in the subtree beneath the given node, then the following query could  
682 be used:

```
683  
684 <ClassificationNodeQuery>
685     <FromSchemeBranch>
686         <ClassificationSchemeFilter>
687             id EQUAL "urn:some:known:scheme"
688         </ClassificationSchemeFilter>
689     </FromSchemeBranch>
690     <HasParentBranch>
691         <HasPathBranch>
692             <PathFilter>
693                 path STARTSWITH "KnownPathOfGivenNode"
694             </PathFilter>
695         </HasPathBranch>
696     </HasParentBranch>
697 </ClassificationNodeQuery>
```

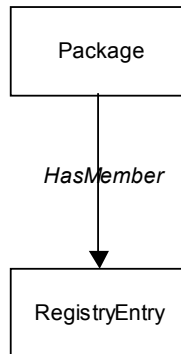
698  
699

699 **8.2.5 RegistryPackageQuery**

700 **Purpose**

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

702 **ebRIM Binding**



703 **Definition**

```
704
705 <!ELEMENT RegistryPackageQuery
706 ( PackageFilter?,
707 HasMemberBranch* )>
708
709 <!ELEMENT HasMemberBranch
710 ( RegistryEntryQuery? )>
```

711

712 **Semantic Rules**

- 713 1. Let RP denote the set of all persistent Package instances in the Registry. The following steps will  
714 eliminate instances in RP that do not satisfy the conditions of the specified filters.
    - 715 a) If a PackageFilter is not specified, or if RP is empty, then continue below; otherwise, let x be a  
716 package instance in RP. If x does not satisfy the PackageFilter as defined in Section 8.2.9, then  
717 remove x from RP.
    - 718 b) If a HasMemberBranch element is not directly contained in the RegistryPackageQuery, or if RP is  
719 empty, then continue below; otherwise, let x be a remaining package instance in RP. If x is an  
720 empty package, then remove x from RP; otherwise, treat each HasMemberBranch element  
721 separately as follows:  
722  
723 If a RegistryEntryQuery element is not directly contained in the HasMemberBranch element, then  
724 let PM be the set of all RegistryEntry instances that are members of the package x; otherwise, let  
725 RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in  
726 Section 8.2.2 and let PM be the subset of RE that are members of the package x. If PM is empty,  
727 then remove x from RP.
  - 728 2. If RP is empty, then raise the warning: *registry package query result is empty*.
  - 729 3. Return RP as the result of the RegistryPackageQuery.
- 730

731 **Examples**

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

```
734
735 <RegistryPackageQuery>
```



```

736     <HasMemberBranch>
737         <RegistryEntryQuery>
738             <RegistryEntryFilter>
739                 objectType EQ "Invoice"      -- code by Clause, Section Error! Reference
740                                             source not found.
741             </RegistryEntryFilter>
742         </RegistryEntryQuery>
743     </HasMemberBranch>
744 </RegistryPackageQuery>
745

```

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

```

747
748     <RegistryEntryQuery>
749         <HasMemberBranch/>
750     </RegistryEntryQuery>
751

```

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

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

762

762 **8.2.6 OrganizationQuery**

763 **Purpose**

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

765 **ebRIM Binding**

766

767

768

769

770

771

772

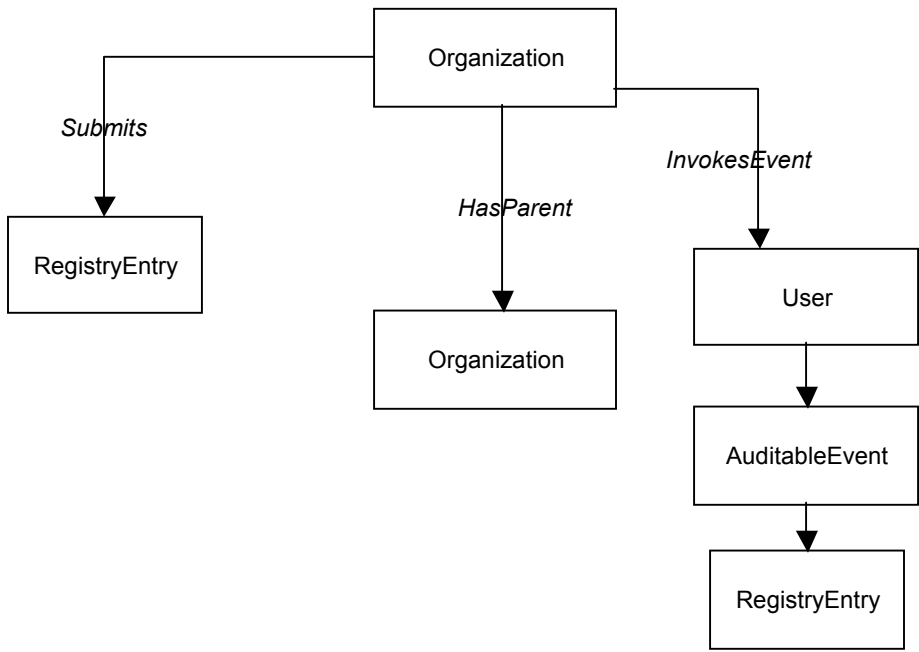
773

774

775

776

777



778 **Definition**

779

```
780 <!ELEMENT OrganizationQuery  
781 ( OrganizationFilter?,  
782 SubmitsRegistryEntry*,  
783 HasParentOrganization?,  
784 InvokesEventBranch* )>
```

```
785 <!ELEMENT SubmitsRegistryEntry ( RegistryEntryQuery? )>
```

```
786 <!ELEMENT HasParentOrganization  
787 ( OrganizationFilter?,  
788 HasParentOrganization? )>
```

```
789 <!ELEMENT InvokesEventBranch  
790 ( UserFilter?,  
791 AuditableEventFilter?,  
792 RegistryEntryQuery? )>
```

796 **Semantic Rules**

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

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

805 If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element, then let RES be  
806 the set of all RegistryEntry instances that have been submitted to the Registry by organization x;  
807 otherwise, let RE be the result of the RegistryEntryQuery as defined in Section 8.2.2 and let RES  
808 be the set of all instances in RE that have been submitted to the Registry by organization x. If  
809 RES is empty, then remove x from ORG.

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

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

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

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

821 If an UserFilter is not specified, and if x is not the submitting organization of some AuditableEvent  
822 instance, then remove x from ORG. If an AuditableEventFilter is not specified, then let AE be the  
823 set of all AuditableEvent instances that have x as the submitting organization; otherwise, let AE  
824 be the set of AuditableEvent instances that satisfy the AuditableEventFilter and have x as the  
825 submitting organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is not  
826 specified in the InvokesEventBranch element, then let RES be the set of all RegistryEntry  
827 instances associated with an event in AE; otherwise, let RE be the result set of the  
828 RegistryEntryQuery, as specified in Section 8.2.2, and let RES be the subset of RE of entries  
829 submitted by x. If RES is empty, then remove x from ORG.

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

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

832

### 833 Examples

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

836

```

837 <OrganizationQuery>
838   <OrganizationFilter>
839     country EQUAL "France"           -- code by Clause, Section Error! Reference
840                                     source not found.
841   </OrganizationFilter>
842   <SubmitsRegistryEntry>
843     <RegistryEntryQuery>
844       <RegistryEntryFilter>
845         objectType EQUAL "CPP"      -- code by Clause, Section Error! Reference
846                                     source not found.
847       </RegistryEntryFilter>
848       <HasAuditableEventBranch>
849         <AuditableEventFilter>
850           timestamp GE "2001-01-01" -- code by Clause, Section Error!
851                                     Reference source not found.
852         </AuditableEventFilter>
853       </HasAuditableEventBranch>

```

```
854         </RegistryEntryQuery>
855     </SubmitsRegistryEntry>
856 </OrganizationQuery>
```

857  
858 A client application wishes to identify all organizations that have XYZ, Corporation as a parent. The client  
859 knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is no guarantee that subsidiaries of  
860 XYZ have a URN that uses the same format, so a full query is required.

```
861
862     <OrganizationQuery>
863         <HasParentOrganization>
864             <OrganizationFilter>
865                 id EQUAL "urn:ebxml:org:xyz" -- code by Clause, Section Error! Reference
866                                     source not found.
867             </OrganizationFilter>
868         </HasParentOrganization>
869     </OrganizationQuery>
870
```

## 870 8.2.7 ReturnRegistryEntry

### 871 Purpose

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

### 875 Definition

```
876  
877 <!ELEMENT ReturnRegistryEntry  
878 ( RegistryEntryQuery,  
879 WithClassifications?,  
880 WithSourceAssociations?,  
881 WithTargetAssociations?,  
882 WithAuditableEvents?,  
883 WithExternalLinks? )>  
884  
885 <!ELEMENT WithClassifications ( ClassificationFilter? )>  
886 <!ELEMENT WithSourceAssociations ( AssociationFilter? )>  
887 <!ELEMENT WithTargetAssociations ( AssociationFilter? )>  
888 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>  
889 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>  
890  
891 <!ELEMENT ReturnRegistryEntryResult  
892 ( RegistryEntryMetadata*)>  
893  
894 <!ELEMENT RegistryEntryMetadata  
895 ( RegistryEntry,  
896 Classification*,  
897 SourceAssociations?,  
898 TargetAssociations?,  
899 AuditableEvent*,  
900 ExternalLink* )>  
901  
902 <!ELEMENT SourceAssociations ( Association* )>  
903 <!ELEMENT TargetAssociations ( Association* )>
```

### 904 Semantic Rules

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

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

- 927 7. If WithSourceAssociations is specified, then for each E in R do the following: If an AssociationFilter is  
 928 not present, then let A be any association instance whose source object is E; otherwise, let A be an  
 929 association instance that satisfies the AssociationFilter (Section 8.2.9) and whose source object is E.  
 930 For each such A, create a new Association element as defined in Appendix A. Add these Association  
 931 elements as subelements of the WithSourceAssociations and add that element to its parent  
 932 RegistryEntryMetadata element.
- 933 8. If WithTargetAssociations is specified, then for each E in R do the following: If an AssociationFilter is  
 934 not present, then let A be any association instance whose target object is E; otherwise, let A be an  
 935 association instance that satisfies the AssociationFilter (Section 8.2.9) and whose target object is E.  
 936 For each such A, create a new Association element as defined in Appendix A. Add these Association  
 937 elements as subelements of the WithTargetAssociations and add that element to its parent  
 938 RegistryEntryMetadata element.
- 939 9. If WithAuditableEvents is specified, then for each E in R do the following: If an AuditableEventFilter is  
 940 not present, then let A be any auditable event instance linked to E; otherwise, let A be any auditable  
 941 event instance linked to E that satisfies the AuditableEventFilter (Section 8.2.9). For each such A,  
 942 create a new AuditableEvent element as defined in Appendix A. Add these AuditableEvent elements  
 943 to their parent RegistryEntryMetadata element.
- 944 10. If WithExternalLinks is specified, then for each E in R do the following: If an ExternalLinkFilter is not  
 945 present, then let L be any external link instance linked to E; otherwise, let L be any external link  
 946 instance linked to E that satisfies the ExternalLinkFilter (Section 8.2.9). For each such D, create a  
 947 new ExternalLink element as defined in Appendix A. Add these ExternalLink elements to their parent  
 948 RegistryEntryMetadata element.
- 949 11. If any warning or error condition results, then add the code and the message to the  
 950 RegistryResponse element that includes the RegistryEntryQueryResult.
- 951 12. Return the set of RegistryEntryMetadata elements as the content of the ReturnRegistryEntryResult.  
 952

### 953 Examples

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

```

961 <ReturnRegistryEntry>
962   <RegistryEntryQuery>
963     <RegistryEntryFilter>
964       id EQUAL "urn:com:xyz:po:325"           -- code by Clause, Section Error!
965                                               Reference source not found.
966     </RegistryEntryFilter>
967   </RegistryEntryQuery>
968   <WithClassifications/>
969   <WithSourceAssociations>
970     <AssociationFilter>                       -- code by Clause, Section Error!
971                                               Reference source not found.
972       associationType EQUAL "SupersededBy" OR
973       associationType EQUAL "ReplacedBy"
974     </AssociationFilter>
975   </WithSourceAssociations>
976 </ReturnRegistryEntry>
977
```

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

980 completely upward compatible with the existing DTD. The client desires a list of all registered items that  
 981 use the existing DTD so they can assess the impact of an incompatible change. The following query  
 982 returns an XML document that is a list of all RegistryEntry elements that represent registered items that  
 983 use, contain, or extend the given DTD. The document also links each RegistryEntry element in the list to  
 984 an element for the identified association.  
 985

```

986
987 <ReturnRegistryEntry>
988   <RegistryEntryQuery>
989     <SourceAssociationBranch>
990       <AssociationFilter>                                -- code by Clause, Section Error! Reference
991                                               source not found.
992         associationType EQUAL "Contains" OR
993         associationType EQUAL "Uses" OR
994         associationType EQUAL "Extends"
995       </AssociationFilter>
996     <RegistryEntryFilter>                                -- code by Clause, Section Error! Reference
997                                               source not found.
998       id EQUAL "urn:xyz:dtd:po97"
999     </RegistryEntryFilter>
1000   </SourceAssociationBranch>
1001 </RegistryEntryQuery>
1002 <WithSourceAssociations>
1003   <AssociationFilter>                                -- code by Clause, Section Error!
1004                                               Reference source not found.
1005     associationType EQUAL "Contains" OR
1006     associationType EQUAL "Uses" OR
1007     associationType EQUAL "Extends"
1008   </AssociationFilter>
1009 </WithSourceAssociations>
1010 </ReturnRegistryEntry>
  
```

1011  
 1012 A user has been browsing the registry and has found a registry entry that describes a package of core-  
 1013 components that should solve the user's problem. The package URN identifier is "urn:com:cc:pkg:ccstuff".  
 1014 Now the user wants to know what's in the package. The following query returns an XML document with a  
 1015 registry entry for each member of the package along with that member's Uses and HasMemberBranch  
 1016 associations.

```

1017
1018 <ReturnRegistryEntry>
1019   <RegistryEntryQuery>
1020     <TargetAssociationBranch>
1021       <AssociationFilter>                                -- code by Clause, Section Error! Reference
1022                                               source not found.
1023         associationType EQUAL "HasMember"
1024       </AssociationFilter>
1025     <RegistryEntryFilter>                                -- code by Clause, Section Error! Reference
1026                                               source not found.
1027       id EQUAL "urn:com:cc:pkg:ccstuff"
1028     </RegistryEntryFilter>
1029   </TargetAssociationBranch>
1030 </RegistryEntryQuery>
1031 <WithSourceAssociations>
1032   <AssociationFilter>                                -- code by Clause, Section Error! Reference
1033                                               source not found.
1034     associationType EQUAL "HasMember" OR
1035     associationType EQUAL "Uses"
1036   </AssociationFilter>
  
```

```
1037     </WithSourceAssociations>
1038 </ReturnRegistryEntry>
1039
```



1039 **8.2.8 ReturnRepositoryItem**

1040 **Purpose**

1041 To construct an XML document that contains one or more repository items, and some associated  
1042 metadata, by submitting a RegistryEntryQuery to the registry/repository that holds the desired objects.  
1043 NOTE: Initially, the RegistryEntryQuery could be the URN identifier for a single registry entry.

1044 **Definition**

```
1045 <!ELEMENT ReturnRepositoryItem
1046 ( RegistryEntryQuery,
1047   RecursiveAssociationOption?,
1048   WithDescription? )>
1049
1050
1051 <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1052 <!ATTLIST RecursiveAssociationOption
1053   depthLimit CDATA #IMPLIED >
1054
1055 <!ELEMENT AssociationType EMPTY >
1056 <!ATTLIST AssociationType
1057   role CDATA #REQUIRED >
1058
1059 <!ELEMENT WithDescription EMPTY >
1060
1061 <!ELEMENT ReturnRepositoryItemResult
1062 ( RepositoryItem*)>
1063
1064 <!ELEMENT RepositoryItem
1065 ( ClassificationSchemeRepresentation
1066   | RegistryPackageElements
1067   | ExtrinsicObjectFile
1068   | WithdrawnObject
1069   | ExternalRegistryItem )>
1070 <!ATTLIST RepositoryItem
1071   id CDATA #REQUIRED
1072   name CDATA #REQUIRED
1073   objectType CDATA #REQUIRED
1074   status CDATA #REQUIRED
1075   stability CDATA #REQUIRED
1076   contentURI CDATA #IMPLIED
1077   description CDATA #IMPLIED >
1078
1079 <!ELEMENT ClassificationSchemeRepresentation
1080 ( ClassificationNode+ )>
1081
1082 <!ELEMENT RegistryPackageElements
1083 ( RegistryObject* )>
1084
1085 <!ELEMENT ExtrinsicObjectFile EMPTY >
1086 <!ATTLIST ExtrinsicObjectFile
1087   contentURI CDATA #REQUIRED > -- REF to attached file
1088
1089 <!ELEMENT WithdrawnObject EMPTY >
1090
1091 <!ELEMENT ExternalRegistryItem EMPTY >
1092
1093
1094
```

1095 **Semantic Rules**

- 1096 1. If the RecursiveOption element is not present , then set Limit=0. If the RecursiveOption element is  
1097 present, interpret its depthLimit attribute as an integer literal. If the depthLimit attribute is not present,  
1098 then set Limit = -1. A Limit of 0 means that no recursion occurs. A Limit of -1 means that recursion  
1099 occurs indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive integer,  
1100 then stop execution and raise the exception: *invalid depth limit*; otherwise, set Limit=N, where N is  
1101 that positive integer. A Limit of N means that exactly N recursive steps will be executed unless the  
1102 process terminates prior to that limit.
- 1103 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned as part of the  
1104 ReturnRepositoryItemResult. Initially Result is empty. Semantic rules 4 through 10 determine the  
1105 content of Result.
- 1106 3. If the WithDescription element is present, then set WSD="yes"; otherwise, set WSD="no".
- 1107 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in Section 8.2.2, and let R  
1108 be the result set of registry entry instances. Let S be the set of warnings and errors returned. If any  
1109 element in S is an error condition, then stop execution and return the same set of warnings and errors  
1110 along with the ReturnRepositoryItemResult.
- 1111 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from R. After  
1112 execution of these rules, if Depth is now equal to Limit, then return the content of Result as the set of  
1113 RepositoryItem elements in the ReturnRepositoryItemResult element; otherwise, continue with  
1114 Semantic Rule 8.
- 1115 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the following:  
1116 a) If E references a repository item in this registry, then create a new RepositoryItem element, with  
1117 values for its attributes derived as specified in Semantic Rule 7.  
1118 1) If E.objectType="ClassificationScheme", then put the classification scheme nodes  
1119 described by E as a ClassificationSchemeRepresentation subelement of this  
1120 RepositoryItem.  
1121 2) If E.objectType="Package", then put the package members described by E as a  
1122 RegistryPackageElements subelement of this RepositoryItem.  
1123 3) Otherwise, i.e., if the repository item referenced by E has an unknown internal structure,  
1124 then attach the file that represents that structure to the ReturnRepositoryItemResult.  
1125 Create a new ExtrinsicObjectFile as the subelement of this RepositoryItem and set the  
1126 contentURI attribute to reference the attached file.
- 1127 b) If E references a registered object in some other registry, then create a new RepositoryItem  
1128 element, with values for its attributes derived as specified in Semantic Rule 7, and create a new  
1129 ExternalRegistryItem element as the subelement of this RepositoryItem.
- 1130 c) If E describes a repository item that has since been withdrawn, then create a new RepositoryItem  
1131 element, with values for its attributes derived as specified in Semantic Rule 7, and create a new  
1132 WithdrawnObject element as the subelement of this RepositoryItem.
- 1133 7. Let E be a registry entry and let RO be the RepositoryItem element created in Semantic Rule 6. Set  
1134 the attributes of RO to the values derived from the corresponding attributes of E. If WSD="yes",  
1135 include the value of the description attribute; otherwise, do not include it. Insert this new  
1136 RepositoryItem element into the Result set.
- 1137 8. Let R be defined as in Semantic Rule **Error! Reference source not found.**. Execute Semantic Rule  
1138 9 with Y as the set of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.
- 1139 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty set of  
1140 RegistryEntry instances. For each registry entry E in Y, and for each AssociationType of the  
1141 RecursiveAssociationOption, do the following:  
1142 a) Let Z be the set of target items E' linked to E under Association instances having E as the source  
1143 object, E' as the target object, and with the associationType of the association equal to the value  
1144 of the role attribute of that AssociationType.  
1145 b) Add the elements of Z to NextLevel.
- 1146 10. Let X be the set of new registry entries that are in NextLevel but are not yet represented in the Result  
1147 set.  
1148 Case:  
1149 a) If X is empty, then return the content of Result as the set of RepositoryItem elements in the  
1150 ReturnRepositoryItemResult element.

- 1151       b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set. When finished,  
 1152       add the elements of X to Y and set Depth=Depth+1. If Depth is now equal to Limit, then return  
 1153       the content of Result as the set of RepositoryItem elements in the ReturnRepositoryItemResult  
 1154       element; otherwise, repeat Semantic Rules 9 and 10 with the new set Y of registry entries.  
 1155 11. If any exception, warning, or other status condition results during the execution of the above, then  
 1156       return appropriate RegistryError elements in the RegistryResult associated with the  
 1157       ReturnRepositoryItemResult element created in Semantic Rule 5 or Semantic Rule 10.

## 1158 **Examples**

1159 A registry client has found a registry entry for a core-component item. The item's URN identity is  
 1160 "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many other registered items. The  
 1161 client desires the collection of all items needed for a complete implementation of "goodthing". The  
 1162 following query returns an XML document that is a collection of all needed items. The query follows all  
 1163 "Uses" and "ValidatesTo" association types through an arbitrary number of recursive steps to return every  
 1164 repository item in this registry that is needed by "goodthing".

```
1166 <ReturnRepositoryItem>
1167   <RegistryEntryQuery>
1168     <RegistryEntryFilter>                                -- code by Clause, Section Error! Reference
1169     id EQUAL "urn:ebxml:cc:goodthing"                    source not found.
1170   </RegistryEntryFilter>
1171 </RegistryEntryQuery>
1172 <RecursiveAssociationOption>
1173   <AssociationType role="Uses" />
1174   <AssociationType role="ValidatesTo" />
1175 </RecursiveAssociationOption>
1176 </ReturnRepositoryItem>
```

1179 A registry client has found a reference to a core-component routine ("urn:ebxml:cc:rtn:nice87") that  
 1180 implements a given business process. The client knows that all routines have a required association to its  
 1181 defining UML specification. The following query returns both the routine and its UML specification as a  
 1182 collection of two items in a single XML document.

```
1184 <ReturnRepositoryItem>
1185   <RegistryEntryQuery>
1186     <RegistryEntryFilter>                                -- code by Clause, Section Error! Reference
1187     id EQUAL "urn:ebxml:cc:rtn:nice87"                    source not found.
1188   </RegistryEntryFilter>
1189 </RegistryEntryQuery>
1190 <RecursiveAssociationOption depthLimit="1" >
1191   <AssociationType role="ValidatesTo" />
1192 </RecursiveAssociationOption>
1193 </ReturnRepositoryItem>
```

1196 A user has been told that the 1997 version of the North American Industry Classification System (NAICS)  
 1197 is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The following query would retrieve the  
 1198 complete classification scheme, with all 1810 nodes, as an XML document that contains all of the  
 1199 ClassificationNode instances for the ClassificationScheme instance identified by that URN.

```
1201 <ReturnRepositoryItem>
1202   <RegistryEntryQuery>
1203     <RegistryEntryFilter>                                -- code by Clause, Section Error!
1204     id EQUAL "urn:nist:cs:naics-1997"                    Reference source not found.
1205   </RegistryEntryFilter>
```

```
1207     </RegistryEntryQuery>
1208 </ReturnRepositoryItem>
1209
1210
1211
```

## 1211 8.2.9 Registry Filters

### 1212 Purpose

1213 To identify a subset of the set of all persistent instances of a given registry class.

### 1214 Definition

1215  
1216 <!ELEMENT RegistryObjectFilter ( Clause )>  
1217  
1218 <!ELEMENT RegistryEntryFilter ( Clause )>  
1219  
1220 <!ELEMENT ExtrinsicObjectFilter ( Clause )>  
1221  
1222 <!ELEMENT PackageFilter ( Clause )>  
1223  
1224 <!ELEMENT OrganizationFilter ( Clause  
1225  
1226 <!ELEMENT ClassificationNodeFilter ( Clause )>  
1227  
1228 <!ELEMENT AssociationFilter ( Clause )>  
1229  
1230 <!ELEMENT ClassificationFilter ( Clause )>  
1231  
1232 <!ELEMENT ExternalLinkFilter ( Clause )>  
1233  
1234 <!ELEMENT ExternalIdentifierFilter ( Clause )>  
1235  
1236 <!ELEMENT SlotFilter ( Clause )>  
1237  
1238 <!ELEMENT AuditableEventFilter ( Clause )>  
1239  
1240 <!ELEMENT UserFilter ( Clause )>  
1241  
1242 <!ELEMENT PathFilter ( Clause )>  
1243  
1244 <!ELEMENT PathElementFilter ( Clause )>  
1245  
1246 <!ELEMENT SlotElementFilter ( Clause )>  
1247

### 1248 Semantic Rules

- 1249 1. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
- 1250 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing  
1251 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in [ebRIM]. If  
1252 not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set of identifiers for  
1253 RegistryObject instances whose attribute values evaluate to *True* for the Clause predicate.
- 1254 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing SimpleClause  
1255 shall identify a public attribute of the RegistryEntry UML class defined in [ebRIM]. If not, raise  
1256 exception: *registry entry attribute error*. The RegistryEntryFilter returns a set of identifiers for  
1257 RegistryEntry instances whose attribute values evaluate to *True* for the Clause predicate.
- 1258 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing  
1259 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in [ebRIM]. If  
1260 not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter returns a set of  
1261 identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause  
1262 predicate.

- 1263 5. For every PackageFilter XML element, the leftArgument attribute of any containing SimpleClause  
1264 shall identify a public attribute of the Package UML class defined in [ebRIM]. If not, raise exception:  
1265 *package attribute error*. The PackageFilter returns a set of identifiers for Package instances whose  
1266 attribute values evaluate to *True* for the Clause predicate.
- 1267 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing SimpleClause  
1268 shall identify a public attribute of the Organization or PostalAddress UML classes defined in [ebRIM].  
1269 If not, raise exception: *organization attribute error*. The OrganizationFilter returns a set of identifiers  
1270 for Organization instances whose attribute values evaluate to *True* for the Clause predicate.
- 1271 7. For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing  
1272 SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM].  
1273 If not, raise exception: *classification node attribute error*. The ClassificationNodeFilter returns a set of  
1274 identifiers for ClassificationNode instances whose attribute values evaluate to *True* for the Clause  
1275 predicate.
- 1276 8. For every AssociationFilter XML element, the leftArgument attribute of any containing SimpleClause  
1277 shall identify a public attribute of the Association UML class defined in [ebRIM]. If not, raise exception:  
1278 *association attribute error*. The AssociationFilter returns a set of identifiers for Association instances  
1279 whose attribute values evaluate to *True* for the Clause predicate.
- 1280 9. For every ClassificationFilter XML element, the leftArgument attribute of any containing SimpleClause  
1281 shall identify a public attribute of the Classification UML class defined in [ebRIM]. If not, raise  
1282 exception: *classification attribute error*. The ClassificationFilter returns a set of identifiers for  
1283 Classification instances whose attribute values evaluate to *True* for the Clause predicate.
- 1284 10. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing SimpleClause  
1285 shall identify a public attribute of the ExternalLink UML class defined in [ebRIM]. If not, raise  
1286 exception: *external link attribute error*. The ExternalLinkFilter returns a set of identifiers for  
1287 ExternalLink instances whose attribute values evaluate to *True* for the Clause predicate.
- 1288 11. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing  
1289 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in [ebRIM]. If  
1290 not, raise exception: *external identifier attribute error*. The ExternalIdentifierFilter returns a set of  
1291 identifiers for ExternalIdentifier instances whose attribute values evaluate to *True* for the Clause  
1292 predicate.
- 1293 12. For every SlotFilter XML element, the leftArgument attribute of any containing SimpleClause shall  
1294 identify a public attribute of the Slot UML class defined in [ebRIM]. If not, raise exception: *slot attribute*  
1295 *error*. The SlotFilter returns a set of identifiers for Slot instances whose attribute values evaluate to  
1296 *True* for the Clause predicate.
- 1297 13. For every AuditableEventFilter XML element, the leftArgument attribute of any containing  
1298 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [ebRIM]. If  
1299 not, raise exception: *auditable event attribute error*. The AuditableEventFilter returns a set of  
1300 identifiers for AuditableEvent instances whose attribute values evaluate to *True* for the Clause  
1301 predicate.
- 1302 14. For every UserFilter XML element, the leftArgument attribute of any containing SimpleClause shall  
1303 identify a public attribute of the User UML class defined in [ebRIM]. If not, raise exception: *auditable*  
1304 *identity attribute error*. The UserFilter returns a set of identifiers for User instances whose attribute  
1305 values evaluate to *True* for the Clause predicate.
- 1306 15. Path is a derived, non-persistent class based on the ClassificationNode and Classification classes  
1307 from ebRIM. The visible attributes of the Path class are “path”, “code”, and “pathDepth”. Each is  
1308 derived from the corresponding method defined in ebRIM for a ClassificationNode or Classification  
1309 instance. The getPath() method acts on a ClassificationNode or Classification instance to produce a  
1310 character string, i.e. path, that can be queried by the predicates of a StringClause element. The  
1311 getCode() method on a Classification instance returns a string value, i.e. code: (i) if an internal  
1312 classification, returns the code attribute of the referenced ClassificationNode, and (ii) if an external  
1313 classification, returns the classification value submitted by the classifier (ebRIM definitions needed!).  
1314 The getPathDepth() method acts on a ClassificationNode or Classification instance to produce an  
1315 integer that identifies the level of the referenced node and that can be queried by the predicates of an  
1316 IntClause element. For an external Classification instance, getPathDepth() may return void since the  
1317 depth of the node referenced by that classification may not be known if it wasn’t supplied by the  
1318 classifier. For every PathFilter XML element, the leftArgument attribute of any containing  
1319 SimpleClause shall identify a public attribute of the Path class just defined. If not, raise exception:

- 1320 *path attribute error*. The PathFilter returns a set of Path instances whose attribute values evaluate to  
 1321 *True* for the Clause predicate.
- 1322 16. PathElement is a derived, non-persistent class based on the ClassificationNode and Classification  
 1323 classes from ebRIM. The visible attributes of PathElement are “level” and “value”. Each is a character  
 1324 string. The dynamic instances of PathElement are derived from the getPathElements() method  
 1325 defined in ebRIM for a ClassificationNode or Classification instance. This method returns a set of  
 1326 level/value pairs for each ClassificationNode or Classification instance. For an external Classification  
 1327 instance, getPathElements() may return void since the explicit structure of the node referenced by  
 1328 that classification may not be known if it wasn’t supplied by the classifier. For every PathElementFilter  
 1329 XML element, the leftArgument attribute of any containing SimpleClause shall identify a public  
 1330 attribute of the PathElement class just defined. If not, raise exception: *path element attribute error*.  
 1331 The PathElementFilter returns a set of PathElement instances whose attribute values evaluate to  
 1332 *True* for the Clause predicate.
- 1333 17. SlotElement is a derived, non-persistent class based on the Slot class from ebRIM. The visible  
 1334 attribute of PathElement is “value”. It is a character string. The dynamic instances of SlotElement are  
 1335 derived from the “values” attribute defined in ebRIM for a Slot instance. For every SlotElementFilter  
 1336 XML element, the leftArgument attribute of any containing SimpleClause shall identify the “value”  
 1337 attribute of the SlotElement class just defined. If not, raise exception: *slot element attribute error*. The  
 1338 SlotElementFilter returns a set of Slot instances whose “value” attribute evaluates to *True* for the  
 1339 Clause predicate.
- 1340

1341 **Example**

1342 The following is a complete example of RegistryEntryQuery combined with Clause expansion of  
 1343 RegistryEntryFilter to return a set of RegistryEntry instances whose objectType attribute is “CPP” and  
 1344 whose status attribute is “Approved”.

```

1345
1346 <RegistryEntryQuery>
1347   <RegistryEntryFilter>
1348     <Clause>
1349       <CompoundClause   connectivePredicate="And" >
1350         <Clause>
1351           <SimpleClause leftArgument="objectType" >
1352             <StringClause stringPredicate="equal" >CPP</StringClause>
1353           </SimpleClause>
1354         </Clause>
1355         <Clause>
1356           <SimpleClause leftArgument="status" >
1357             <StringClause stringPredicate="equal" >Approved</StringClause>
1358           </SimpleClause>
1359         </Clause>
1360       </CompoundClause>
1361     </Clause>
1362   </RegistryEntryFilter>
1363 </RegistryEntryQuery>
1364
1365
```

1366 **8.2.10 XML Clause Constraint Representation**

1367  
 1368 NOTE to Editor: This proposal makes no changes to Section 8.2.10, so it remains as currently specified  
 1369 in ebRS v1.1.