

1 **This document is a proposed revision**
2 **of Section 8 of ebRS v1.0 (27 June 2001)**
3 **Submitted on behalf of the Registry Query Team**
4 **by Len Gallagher, Oct 12, 2001**

5 **8. Object(?) Query Management Service**

6
7 NOTE to EDITOR: Some changes will be needed to the introduction of Section 8, "Query Management",
8 in the existing ebRS document to reflect the status of Section 8.1, Browse and Drill Down Query Support",
9 and Section 8.3, "SQL Query Support", after voting results are final. This proposal deals primarily with
10 Section 8.2, "Filter Query Support" and with the re-ordering the existing Section 8.4, "Ad Hoc Query
11 Request/Response", to make it the new Section 8.1.

12 **8.1 Ad Hoc Query Request/Response**

13
14 NOTE to EDITOR: This section is the result of the Query team's decision to move Section 8.4, Ad Hoc
15 Query Request/Response", to make it the new Section 8.1, and to change the name of
16 "ObjectQueryManager" to just "QueryManager". I've copied the text from ebRS v1.0 and removed the
17 word "Object" as a qualifier of Query. This "Object" wording change will have ramifications in Appendix A
18 and in the RAWs proposal that will have to be addressed by the Editor.

19
20 A client submits an ad hoc query to the ObjectQueryManager by sending an AdhocQueryRequest. The
21 AdhocQueryRequest contains a sub-element that defines a query in one of the supported Registry query
22 mechanisms.

23 The ObjectQueryManager sends an AdhocQueryResponse either synchronously or asynchronously back
24 to the client. The AdhocQueryResponse returns a collection of objects whose element type is in the set of
25 element types represented by the leaf nodes of the RegistryEntry hierarchy in [ebRIM].



27
28 **Figure 1: Submit Ad Hoc Query Sequence Diagram**

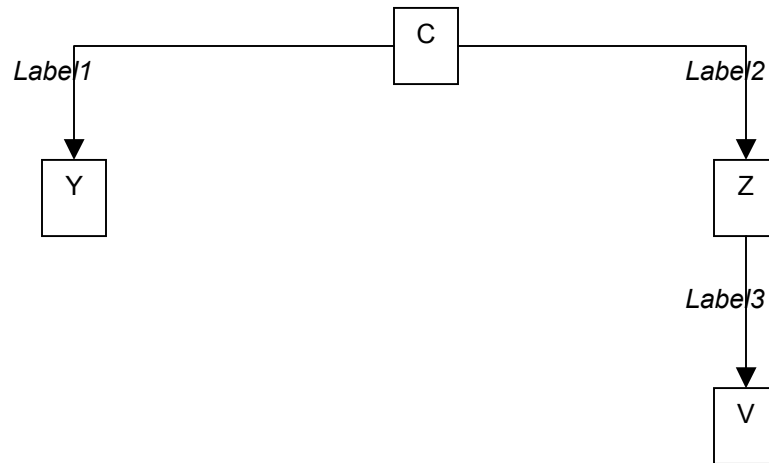
29 For details on the schema for the business documents shown in this process refer to **Error! Reference**
30 **source not found..**

32 8.2 Filter Query Support

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

38 A client submits a FilterQuery, a ReturnRegistryEntry query, or a ReturnRepositoryItem query to the
39 ObjectQueryManager as part of an AdhocQueryRequest. The ObjectQueryManager sends an
40 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResponse,
41 ReturnRegistryEntryResponse, or ReturnRepositoryItemResponse specified herein. The sequence
42 diagrams for AdhocQueryRequest and AdhocQueryResponse are specified in Section 8.1.

43 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of classes
44 derived from a single class and its associations with other classes as defined by ebRIM. Each choice of a
45 class pre-determines a virtual XML document that can be queried as a tree. For example, let C be a
46 class, let Y and Z be classes that have direct associations to C, and let V be a class that is associated
47 with Z. The ebRIM Binding for C might be as in Figure 2.



48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65 **Figure 2: Example ebRIM Binding**

66
67 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and Label3
68 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to which ebRIM
69 association is intended. The name of the query is determined by the root class, i.e. this is an ebRIM
70 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
71 association identified by Label1. Similarly, the Z and V nodes are limited to instances that are linked to
72 their parent node by the identified association.

73 Each FilterQuery alternative depends upon one or more *class filters*, where a class filter is a restricted
74 *predicate clause* over the attributes of a single class. The supported class filters are specified in Section
75 8.2.9 and the supported predicate clauses are defined in Section **Error! Reference source not found.**
76 A FilterQuery will be composed of elements that traverse the tree to determine which branches satisfy the
77 designated class filters, and the query result will be the set of root node instances that support such a
78 branch.

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

86 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
87 element on Y is allowed. However, if the association is one-to-many, then multiple filter or branch
88 elements are allowed. This allows one to specify that an instance of C must have associations with
89 multiple instances of Y before the instance of C is said to satisfy the branch element.
90 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be stable,
91 the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then the FilterQuery
92 syntax and semantics can be extended at the same time.
93 Support for FilterQuery is required of every conforming ebXML Registry implementation, but other query
94 options are possible. The Registry will hold a self-describing CPP that identifies all supported
95 AdhocQuery options. This profile is described in Section **Error! Reference source not found.**
96 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.6 below identify the virtual hierarchy for
97 each FilterQuery alternative. The Semantic Rules for each query alternative specify the effect of that
98 binding on query semantics.
99 The ReturnRegistryEntry and ReturnRepositoryItem services defined below provide a way to structure an
100 XML document as an expansion of the result of a RegistryEntryQuery. The ReturnRegistryEntry element
101 specified in Section 8.2.7 allows one to specify what metadata one wants returned with each registry
102 entry identified in the result of a RegistryEntryQuery. The ReturnRepositoryItem specified in Section
103 8.2.8 allows one to specify what repository items one wants returned based on their relationships to the
104 registry entries identified by the result of a RegistryEntryQuery.
105

105 8.2.1 FilterQuery

106 Purpose

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

111 Definition

```
112
113 <!ELEMENT FilterQuery
114   ( RegistryEntryQuery
115     | AuditableEventQuery
116     | ClassificationNodeQuery
117     | RegistryPackageQuery
118     | OrganizationQuery )>
119
120 <!ELEMENT FilterQueryResult
121   ( RegistryEntryQueryResult
122     | AuditableEventQueryResult
123     | ClassificationNodeQueryResult
124     | RegistryPackageQueryResult
125     | OrganizationQueryResult )>
126
127 <!ELEMENT RegistryEntryQueryResult ( RegistryEntryView* )>
128
129 <!ELEMENT RegistryEntryView EMPTY >
130 <!ATTLIST RegistryEntryView
131   id          CDATA      #REQUIRED
132   name        CDATA      #REQUIRED
133   contentURI  CDATA      #IMPLIED >
134
135 <!ELEMENT AuditableEventQueryResult ( AuditableEventView* )>
136
137 <!ELEMENT AuditableEventView EMPTY >
138 <!ATTLIST AuditableEventView
139   id          CDATA      #REQUIRED
140   name        CDATA      #REQUIRED
141   eventType   CDATA      #REQUIRED
142   timestamp   CDATA      #REQUIRED >
143
144 <!ELEMENT ClassificationNodeQueryResult (ClassificationNodeView*)>
145
146 <!ELEMENT ClassificationNodeView EMPTY >
147 <!ATTLIST ClassificationNodeView
148   id          CDATA      #REQUIRED
149   name        CDATA      #REQUIRED
150   code        CDATA      #REQUIRED
151   parent      CDATA      #REQUIRED >
152
153 <!ELEMENT RegistryPackageQueryResult ( RegistryPackageView* )>
154
155 <!ELEMENT RegistryPackageView EMPTY >
156 <!ATTLIST RegistryPackageView
157   id          CDATA      #REQUIRED
158   name        CDATA      #REQUIRED >
```

```
159
160 <!ELEMENT OrganizationQueryResult ( OrganizationView* )>
161
162 <!ELEMENT OrganizationView EMPTY >
163 <!ATTLIST OrganizationView
164     id          CDATA      #REQUIRED
165     name        CDATA      #REQUIRED >
166
167
```

168 **Semantic Rules**

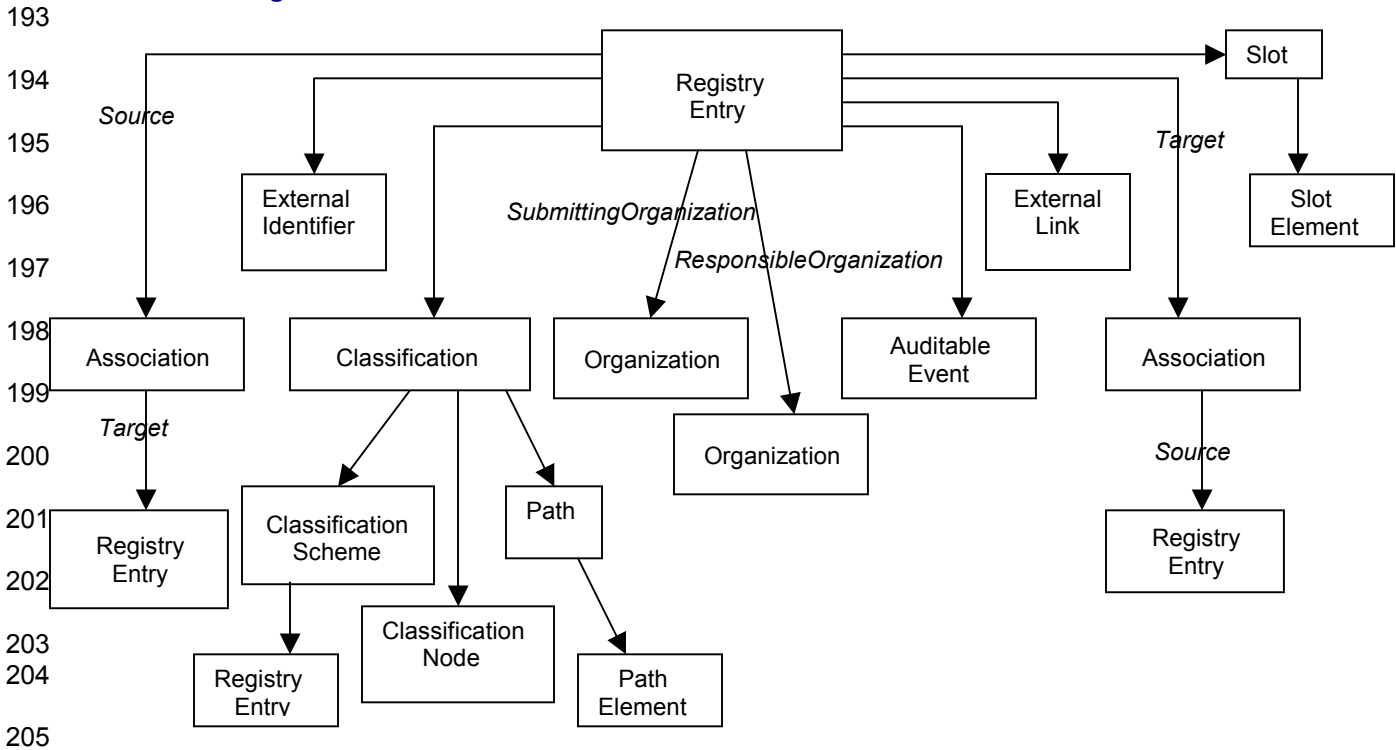
- 169 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- 170 2. Each FilterQueryResult is a set of XML elements to identify each instance of the result set. Each XML
- 171 attribute carries a value derived from the value of an attribute specified in the Registry Information
- 172 Model as follows:
 - 173 a) id carries the value of the id attribute of the RegistryObject class,
 - 174 b) name carries the value of the name attribute of the RegistryObjectClass,
 - 175 c) contentURI, if present, carries the value of the contentURI attribute of the ExtrinsicObject class,
 - 176 d) timestamp carries a character string literal value to represent the value of the timestamp attribute
 - 177 of the AuditableEvent class,
 - 178 e) code carries the value of the code attribute of the ClassificationNode class.
- 179 3. If an error condition is raised during any part of the execution of a FilterQuery, then the status
- 180 attribute of the XML RegistryResult is set to “failure” and no query result element is returned; instead,
- 181 a RegistryErrorList element must be returned with its highestSeverity element set to “error”. At least
- 182 one of the RegistryError elements in the RegistryErrorList will have its severity attribute set to “error”.
- 183 4. If no error conditions are raised during execution of a FilterQuery, then the status attribute of the XML
- 184 RegistryResult is set to “success” and an appropriate query result element must be included. If a
- 185 RegistryErrorList is also returned, then the highestSeverity attribute of the RegistryErrorList is set to
- 186 “warning” and the serverity attribute of each RegistryError is set to “warning”.
- 187
- 188
- 189

189 **8.2.2 RegistryEntryQuery**

190 **Purpose**

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

192 **ebRIM Binding**



206 **Definition**

```

207
208 <!ELEMENT RegistryEntryQuery
209   ( RegistryEntryFilter?,
210     SourceAssociationBranch*,
211     TargetAssociationBranch*,
212     HasClassificationBranch*,
213     SubmittingOrganizationBranch?,
214     ResponsibleOrganizationBranch?,
215     ExternalIdentifierBranch*,
216     ExternalLinkBranch*,
217     HasSlotBranch*,
218     HasAuditableEventBranch*      )>
219
220 <!ELEMENT SourceAssociationBranch
221   ( AssociationFilter?,
222     ( RegistryEntryFilter? | RegistryEntryQuery? ) )>
223
224 <!ELEMENT TargetAssociationBranch
225   ( AssociationFilter?,
226     ( RegistryEntryFilter? | RegistryEntryQuery? ) )>
227
228 <!ELEMENT HasClassificationBranch
229   ( ClassificationFilter?,
  
```

```

230     FromSchemeBranch?,
231     HasPathBranch?,
232     LocalNodeBranch?,
233     SubmittingOrganizationBranch?    )>
234
235 <!ELEMENT FromSchemeBranch
236   ( ClassificationSchemeFilter | RegistryEntryQuery )>
237
238 <!ELEMENT HasPathBranch
239   ( PathFilter | XpathNodeExpression | PathElementFilter+ )>
240
241 <!ELEMENT XpathNodeExpression ( #PCDATA )>
242

```

243 Author'sNOTE: The HasPathBranch specifies 3 alternatives, each of which has flaws. PathFilter and
244 PathElementFilter depend upon the definition of new methods for the ClassificationNode and
245 Classification classes in ebRIM and the XpathNodeExpression depends upon a fully specified getPath()
246 method in the ClassificationNode class and a specification of an XPATH Expresssion. The PathFilter
247 option remains in the ebRS specification even if none of the supporting proposals are adopted, although
248 the levelNumber attribute is deleted if the getLevelNumber() proposal fails. The DEFAULT action for the
249 other two is that they remain in the specification if all of their supporting proposals pass, but are deleted if
250 any one of the supporting proposals for it fails when voted on by the full Registry TC. The supporting
251 proposals for each option are:

```

252
253 PathFilter
254   getLevelNumber()    http://lists.oasis-open.org/archives/regrep-query/200110/pdf00001.pdf
255   getPath()           http://lists.oasis-open.org/archives/regrep-query/200110/pdf00006.pdf
256
257 XpathNodeExpression
258   getPath()           http://lists.oasis-open.org/archives/regrep-query/200110/pdf00006.pdf
259   XPATH Expression    http://lists.oasis-open.org/archives/regrep-query/200110/pdf00007.pdf
260
261 PathElementFilter
262   getLevelNumber()    http://lists.oasis-open.org/archives/regrep-query/200110/pdf00001.pdf
263   getPathElements()   http://lists.oasis-open.org/archives/regrep-query/200110/pdf00004.pdf
264

```

265 It is possible that one or more of these alternatives will become superfluous with respect to the others,
266 with likely deletion of the less useful alternative(s) even if fully specified.

```

267
268 <!ELEMENT LocalNodeBranch
269   ( ClassificationNodeFilter? | ClassificationNodeQuery? )>
270
271 <!ELEMENT SubmittingOrganizationBranch
272   ( OrganizationFilter | OrganizationQuery )>
273
274 <!ELEMENT ResponsibleOrganizationBranch
275   ( OrganizationFilter? | OrganizationQuery? )>
276
277 <!ELEMENT ExternalIdentifierBranch
278   ( ExternalIdentifierFilter?,
279     SubmittingOrganizationBranch? )>
280
281 <!ELEMENT ExternalLinkBranch
282   ( ExternalLinkFilter )>
283
284 <!ELEMENT HasSlotBranch
285   ( SlotFilter?,
286     SlotElementFilter* )>
287

```

```
288 <!ELEMENT HasAuditableEventBranch
289 ( AuditableEventFilter? | AuditableEventQuery? )>
290
291
```

292 **Semantic Rules**

- 293 1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following steps will
294 eliminate instances in RE that do not satisfy the conditions of the specified filters.
 - 295 a) If a RegistryEntryFilter is not specified, or if RE is empty, then continue below; otherwise, let x be
296 a registry entry in RE. If x does not satisfy the RegistryEntryFilter as defined in Section 8.2.9, then
297 remove x from RE.
 - 298 b) If a SourceAssociationBranch element is not specified, or if RE is empty, then continue below;
299 otherwise, let x be a remaining registry entry in RE. If x is not the source object of some
300 Association instance, then remove x from RE; otherwise, treat each SourceAssociationBranch
301 element separately as follows:
302 If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be the set of
303 all Association instances that have x as a source object; otherwise, let AF be the set of
304 Association instances that satisfy the AssociationFilter and have x as the source object. If AF is
305 empty, then remove x from RE. If no RegistryEntryFilter or RegistryEntryQuery is specified within
306 SourceAssociationBranch, then let RET be the set of all RegistryEntry instances that are the
307 target object of some element of AF; otherwise, let RET be the set of RegistryEntry instances that
308 satisfy the RegistryEntryFilter or RegistryEntryQuery and are the target object of some element
309 of AF. If RET is empty, then remove x from RE.
 - 310 c) If a TargetAssociationBranch element is not specified, or if RE is empty, then continue below;
311 otherwise, let x be a remaining registry entry in RE. If x is not the target object of some
312 Association instance, then remove x from RE; otherwise, treat each TargetAssociationBranch
313 element separately as follows:
314 If no AssociationFilter is specified within TargetAssociationBranch, then let AF be the set of all
315 Association instances that have x as a target object; otherwise, let AF be the set of Association
316 instances that satisfy the AssociationFilter and have x as the target object. If AF is empty, then
317 remove x from RE. If no RegistryEntryFilter or RegistryEntryQuery is specified within
318 TargetAssociationBranch, then let RES be the set of all RegistryEntry instances that are the
319 source object of some element of AF; otherwise, let RES be the set of RegistryEntry instances
320 that satisfy the RegistryEntryFilter or RegistryEntryQuery and are the source object of some
321 element of AF. If RES is empty, then remove x from RE.
 - 322 d) If a HasClassificationBranch 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 the classifiedObject of some
324 Classification instance, then remove x from RE; otherwise, treat each HasClassificationBranch
325 element separately as follows:
326 If no ClassificationFilter is specified within the HasClassificationBranch, then let CL be the set of
327 all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of
328 Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If
329 CL is empty, then remove x from RE and continue below. Otherwise, if CL is not empty, and if a
330 FromSchemeBranch is specified, then replace CL by the set of remaining Classification instances
331 in CL whose defining classification scheme satisfies the ClassificationSchemeFilter or the
332 RegistryEntryQuery immediately contained in the FromSchemeBranch. If the new CL is empty,
333 then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a
334 HasPathBranch is specified, then replace CL by the set of remaining Classification instances in
335 CL that satisfy the PathFilter, the XpathNodeExpression, or every one of the PathElementFilter
336 elements immediately contained in the HasPathBranch. If the new CL is empty, then remove x
337 from RE and continue below. Otherwise, if CL remains not empty, and if a LocalNodeBranch is
338 specified, then replace CL by the set of remaining Classification instances in CL for which a local
339 node exists and for which that local node satisfies the ClassificationNodeFilter or the
340 ClassificationNodeQuery immediately contained in the LocalNodeBranch. . If the new CL is
341 empty, then remove x from RE and continue below. Otherwise, if CL remains not empty, and if a
342 SubmittingOrganizationBranch is specified, then replaceCL by the set of remaining Classification
343 instances in CL for which the submitting organization of that classification satisfies the

344 OrganizationFilter or OrganizationQuery immediately contained in the
345 SubmittingOrganizationBranch. If the new CL is empty, then remove x from RE.

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

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

356 g) If an ExternalIdentifierBranch element is not specified, or if RE is empty, then continue below;
357 otherwise, let x be a remaining registry entry in RE. If x is not linked to some ExternalIdentifier
358 instance, then remove x from RE; otherwise, treat each ExternalIdentifierBranch element
359 separately as follows: If an ExternalIdentifierFilter is not specified, then let EI be the set of
360 ExternalIdentifier instances that are linked to x; otherwise, let EI be the set of ExternalIdentifier
361 instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x
362 from RE and continue below. Otherwise, if EI remains not empty, and if a
363 SubmittingOrganizationBranch is specified, replace EI by the set of remaining ExternalIdentifier
364 instances in EI for which the OrganizationFilter or OrganizationQuery immediately contained in
365 the SubmittingOrganizationBranch is valid. If the new EI is empty, then remove x from RE.

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

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

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

386 2. If RE is empty, then raise the warning: *registry entry query result is empty*; otherwise, return RE as
387 the result of the RegistryEntryQuery.

388 3. Return any accumulated warnings or exceptions as the StatusResult associated with the
389 RegistryEntryQuery.

390 Examples

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

```
396
397 <RegistryEntryQuery>
398   <RegistryEntryFilter>
```

```

399         status EQUAL "Approved"           -- code by Clause, Section Error! Reference
400                                           source not found.
401     </RegistryEntryFilter>
402     <SubmittingOrganizationBranch>
403         <OrganizationFilter>
404             name CONTAINS "XYZ"           -- code by Clause, Section Error! Reference
405                                           source not found.
406         </OrganizationFilter>
407     </SubmittingOrganizationBranch>
408 </RegistryEntryquery>
409

```

410 A client is using the United Nations Standard Product and Services Classification (UNSPSC) scheme and
411 wants to identify all companies that deal with products classified as "Integrated circuit components", i.e.
412 UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol
413 Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC
414 according to the products the company deals with. However, the client does not know if the UNSPSC
415 classification scheme is internal or external to this registry. The following query returns a set of approved
416 registry entry instances for CPP's of companies that deal with integrated circuit components.

```

417
418     <RegistryEntryQuery>
419         <RegistryEntryFilter>
420             objectType EQUAL "CPP" AND           -- code by Clause, Section Error! Reference
421                                           source not found.
422             status EQUAL "Approved"
423         </RegistryEntryFilter>
424         <HasClassificationBranch>
425             <FromSchemeBranch>
426                 <ClassificationSchemeFilter>
427                     id EQUAL "urn:org:un:spsc:cs2001" -- code by Clause, Section Error!
428                                                         Reference source not found.
429                 </ClassificationSchemeFilter>
430             </FromSchemeBranch>
431             <HasPathBranch>
432                 <PathFilter>
433                     code EQUAL "321118"
434                 </PathFilter>
435             </HasPathBranch>
436         </HasClassificationBranch>
437     </RegistryEntryQuery>

```

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

```

443
444     <RegistryEntryQuery>
445         <HasClassificationBranch>
446             <FromSchemeBranch>
447                 <ClassificationSchemeFilter>
448                     id EQUAL "urn:ebxml:cs:industry" -- code by Clause, Section Error!
449                                                         Reference source not found.
450                 </ClassificationSchemeFilter>
451             </FromSchemeBranch>
452             <HasPathBranch>
453                 <XPathExpression>
454                     getPath = "//Automotive"
455

```

```

456         </XPathExpression>
457     </HasPathExpression>
458 </HasClassificationBranch>
459 <HasClassificationBranch>
460     <FromSchemeBranch>
461         <ClassificationSchemeFilter>
462             id EQUAL "urn:ebxml:cs:geography"    -- code by Clause, Section Error!
463                                                     Reference source not found.
464         </ClassificationSchemeFilter>
465     </FromSchemeBranch>
466     <HasPathBranch>
467         <PathFilter>
468             path STARTSWITH "/Geography-id/Asia/Japan"
469         </PathFilter>
470     </HasPathBranch>
471 </HasClassificationBranch>
472 </RegistryEntryQuery>
473

```

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

```

477
478 <RegistryEntryQuery>
479     <RegistryEntryFilter>
480         objectType EQUAL "Package"    -- code by Clause, Section Error! Reference
481                                     source not found.
482     </RegistryEntryFilter>
483     <SourceAssociationBranch>
484         <AssociationFilter>
485             associationType EQUAL "HasMember"    -- code by Clause, Section Error!
486                                                     Reference source not found.
487         </AssociationFilter>
488         <RegistryEntryFilter>
489             id EQUAL "urn:path:myitem"    -- code by Clause, Section Error! Reference
490                                             source not found.
491         </RegistryEntryFilter>
492     </SourceAssociationBranch>
493 </RegistryEntryQuery>
494

```

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

```

500
501 <RegistryEntryQuery>
502     <HasClassificationBranch>
503         <LocalNodeBranch>
504             <ClassificationNodeFilter>
505
506                 name CONTAINS "transistor" OR    -- code by Clause, Section Error!
507                                                     Reference source not found.
508                 description CONTAINS "transistor"
509             </ClassificationNodeFilter>
510         </LocalNodeBranch>
511     </HasClassificationBranch>
512 </RegistryEntryQuery>
513

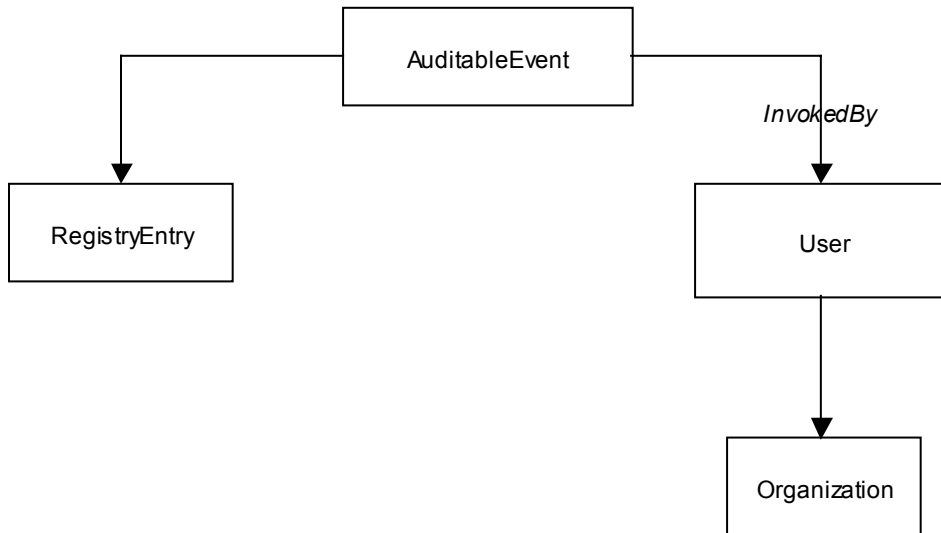
```

513 **8.2.3 AuditableEventQuery**

514 **Purpose**

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

516 **ebRIM Binding**



517 **Definition**

```
518
519 <!ELEMENT AuditableEventQuery
520   ( AuditableEventFilter?,
521     RegistryEntryQuery*,
522     InvokedByBranch? )>
523
524 <!ELEMENT InvokedByBranch
525   ( UserFilter?,
526     OrganizationQuery? )>
527
```

528 **Semantic Rules**

- 529 1. Let AE denote the set of all persistent AuditableEvent instances in the Registry. The following steps
530 will eliminate instances in AE that do not satisfy the conditions of the specified filters.
- 531
- 532 a) If an AuditableEventFilter is not specified, or if AE is empty, then continue below; otherwise, let x
533 be an auditable event in AE. If x does not satisfy the AuditableEventFilter as defined in Section
534 8.2.9, then remove x from AE.
- 535 b) If a RegistryEntryQuery element is not specified, or if AE is empty, then continue below;
536 otherwise, let x be a remaining auditable event in AE. Treat each RegistryEntryQuery element
537 separately as follows:
538 Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.2. If x is not an
539 auditable event for some registry entry in RE, then remove x from AE.
- 540 c) If an InvokedByBranch element is not specified, or if AE is empty, then continue below; otherwise,
541 let x be a remaining auditable event in AE.
- 542 Let u be the user instance that invokes x. If a UserFilter element is specified within the InvokedByBranch,
543 and if u does not satisfy that filter, then remove x from AE; otherwise, continue below.

544 If an OrganizationQuery element is not specified within the InvokedByBranch, then continue
 545 below; otherwise, let OG be the set of Organization instances that are identified by the
 546 organization attribute of u and are in the result set of the OrganizationQuery. If OG is empty, then
 547 remove x from AE.
 548 2. If AE is empty, then raise the warning: *auditable event query result is empty*.
 549 3. Return AE as the result of the AuditableEventQuery.
 550

551 **Examples**

552 A Registry client has registered an item and it has been assigned a URN identifier "urn:path:myitem".
 553 The client is now interested in all events since the beginning of the year that have impacted that item. The
 554 following query will return a set of AuditableEvent instances for all such events.
 555

```
556 <AuditableEventquery>
557   <AuditableEventFilter>
558     timestamp GE "2001-01-01" AND
559     registryEntry EQUAL "urn:path:myitem"
560   </AuditableEventFilter>
561 </AuditableEventQuery>
```

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

563 A client company has many registered objects in the Registry. The Registry allows events submitted by
 564 other organizations to have an impact on your registered items, e.g. new classifications and new
 565 associations. The following query will return a set of identifiers for all auditable events, invoked by some
 566 other party, that had an impact on an item submitted by "myorg" and for which "myorg" is the responsible
 567 organization.
 568

```
570 <AuditableEventQuery>
571   <RegistryEntryQuery>
572     <SubmittingOrganizationBranch>
573       <OrganizationFilter>
574         id EQUAL "urn:somepath:myorg"
575       </OrganizationFilter>
576     </SubmittingOrganizationBranch>
577     <ResponsibleOrganizationBranch>
578       <OrganizationFilter>
579         id EQUAL "urn:somepath:myorg"
580       </OrganizationFilter>
581     </ResponsibleOrganizationBranch>
582   </RegistryEntryQuery>
583   <InvokedByBranch>
584     <OrganizationQuery>
585       <OrganizationFilter>
586         id NOT_EQUAL "urn:somepath:myorg"
587       </OrganizationFilter>
588     </OrganizationQuery>
589   </InvokedByBranch>
590 </AuditableEventQuery>
```

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

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

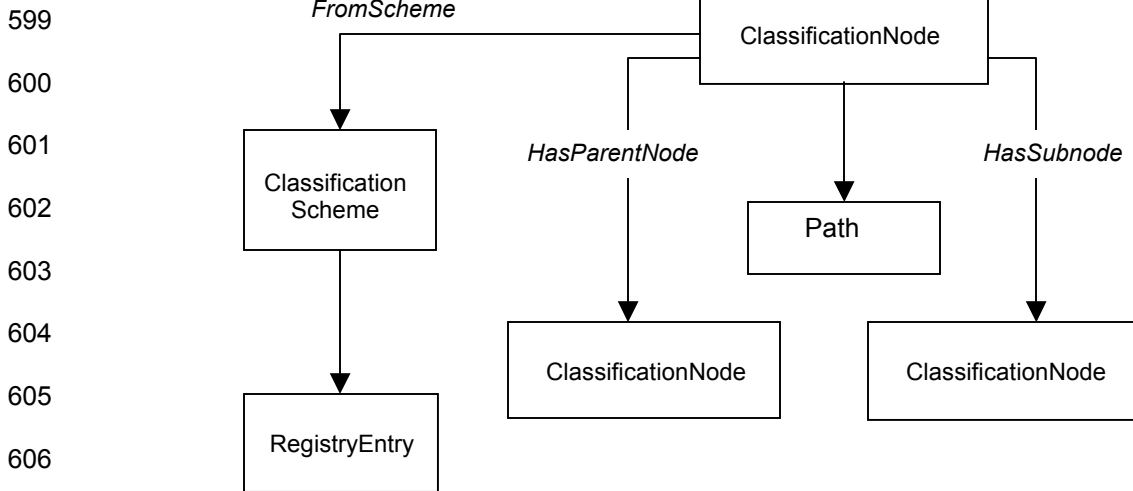
594 **8.2.4 ClassificationNodeQuery**

595 **Purpose**

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

597 **ebRIM Binding**

598



607

608

609 **Definition**

610

```
611 <!ELEMENT ClassificationNodeQuery
612 ( ClassificationNodeFilter?,
613 FromSchemeBranch?,
614 HasPathBranch?,
615 HasParentNodeBranch?,
616 HasSubnodeBranch* )>
```

617

```
618 <!ELEMENT HasParentNodeBranch
619 ( ClassificationNodeFilter?,
620 HasPathBranch?,
621 HasParentNodeBranch? )>
```

622

```
623 <!ELEMENT HasSubnodeBranch
624 ( ClassificationNodeFilter?,
625 HasPathBranch?,
626 HasSubnodeBranch* )>
```

627

628 **Semantic Rules**

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

- 634 b) If a FromSchemeBranch is not specified, or if CN is empty, then continue below; otherwise, let x
 635 be a remaining classification node in CN. If the defining classification scheme of x does not
 636 satisfy the ClassificationSchemeFilter or the RegistryEntryQuery immediately contained in the
 637 FromSchemeBranch, then remove x from CN.
- 638 c) If a HasPathBranch is not specified, or if CN is empty, then continue below; otherwise, let x be a
 639 remaining classification node in CN. If the path derived from x does not satisfy the PathFilter, the
 640 XpathNodeExpression, or every one of the PathElementFilter elements immediately contained in
 641 the HasPathBranch, then remove x from CN.
- 642 d) If a HasParentNodeBranch element is not specified, or if CN is empty, then continue below;
 643 otherwise, let x be a remaining classification node in CN and execute the following paragraph
 644 with n=x.

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

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

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

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

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

- 670 2. If CN is empty, then raise the warning: *classification node query result is empty*.
 671 3. Return CN as the result of the ClassificationNodeQuery.
 672

673 Examples

674
 675 A client application wishes to identify all of the classification nodes in the first three levels of a
 676 classification scheme hierarchy. The client knows that the URN identifier for the underlying classification
 677 scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three levels.
 678

```
679 <ClassificationNodeQuery>
680   <FromSchemeBranch>
681     <ClassificationSchemeFilter>
682       id EQUAL "urn:ebxml:cs:myscheme" -- code by Clause, Section Error!
683                                     Reference source not found.
684     </ClassificationSchemeFilter>
685   </FromSchemeBranch>
686   <HasPathBranch>
687     <PathFilter>
```

```
688         levelNumber LE "3"
689     </PathFilter>
690 </HasPathBranch>
691 </ClassificationNodeQuery>
```

692
693 If, instead, the client wishes all levels returned, they could simply delete the HasPathBranch element from
694 the query.

695
696 By assuming that the "path" of a node is known, and the URN of the classification scheme it comes from,
697 one could get all nodes at the next level below that node as follows:

```
698  
699 <ClassificationNodeQuery>
700     <FromSchemeBranch>
701         <ClassificationSchemeFilter>
702             id EQUAL "urn:some:known:scheme"
703         </ClassificationSchemeFilter>
704     </FromSchemeBranch>
705     <HasParentBranch>
706         <HasPathBranch>
707             <PathFilter>
708                 path EQUAL "KnownPathOfGivenNode"
709             </PathFilter>
710         </HasPathBranch>
711     </HasParentBranch>
712 </ClassificationNodeQuery>
```

713
714 If instead, one wanted ALL nodes in the subtree beneath the given node, then the following query could
715 be used:

```
716  
717 <ClassificationNodeQuery>
718     <FromSchemeBranch>
719         <ClassificationSchemeFilter>
720             id EQUAL "urn:some:known:scheme"
721         </ClassificationSchemeFilter>
722     </FromSchemeBranch>
723     <HasParentBranch>
724         <HasPathBranch>
725             <PathFilter>
726                 path STARTSWITH "KnownPathOfGivenNode"
727             </PathFilter>
728         </HasPathBranch>
729     </HasParentBranch>
730 </ClassificationNodeQuery>
731
```

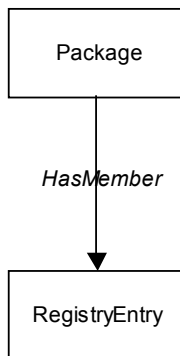

731 **8.2.5 RegistryPackageQuery**

732
733 NOTE to Registry TC: With the proposed re-structuring of ebRIM to make Package a subtype of
734 RegistryEntry, a RegistryPackageQuery may be superfluous because the Package class has no
735 attributes and no methods beyond those specified for RegistryEntry. There is nothing that can be done
736 with this RegistryPackageQuery that can't also be done in a straight-forward manner with just a
737 RegistryEntryQuery. For example, see the penultimate example of Section 8.2.2, "RegistryEntryQuery", or
738 the last example of Section 8.2.7, "ReturnRegistryEntry". The possible deletion of RegistryPackageQuery
739 was not considered by the Query Team! It should be considered by the Registry TC. The DEFAULT
740 action is that this section will be deleted as unnecessary.

741 **Purpose**

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

743 **ebRIM Binding**



744 **Definition**

```
745  
746 <!ELEMENT RegistryPackageQuery  
747   ( PackageFilter?,  
748     HasMemberBranch* )>  
749  
750 <!ELEMENT HasMemberBranch  
751   ( RegistryEntryQuery? )>
```

752

753 **Semantic Rules**

- 754 1. Let RP denote the set of all persistent Package instances in the Registry. The following steps will
755 eliminate instances in RP that do not satisfy the conditions of the specified filters.
- 756 a) If a PackageFilter is not specified, or if RP is empty, then continue below; otherwise, let x be a
757 package instance in RP. If x does not satisfy the PackageFilter as defined in Section 8.2.9, then
758 remove x from RP.
 - 759 b) If a HasMemberBranch element is not directly contained in the RegistryPackageQuery, or if RP is
760 empty, then continue below; otherwise, let x be a remaining package instance in RP. If x is an
761 empty package, then remove x from RP; otherwise, treat each HasMemberBranch element
762 separately as follows:
763
764 If a RegistryEntryQuery element is not directly contained in the HasMemberBranch element, then
765 let PM be the set of all RegistryEntry instances that are members of the package x; otherwise, let
766 RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in
767 Section 8.2.2 and let PM be the subset of RE that are members of the package x. If PM is empty,
768 then remove x from RP.

- 769 2. If RP is empty, then raise the warning: *registry package query result is empty*.
770 3. Return RP as the result of the RegistryPackageQuery.
771

772 **Examples**

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

```
775 <RegistryPackageQuery>  
776   <HasMemberBranch>  
777     <RegistryEntryQuery>  
778       <RegistryEntryFilter>  
779         objectType EQ "Invoice" -- code by Clause, Section Error! Reference  
780                                     source not found.  
781       </RegistryEntryFilter>  
782     </RegistryEntryQuery>  
783   </HasMemberBranch>  
784 </RegistryPackageQuery>  
785  
786
```

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

```
788  
789 <RegistryEntryQuery>  
790   <HasMemberBranch/>  
791 </RegistryEntryQuery>  
792
```

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

799 **Note:** A registry package is an intrinsic RegistryEntry instance that is completely determined by its
800 associations with its members. Thus a RegistryPackageQuery can always be re-specified as an
801 equivalent RegistryEntryQuery using appropriate "Source" and "Target" associations. However, the
802 equivalent RegistryEntryQuery is often more complicated to write.
803

803 **8.2.6 OrganizationQuery**

804 **Purpose**

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

806 **ebRIM Binding**

807

808

809

810

811

812

813

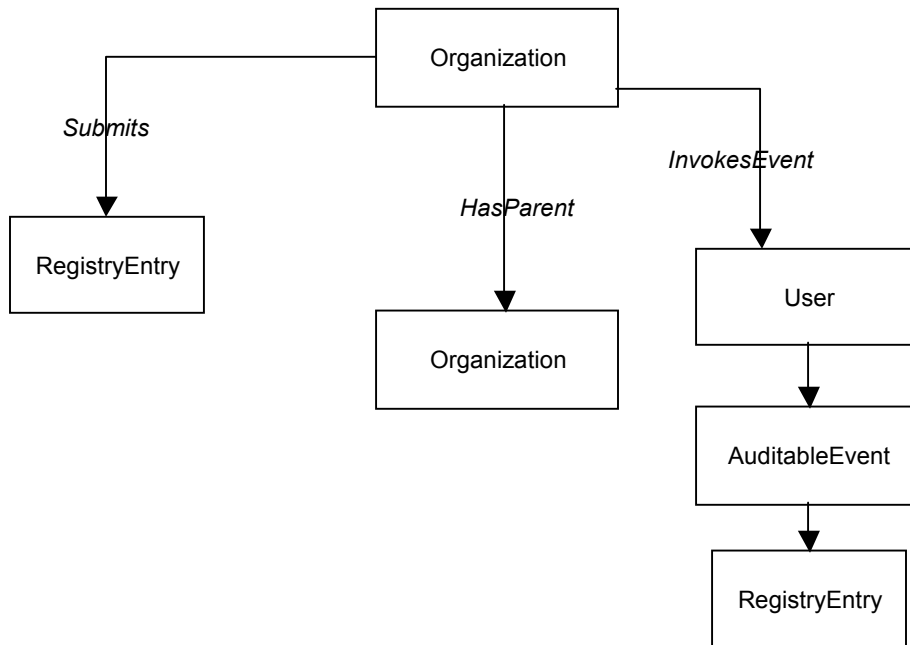
814

815

816

817

818



819 **Definition**

820

```
821 <!ELEMENT OrganizationQuery  
822 ( OrganizationFilter?,  
823 SubmitsRegistryEntry*,  
824 HasParentOrganization?,  
825 InvokesEventBranch* )>
```

826

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

828

```
829 <!ELEMENT HasParentOrganization  
830 ( OrganizationFilter?,  
831 HasParentOrganization? )>
```

832

```
833 <!ELEMENT InvokesEventBranch  
834 ( UserFilter?,  
835 AuditableEventFilter?,  
836 RegistryEntryQuery? )>
```

837 **Semantic Rules**

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

- 840 a) If an OrganizationFilter element is not directly contained in the OrganizationQuery element, or if
841 ORG is empty, then continue below; otherwise, let x be an organization instance in ORG. If x
842 does not satisfy the OrganizationFilter as defined in Section 8.2.9, then remove x from RP.
843 b) If a SubmitsRegistryEntry element is not specified within the OrganizationQuery, or if ORG is
844 empty, then continue below; otherwise, consider each SubmitsRegistryEntry element separately
845 as follows:
- 846 If no RegistryEntryQuery is specified within the SubmitsRegistryEntry element, then let RES be
847 the set of all RegistryEntry instances that have been submitted to the Registry by organization x;
848 otherwise, let RE be the result of the RegistryEntryQuery as defined in Section 8.2.2 and let RES
849 be the set of all instances in RE that have been submitted to the Registry by organization x. If
850 RES is empty, then remove x from ORG.
- 851 c) If a HasParentOrganization element is not specified within the OrganizationQuery, or if ORG is
852 empty, then continue below; otherwise, execute the following paragraph with o = x:
- 853 Let o be an organization instance. If an OrganizationFilter is not specified within the
854 HasParentOrganization and if o has no parent (i.e. if o is a root organization in the Organization
855 hierarchy), then remove x from ORG; otherwise, let p be the parent organization of o. If p does
856 not satisfy the OrganizationFilter, then remove x from ORG.
- 857 If another HasParentOrganization element is directly contained within this HasParentOrganization
858 element, then repeat the previous paragraph with o = p.
- 859 d) If an InvokesEventBranch element is not specified within the OrganizationQuery, or if ORG is
860 empty, then continue below; otherwise, consider each InvokesEventBranch element separately
861 as follows:
- 862 If an UserFilter is not specified, and if x is not the submitting organization of some AuditableEvent
863 instance, then remove x from ORG. If an AuditableEventFilter is not specified, then let AE be the
864 set of all AuditableEvent instances that have x as the submitting organization; otherwise, let AE
865 be the set of AuditableEvent instances that satisfy the AuditableEventFilter and have x as the
866 submitting organization. If AE is empty, then remove x from ORG. If a RegistryEntryQuery is not
867 specified in the InvokesEventBranch element, then let RES be the set of all RegistryEntry
868 instances associated with an event in AE; otherwise, let RE be the result set of the
869 RegistryEntryQuery, as specified in Section 8.2.2, and let RES be the subset of RE of entries
870 submitted by x. If RES is empty, then remove x from ORG.
- 871 2. If ORG is empty, then raise the warning: *organization query result is empty*.
872 3. Return ORG as the result of the OrganizationQuery.
873

874 Examples

875 A client application wishes to identify a set of organizations, based in France, that have submitted a
876 PartyProfile extrinsic object this year.
877

```

878 <OrganizationQuery>
879   <OrganizationFilter>
880     country EQUAL "France"           -- code by Clause, Section Error! Reference
881                                     source not found.
882   </OrganizationFilter>
883   <SubmitsRegistryEntry>
884     <RegistryEntryQuery>
885       <RegistryEntryFilter>
886         objectType EQUAL "CPP"      -- code by Clause, Section Error! Reference
887                                     source not found.
888       </RegistryEntryFilter>
889       <HasAuditableEventBranch>
890         <AuditableEventFilter>
891           timestamp GE "2001-01-01" -- code by Clause, Section Error!
892                                     Reference source not found.
893         </AuditableEventFilter>
894       </HasAuditableEventBranch>

```

```
895         </RegistryEntryQuery>
896     </SubmitsRegistryEntry>
897 </OrganizationQuery>
```

898
899 A client application wishes to identify all organizations that have XYZ, Corporation as a parent. The client
900 knows that the URN for XYZ, Corp. is urn:ebxml:org:xyz, but there is no guarantee that subsidiaries of
901 XYZ have a URN that uses the same format, so a full query is required.

```
902
903 <OrganizationQuery>
904     <HasParentOrganization>
905         <OrganizationFilter>
906             id EQUAL "urn:ebxml:org:xyz" -- code by Clause, Section Error! Reference
907                                     source not found.
908         </OrganizationFilter>
909     </HasParentOrganization>
910 </OrganizationQuery>
911
```

911 8.2.7 ReturnRegistryEntry

912 Purpose

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

916 Definition

```
917
918 <!ELEMENT ReturnRegistryEntry
919   ( RegistryEntryQuery,
920     WithClassifications?,
921     WithSourceAssociations?,
922     WithTargetAssociations?,
923     WithAuditableEvents?,
924     WithExternalIdentifiers?,
925     WithExternalLinks? )>
926
927 <!ELEMENT WithClassifications ( ClassificationFilter? )>
928 <!ELEMENT WithSourceAssociations ( AssociationFilter? )>
929 <!ELEMENT WithTargetAssociations ( AssociationFilter? )>
930 <!ELEMENT WithAuditableEvents ( AuditableEventFilter? )>
931 <!ELEMENT WithExternalIdentifiers ( ExternalIdentifierFilter? )>
932 <!ELEMENT WithExternalLinks ( ExternalLinkFilter? )>
933
934 <!ELEMENT ReturnRegistryEntryResult
935   ( RegistryEntryMetadata*)>
936
937 <!ELEMENT RegistryEntryMetadata
938   ( RegistryEntry,
939     Classification*,
940     SourceAssociations?,
941     TargetAssociations?,
942     AuditableEvent*,
943     ExternalIdentifier*,
944     ExternalLink* )>
945
946 <!ELEMENT SourceAssociations ( Association* )>
947 <!ELEMENT TargetAssociations ( Association* )>
```

948 Semantic Rules

- 949 1. The RegistryEntry, Classification, Association, AuditableEvent, and ExternalLink elements contained
950 in the ReturnRegistryEntryResult are defined by the ebXML Registry schema specified in Appendix A.
- 951 2. Execute the RegistryEntryQuery according to the Semantic Rules specified in Section 8.2.2, and let R
952 be the result set of registry entry instances. Let S be the set of warnings and errors returned. If any
953 element in S is an error condition, then stop execution and return the same set of warnings and errors
954 along with the ReturnRegistryEntryResult.
- 955 3. If the set R is empty, then do not return a RegistryEntryMetadata subelement in the
956 ReturnRegistryEntryResult. Instead, raise the warning: *no resulting registry entry*. Add this warning to
957 the error list returned by the RegistryEntryQuery and return this enhanced error list with the
958 ReturnRegistryEntryResult.
- 959 4. For each registry entry E referenced by an element of R, use the attributes of E to create a new
960 RegistryEntry element as defined in Appendix A. Then create a new RegistryEntryMetadata element
961 as defined above to be the parent element of that RegistryEntry element.
- 962 5. If no With option is specified, then the resulting RegistryEntryMetadata element has no Classification,
963 SourceAssociations, TargetAssociations, AuditableEvent, or ExternalData subelements. The set of

- 964 RegistryEntryMetadata elements, with the Error list from the RegistryEntryQuery, is returned as the
 965 ReturnRegistryEntryResult.
- 966 6. If WithClassifications is specified, then for each E in R do the following: If a ClassificationFilter is not
 967 present, then let C be any classification instance linked to E; otherwise, let C be a classification
 968 instance linked to E that satisfies the ClassificationFilter (Section 8.2.9). For each such C, create a
 969 new Classification element as defined in Appendix A. Add these Classification elements to their
 970 parent RegistryEntryMetadata element.
 - 971 7. If WithSourceAssociations is specified, then for each E in R do the following: If an AssociationFilter is
 972 not present, then let A be any association instance whose source object is E; otherwise, let A be an
 973 association instance that satisfies the AssociationFilter (Section 8.2.9) and whose source object is E.
 974 For each such A, create a new Association element as defined in Appendix A. Add these Association
 975 elements as subelements of the WithSourceAssociations and add that element to its parent
 976 RegistryEntryMetadata element.
 - 977 8. If WithTargetAssociations is specified, then for each E in R do the following: If an AssociationFilter is
 978 not present, then let A be any association instance whose target object is E; otherwise, let A be an
 979 association instance that satisfies the AssociationFilter (Section 8.2.9) and whose target object is E.
 980 For each such A, create a new Association element as defined in Appendix A. Add these Association
 981 elements as subelements of the WithTargetAssociations and add that element to its parent
 982 RegistryEntryMetadata element.
 - 983 9. If WithAuditableEvents is specified, then for each E in R do the following: If an AuditableEventFilter is
 984 not present, then let A be any auditable event instance linked to E; otherwise, let A be any auditable
 985 event instance linked to E that satisfies the AuditableEventFilter (Section 8.2.9). For each such A,
 986 create a new AuditableEvent element as defined in Appendix A. Add these AuditableEvent elements
 987 to their parent RegistryEntryMetadata element.
 - 988 10. If WithExternalIdentifiers is specified, then for each E in R do the following: If an
 989 ExternalIdentifierFilter is not present, then let I be any external identifier instance linked to E;
 990 otherwise, let I be any external identifier instance linked to E that satisfies the ExternalIdentifierFilter
 991 (Section 8.2.9). For each such I, create a new ExternalIdentifier element as defined in Appendix A.
 992 Add these ExternalIdentifier elements to their parent RegistryEntryMetadata element.
 - 993 11. If WithExternalLinks is specified, then for each E in R do the following: If an ExternalLinkFilter is not
 994 present, then let L be any external link instance linked to E; otherwise, let L be any external link
 995 instance linked to E that satisfies the ExternalLinkFilter (Section 8.2.9). For each such D, create a
 996 new ExternalLink element as defined in Appendix A. Add these ExternalLink elements to their parent
 997 RegistryEntryMetadata element.
 - 998 12. If any warning or error condition results, then add the code and the message to the
 999 RegistryResponse element that includes the RegistryEntryQueryResult.
 - 1000 13. Return the set of RegistryEntryMetadata elements as the content of the ReturnRegistryEntryResult.
 1001

1002 Examples

1003 A customer of XYZ Corporation has been using a PurchaseOrder DTD registered by XYZ some time ago.
 1004 Its URN identifier is "urn:com:xyz:po:325". The customer wishes to check on the current status of that
 1005 DTD, especially if it has been superceded or replaced, and get all of its current classifications. The
 1006 following query request will return an XML document with the registry entry for the existing DTD as the
 1007 root, with all of its classifications, and with associations to registry entries for any items that have
 1008 superceded or replaced it.

```

1010 <ReturnRegistryEntry>
1011   <RegistryEntryQuery>
1012     <RegistryEntryFilter>
1013       id EQUAL "urn:com:xyz:po:325"           -- code by Clause, Section Error!
1014                                             Reference source not found.
1015     </RegistryEntryFilter>
1016   </RegistryEntryQuery>
1017   <WithClassifications/>
1018   <WithSourceAssociations>
  
```

```

1019         <AssociationFilter>                                     -- code by Clause, Section Error!
1020                                                         Reference source not found.
1021             associationType EQUAL "SupersededBy" OR
1022             associationType EQUAL "ReplacedBy"
1023         </AssociationFilter>
1024     </WithSourceAssociations>
1025 </ReturnRegistryEntry>
1026

```

A client of the Registry registered an XML DTD several years ago and is now thinking of replacing it with a revised version. The identifier for the existing DTD is "urn:xyz:dtd:po97". The proposed revision is not completely upward compatible with the existing DTD. The client desires a list of all registered items that use the existing DTD so they can assess the impact of an incompatible change. The following query returns an XML document that is a list of all RegistryEntry elements that represent registered items that use, contain, or extend the given DTD. The document also links each RegistryEntry element in the list to an element for the identified association.

```

1035
1036 <ReturnRegistryEntry>
1037     <RegistryEntryQuery>
1038         <SourceAssociationBranch>
1039             <AssociationFilter>                                     -- code by Clause, Section Error! Reference
1040                                                         source not found.
1041                 associationType EQUAL "Contains" OR
1042                 associationType EQUAL "Uses" OR
1043                 associationType EQUAL "Extends"
1044             </AssociationFilter>
1045             <RegistryEntryFilter>                                     -- code by Clause, Section Error! Reference
1046                                                         source not found.
1047                 id EQUAL "urn:xyz:dtd:po97"
1048             </RegistryEntryFilter>
1049         </SourceAssociationBranch>
1050     </RegistryEntryQuery>
1051     <WithSourceAssociations>
1052         <AssociationFilter>                                     -- code by Clause, Section Error!
1053                                                         Reference source not found.
1054             associationType EQUAL "Contains" OR
1055             associationType EQUAL "Uses" OR
1056             associationType EQUAL "Extends"
1057         </AssociationFilter>
1058     </WithSourceAssociations>
1059 </ReturnRegistryEntry>
1060

```

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

```

1066
1067 <ReturnRegistryEntry>
1068     <RegistryEntryQuery>
1069         <TargetAssociationBranch>
1070             <AssociationFilter>                                     -- code by Clause, Section Error! Reference
1071                                                         source not found.
1072                 associationType EQUAL "HasMember"
1073             </AssociationFilter>
1074             <RegistryEntryFilter>                                     -- code by Clause, Section Error! Reference
1075                                                         source not found.

```



```
1076         id EQUAL "urn:com:cc:pkg:ccstuff"
1077     </RegistryEntryFilter>
1078 </TargetAssociationBranch>
1079 </RegistryEntryQuery>
1080 <WithSourceAssociations>
1081     <AssociationFilter>
1082         associationType EQUAL "HasMember" OR
1083         associationType EQUAL "Uses"
1084     </AssociationFilter>
1085 </WithSourceAssociations>
1086 </ReturnRegistryEntry>
1087
1088
```

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

1088 8.2.8 ReturnRepositoryItem

1089 Purpose

1090 To construct an XML document that contains one or more repository items, and some associated
1091 metadata, by submitting a RegistryEntryQuery to the registry/repository that holds the desired objects.
1092 NOTE: Initially, the RegistryEntryQuery could be the URN identifier for a single registry entry.

1094 NOTE to EDITOR: During the Query Team's discussion of this proposal there was support for deleting the
1095 "WithDescription" option from the ReturnRepositoryItem XML element, but no vote to take that action. The
1096 ~~strikeout~~ in the Description and Semantic Rules sections below, if retained, would achieve that deletion.

1097 Definition

```
1098
1099 <!ELEMENT ReturnRepositoryItem
1100   ( RegistryEntryQuery,
1101     RecursiveAssociationOption?,
1102     WithDescription? )>
1103
1104 <!ELEMENT RecursiveAssociationOption ( AssociationType+ )>
1105 <!ATTLIST RecursiveAssociationOption
1106   depthLimit CDATA #IMPLIED >
1107
1108 <!ELEMENT AssociationType EMPTY >
1109 <!ATTLIST AssociationType
1110   role CDATA #REQUIRED >
1111
1112 <!ELEMENT WithDescription EMPTY >
1113
1114 <!ELEMENT ReturnRepositoryItemResult
1115   ( RepositoryItem*)>
1116
1117 <!ELEMENT RepositoryItem
1118   ( ClassificationSchemeRepresentation
1119     | RegistryPackageElements
1120     | ExtrinsicObjectFile
1121     | WithdrawnObject
1122     | ExternalRegistryItem )>
1123 <!ATTLIST RepositoryItem
1124   id CDATA #REQUIRED
1125   name CDATA #REQUIRED
1126   objectType CDATA #REQUIRED
1127   status CDATA #REQUIRED
1128   stability CDATA #REQUIRED
1129   contentURI CDATA #IMPLIED
1130   description CDATA #IMPLIED #REQUIRED >
1131
1132 <!ELEMENT ClassificationSchemeRepresentation
1133   ( ClassificationNode+ )>
1134
1135 <!ELEMENT RegistryPackageElements
1136   ( RegistryObject* )>
1137
1138 <!ELEMENT ExtrinsicObjectFile EMPTY >
1139 <!ATTLIST ExtrinsicObjectFile
1140   contentURI CDATA #REQUIRED > -- REF to attached file
1141
1142 <!ELEMENT WithdrawnObject EMPTY >
1143
```

1144 <!ELEMENT ExternalRegistryItem EMPTY >
1145
1146

1147 Semantic Rules

- 1148 1. If the RecursiveOption element is not present , then set Limit=0. If the RecursiveOption element is
1149 present, interpret its depthLimit attribute as an integer literal. If the depthLimit attribute is not present,
1150 then set Limit = -1. A Limit of 0 means that no recursion occurs. A Limit of -1 means that recursion
1151 occurs indefinitely. If a depthLimit value is present, but it cannot be interpreted as a positive integer,
1152 then stop execution and raise the exception: *invalid depth limit*; otherwise, set Limit=N, where N is
1153 that positive integer. A Limit of N means that exactly N recursive steps will be executed unless the
1154 process terminates prior to that limit.
 - 1155 2. Set Depth=0. Let Result denote the set of RepositoryItem elements to be returned as part of the
1156 ReturnRepositoryItemResult. Initially Result is empty. Semantic rules 4 through 10 determine the
1157 content of Result.
 - 1158 ~~3. If the WithDescription element is present, then set WSD="yes"; otherwise, set WSD="no".~~
 - 1159 4. Execute the RegistryEntryQuery according to the Semantic Rules specified in Section 8.2.2, and let R
1160 be the result set of registry entry instances. Let S be the set of warnings and errors returned. If any
1161 element in S is an error condition, then stop execution and return the same set of warnings and errors
1162 along with the ReturnRepositoryItemResult.
 - 1163 5. Execute Semantic Rules 6 and 7 with X as a set of registry references derived from R. After
1164 execution of these rules, if Depth is now equal to Limit, then return the content of Result as the set of
1165 RepositoryItem elements in the ReturnRepositoryItemResult element; otherwise, continue with
1166 Semantic Rule 8.
 - 1167 6. Let X be a set of RegistryEntry instances. For each registry entry E in X, do the following:
1168 a) If E references a repository item in this registry, then create a new RepositoryItem element, with
1169 values for its attributes derived as specified in Semantic Rule 7.
 - 1170 1) If E.objectType="ClassificationScheme", then put the classification scheme nodes
1171 described by E as a ClassificationSchemeRepresentation subelement of this
1172 RepositoryItem.
 - 1173 2) If E.objectType="Package", then put the package members described by E as a
1174 RegistryPackageElements subelement of this RepositoryItem.
 - 1175 3) Otherwise, i.e., if the repository item referenced by E has an unknown internal structure,
1176 then attach the file that represents that structure to the ReturnRepositoryItemResult.
1177 Create a new ExtrinsicObjectFile as the subelement of this RepositoryItem and set the
1178 contentURI attribute to reference the attached file.
 - 1179 b) If E references a registered object in some other registry, then create a new RepositoryItem
1180 element, with values for its attributes derived as specified in Semantic Rule 7, and create a new
1181 ExternalRegistryItem element as the subelement of this RepositoryItem.
 - 1182 c) If E describes a repository item that has since been withdrawn, then create a new RepositoryItem
1183 element, with values for its attributes derived as specified in Semantic Rule 7, and create a new
1184 WithdrawnObject element as the subelement of this RepositoryItem.
- 1185 7. Let E be a registry entry and let RO be the RepositoryItem element created in Semantic Rule 6. Set
1186 the attributes of RO to the values derived from the corresponding attributes of E. ~~If WSD="yes",~~
1187 ~~include the value of the description attribute; otherwise, do not include it.~~ Insert this new
1188 RepositoryItem element into the Result set.
- 1189 8. Let R be defined as in Semantic Rule **Error! Reference source not found.** Execute Semantic Rule
1190 9 with Y as the set of RegistryEntry instances referenced by R. Then continue with Semantic rule 10.
- 1191 9. Let Y be a set of references to RegistryEntry instances. Let NextLevel be an empty set of
1192 RegistryEntry instances. For each registry entry E in Y, and for each AssociationType of the
1193 RecursiveAssociationOption, do the following:
 - 1194 a) Let Z be the set of target items E' linked to E under Association instances having E as the source
1195 object, E' as the target object, and with the associationType of the association equal to the value
1196 of the role attribute of that AssociationType.
 - 1197 b) Add the elements of Z to NextLevel.
- 1198 10. Let X be the set of new registry entries that are in NextLevel but are not yet represented in the Result
1199 set.

- 1200 Case:
- 1201 a) If X is empty, then return the content of Result as the set of RepositoryItem elements in the
- 1202 ReturnRepositoryItemResult element.
- 1203 b) If X is not empty, then execute Semantic Rules 6 and 7 with X as the input set. When finished,
- 1204 add the elements of X to Y and set Depth=Depth+1. If Depth is now equal to Limit, then return
- 1205 the content of Result as the set of RepositoryItem elements in the ReturnRepositoryItemResult
- 1206 element; otherwise, repeat Semantic Rules 9 and 10 with the new set Y of registry entries.
- 1207 11. If any exception, warning, or other status condition results during the execution of the above, then
- 1208 return appropriate RegistryError elements in the RegistryResult associated with the
- 1209 ReturnRepositoryItemResult element created in Semantic Rule 5 or Semantic Rule 10.

1210 Examples

1211 A registry client has found a registry entry for a core-component item. The item's URN identity is

1212 "urn:ebxml:cc:goodthing". But "goodthing" is a composite item that uses many other registered items. The

1213 client desires the collection of all items needed for a complete implementation of "goodthing". The

1214 following query returns an XML document that is a collection of all needed items. The query follows all

1215 "Uses" and "ValidatesTo" association types through an arbitrary number of recursive steps to return every

1216 repository item in this registry that is needed by "goodthing".

```

1217
1218 <ReturnRepositoryItem>
1219   <RegistryEntryQuery>
1220     <RegistryEntryFilter>
1221       id EQUAL "urn:ebxml:cc:goodthing"
1222     </RegistryEntryFilter>
1223   </RegistryEntryQuery>
1224   <RecursiveAssociationOption>
1225     <AssociationType role="Uses" />
1226     <AssociationType role="ValidatesTo" />
1227   </RecursiveAssociationOption>
1228 </ReturnRepositoryItem>
1229
1230

```

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

1231 A registry client has found a reference to a core-component routine ("urn:ebxml:cc:rtn:nice87") that

1232 implements a given business process. The client knows that all routines have a required association to

1233 their defining UML specification. The following query returns both the routine and its UML specification as

1234 a collection of two items in a single XML document.

```

1235
1236 <ReturnRepositoryItem>
1237   <RegistryEntryQuery>
1238     <RegistryEntryFilter>
1239       id EQUAL "urn:ebxml:cc:rtn:nice87"
1240     </RegistryEntryFilter>
1241   </RegistryEntryQuery>
1242   <RecursiveAssociationOption depthLimit="1" >
1243     <AssociationType role="ValidatesTo" />
1244   </RecursiveAssociationOption>
1245 </ReturnRepositoryItem>
1246
1247

```

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

1248 A user has been told that the 1997 version of the North American Industry Classification System (NAICS)

1249 is stored in a registry with URN identifier "urn:nist:cs:naics-1997". The following query would retrieve the

1250 complete classification scheme, with all 1810 nodes, as an XML document that contains all of the

1251 ClassificationNode instances for the ClassificationScheme instance identified by that URN.

```

1252
1253 <ReturnRepositoryItem>
1254   <RegistryEntryQuery>

```

```
1255         <RegistryEntryFilter>                                -- code by Clause, Section Error!
1256                                                     Reference source not found.
1257             id EQUAL "urn:nist:cs:naics-1997"
1258         </RegistryEntryFilter>
1259     </RegistryEntryQuery>
1260 </ReturnRepositoryItem>
1261
1262
1263
```

1263 8.2.9 Registry Filters

1264 Purpose

1265 To identify a subset of the set of all persistent instances of a given registry class.

1266 Definition

1267
1268 <!ELEMENT RegistryObjectFilter (Clause)>
1269
1270 <!ELEMENT RegistryEntryFilter (Clause)>
1271
1272 <!ELEMENT ExtrinsicObjectFilter (Clause)>
1273
1274 <!ELEMENT PackageFilter (Clause)>
1275
1276 <!ELEMENT OrganizationFilter (Clause)>
1277
1278 <!ELEMENT ClassificationSchemeFilter (Clause)>
1279
1280 <!ELEMENT ClassificationNodeFilter (Clause)>
1281
1282 <!ELEMENT AssociationFilter (Clause)>
1283
1284 <!ELEMENT ClassificationFilter (Clause)>
1285
1286 <!ELEMENT ExternalLinkFilter (Clause)>
1287
1288 <!ELEMENT ExternalIdentifierFilter (Clause)>
1289
1290 <!ELEMENT SlotFilter (Clause)>
1291
1292 <!ELEMENT AuditableEventFilter (Clause)>
1293
1294 <!ELEMENT UserFilter (Clause)>
1295
1296 <!ELEMENT PathFilter (Clause)>
1297
1298 <!ELEMENT PathElementFilter (Clause)>
1299
1300 <!ELEMENT SlotElementFilter (Clause)>
1301

1302 Semantic Rules

- 1303 1. The Clause element is defined in Section **Error! Reference source not found.**, Clause.
- 1304 2. For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
1305 SimpleClause shall identify a public attribute of the RegistryObject UML class defined in [ebRIM]. If
1306 not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set of identifiers for
1307 RegistryObject instances whose attribute values evaluate to *True* for the Clause predicate.
- 1308 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing SimpleClause
1309 shall identify a public attribute of the RegistryEntry UML class defined in [ebRIM]. If not, raise
1310 exception: *registry entry attribute error*. The RegistryEntryFilter returns a set of identifiers for
1311 RegistryEntry instances whose attribute values evaluate to *True* for the Clause predicate.
- 1312 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
1313 SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in [ebRIM]. If
1314 not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter returns a set of

- 1315 identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause
1316 predicate.
- 1317 5. For every PackageFilter XML element, the leftArgument attribute of any containing SimpleClause
1318 shall identify a public attribute of the Package UML class defined in [ebRIM]. If not, raise exception:
1319 *package attribute error*. The PackageFilter returns a set of identifiers for Package instances whose
1320 attribute values evaluate to *True* for the Clause predicate.
 - 1321 6. For every OrganizationFilter XML element, the leftArgument attribute of any containing SimpleClause
1322 shall identify a public attribute of the Organization or PostalAddress UML classes defined in [ebRIM].
1323 If not, raise exception: *organization attribute error*. The OrganizationFilter returns a set of identifiers
1324 for Organization instances whose attribute values evaluate to *True* for the Clause predicate.
 - 1325 7. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any containing
1326 SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM].
1327 If not, raise exception: *classification scheme attribute error*. The ClassificationSchemeFilter returns a
1328 set of identifiers for ClassificationScheme instances whose attribute values evaluate to *True* for the
1329 Clause predicate.
 - 1330 8. For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing
1331 SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM].
1332 If not, raise exception: *classification node attribute error*. The ClassificationNodeFilter returns a set of
1333 identifiers for ClassificationNode instances whose attribute values evaluate to *True* for the Clause
1334 predicate.
 - 1335 9. For every AssociationFilter XML element, the leftArgument attribute of any containing SimpleClause
1336 shall identify a public attribute of the Association UML class defined in [ebRIM]. If not, raise exception:
1337 *association attribute error*. The AssociationFilter returns a set of identifiers for Association instances
1338 whose attribute values evaluate to *True* for the Clause predicate.
 - 1339 10. For every ClassificationFilter XML element, the leftArgument attribute of any containing SimpleClause
1340 shall identify a public attribute of the Classification UML class defined in [ebRIM]. If not, raise
1341 exception: *classification attribute error*. The ClassificationFilter returns a set of identifiers for
1342 Classification instances whose attribute values evaluate to *True* for the Clause predicate.
 - 1343 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing SimpleClause
1344 shall identify a public attribute of the ExternalLink UML class defined in [ebRIM]. If not, raise
1345 exception: *external link attribute error*. The ExternalLinkFilter returns a set of identifiers for
1346 ExternalLink instances whose attribute values evaluate to *True* for the Clause predicate.
 - 1347 12. For every ExternalIdentifierFilter XML element, the leftArgument attribute of any containing
1348 SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in [ebRIM]. If
1349 not, raise exception: *external identifier attribute error*. The ExternalIdentifierFilter returns a set of
1350 identifiers for ExternalIdentifier instances whose attribute values evaluate to *True* for the Clause
1351 predicate.
 - 1352 13. For every SlotFilter XML element, the leftArgument attribute of any containing SimpleClause shall
1353 identify a public attribute of the Slot UML class defined in [ebRIM]. If not, raise exception: *slot attribute*
1354 *error*. The SlotFilter returns a set of identifiers for Slot instances whose attribute values evaluate to
1355 *True* for the Clause predicate.
 - 1356 14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
1357 SimpleClause shall identify a public attribute of the AuditableEvent UML class defined in [ebRIM]. If
1358 not, raise exception: *auditable event attribute error*. The AuditableEventFilter returns a set of
1359 identifiers for AuditableEvent instances whose attribute values evaluate to *True* for the Clause
1360 predicate.
 - 1361 15. For every UserFilter XML element, the leftArgument attribute of any containing SimpleClause shall
1362 identify a public attribute of the User UML class defined in [ebRIM]. If not, raise exception: *auditable*
1363 *identity attribute error*. The UserFilter returns a set of identifiers for User instances whose attribute
1364 values evaluate to *True* for the Clause predicate.
 - 1365 16. Path is a derived, non-persistent class based on the ClassificationNode and Classification classes
1366 from ebRIM. The visible attributes of the Path class are "path", "code", and "levelNumber". Each is
1367 derived from the corresponding attribute or method defined in ebRIM for a ClassificationNode or
1368 Classification instance. The `getPath()` method acts on a ClassificationNode or Classification instance
1369 to produce a character string, i.e. path, that can be queried by the predicates of a StringClause
1370 element. The `getCode()` method on a Classification instance returns a string value, i.e. code: (i) if an
1371 internal classification, returns the code attribute of the referenced ClassificationNode, and (ii) if an

1372 external classification, returns the classification value submitted by the classifier (ebRIM definitions
 1373 needed!). The `getPathDepth()` method acts on a `ClassificationNode` or `Classification` instance to
 1374 produce an integer that identifies the level of the referenced node and that can be queried by the
 1375 predicates of an `IntClause` element. For an external `Classification` instance, `getPathDepth()` may
 1376 return `void` since the depth of the node referenced by that classification may not be known if it wasn't
 1377 supplied by the classifier. For every `PathFilter` XML element, the `leftArgument` attribute of any
 1378 containing `SimpleClause` shall identify a public attribute of the `Path` class just defined. If not, raise
 1379 exception: *path attribute error*. The `PathFilter` returns a set of `Path` instances whose attribute values
 1380 evaluate to `True` for the `Clause` predicate.

1381 17. `PathElement` is a derived, non-persistent class based on the `ClassificationNode` and `Classification`
 1382 classes from ebRIM. [NOTE: There is a pending proposal to add a `getPathElements()` method to the
 1383 `ClassificationNodeClass`. This query option depends upon that method.] The set of persistent
 1384 `PathElement` instances is the `Collection` of `ClassificationNode` instances returned by the
 1385 `getPathElements()` method. The visible attributes of each `PathElement` instance are "levelNumber"
 1386 and "code". The `levelNumber` is the integer returned by the `getLevelNumber()` method and code is the
 1387 value of the code attribute of a `ClassificationNode` instance or the result of the `getCode()` method of a
 1388 `Classification` instance. Each is a character string. The dynamic instances of `PathElement` are derived
 1389 from the `getPathElements()` method defined in ebRIM for a `ClassificationNode` or `Classification`
 1390 instance. This method returns a set of level/value pairs for each `ClassificationNode` or `Classification`
 1391 instance. For an external `Classification` instance, `getPathElements()` may return `void` since the
 1392 explicit structure of the node referenced by that classification may not be known if it wasn't supplied
 1393 by the classifier. For every `PathElementFilter` XML element, the `leftArgument` attribute of any
 1394 containing `SimpleClause` shall identify a public attribute of the `PathElement` class just defined. If not,
 1395 raise exception: *path element attribute error*. The `PathElementFilter` returns a set of `PathElement`
 1396 instances whose attribute values evaluate to `True` for the `Clause` predicate.

1397 18. `SlotElement` is a derived, non-persistent class based on the `Slot` class from ebRIM. There is one
 1398 `SlotElement` instance for each "value" in the "values" list of a `Slot` instance. The visible attribute of
 1399 `PathElement` is "value". [NOTE to EDITOR: There is a pending proposal to modify the "values"
 1400 attribute of a `Slot` instance to be just a single "value". If that proposal passes then everything dealing
 1401 with `SlotElement` and `SlotElementFilter` can be deleted from this specification.] It is a character string.
 1402 The dynamic instances of `SlotElement` are derived from the "values" attribute defined in ebRIM for a
 1403 `Slot` instance. For every `SlotElementFilter` XML element, the `leftArgument` attribute of any containing
 1404 `SimpleClause` shall identify the "value" attribute of the `SlotElement` class just defined. If not, raise
 1405 exception: *slot element attribute error*. The `SlotElementFilter` returns a set of `Slot` instances whose
 1406 "value" attribute evaluates to `True` for the `Clause` predicate.
 1407

1408 Example

1409 The following is a complete example of `RegistryEntryQuery` combined with `Clause` expansion of
 1410 `RegistryEntryFilter` to return a set of `RegistryEntry` instances whose `objectType` attribute is "CPP" and
 1411 whose `status` attribute is "Approved".

```

1412
1413 <RegistryEntryQuery>
1414   <RegistryEntryFilter>
1415     <Clause>
1416       <CompoundClause   connectivePredicate="And" >
1417         <Clause>
1418           <SimpleClause   leftArgument="objectType" >
1419             <StringClause  stringPredicate="equal" >CPP</StringClause>
1420           </SimpleClause>
1421         </Clause>
1422       </Clause>
1423       <SimpleClause   leftArgument="status" >
1424         <StringClause  stringPredicate="equal" >Approved</StringClause>
1425       </SimpleClause>
1426     </Clause>
1427   </CompoundClause>

```


1428 </Clause>
1429 </RegistryEntryFilter>
1430 </RegistryEntryQuery>
1431
1432

1433 **8.2.10 XML Clause Constraint Representation**

1434
1435 NOTE to EDITOR: This proposal makes no changes to Section 8.2.10, so that section remains in the
1436 ebRS specification as currently worded in ebRS v1.0.
1437
1438

1438 **8.3 SQL Query Support**

1439
1440 NOTE to EDITOR: Section 8.3, “SQL Query Support”, is an optional alternative for AdhocQueryRequest.
1441 It’s implementation is not required for conformance to this specification. There is discussion to move it to
1442 a non-normative Appendix, but at present no action has been recommended so it remains as Section 8.3.
1443 There have been no recommended changes to its content. The DEFAULT action is that this section
1444 remains.
1445
1446

1447 **8.4 Ad Hoc Query Request/Response**

1448
1449 NOTE to EDITOR: Section 8.4, “Ad Hoc Query Request/Response”, from the existing ebRS document is
1450 recommended to be moved as a new Section 8.1 under Query Management. If that recommendation is
1451 approved by the Registry TC, then this place-holder section should be deleted from the proposal.
1452
1453

1454 **8.5 Content Retrieval**

1455
1456 NOTE to EDITOR: Section 8.5, “Content Retrieval”, of the existing ebRS specification has not yet been
1457 addressed by the Query Team. It should be retained in the re-structured Section 8 as currently worded.
1458
1459

1460 **8.6 Query and Retrieval: Typical Sequence**

1461
1462 NOTE to EDITOR: Section 8.6, “Query and Retrieval: Typical Sequence”, of the existing ebRS
1463 specification has not yet been addressed by the Query Team. It should be retained in the re-structured
1464 Section 8 as currently worded.
1465
1466

1467 **8.7 Browse and Drill Down Query Support**

1468
1469 NOTE to EDITOR: Section 8.1, “Browse and Drill Down Query Support”, of the existing ebRS v1.0 is
1470 recommended to be deleted in favor of Filter Query. If that deletion proposal is adopted by the Registry
1471 TC, then this place-holder Section titled “Browse and Drill Down Query Support”, should be deleted
1472 entirely. The DEFAULT action is that this section is deleted.
1473