

# OASIS/ebXML Registry Services Specification v2.0 DRAFT

# **OASIS/ebXML Registry Technical Committee**

6 December 2001

## 2 This page intentionally left blank.

# 3 **1 Status of this Document**

- 4 Distribution of this document is unlimited.
- 5 The document formatting is based on the Internet Society's Standard RFC format.
- 6 *This version:*
- 7 http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf
- 8 Latest version:

# 9 2 <u>http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRS.pdf</u>OASIS/ebXML 10 Registry Technical Committee

- 11 The OASIS/ebXML Registry Technical Committee has approved this document, as a DRAFT
- 12 Specification, in its current form. At the time of this approval the following were members of
- 13 the OASIS/ebXML Registry Technical Committee.
- 14 Kathryn Breininger, Boeing
- 15 Lisa Carnahan, NIST
- 16 Joseph M. Chiusano, LMI
- 17 Suresh Damodaran, Sterling Commerce
- 18 Mike DeNicola, Fujitsu
- 19 Anne Fischer, Drummond Group, Inc.
- 20 Sally Fuger, AIAG
- 21 Jong Kim, InnoDigital
- 22 Kyu-Chul Lee, Chungnam National University
- 23 Joel Munter, Intel
- 24 Farrukh Najmi, Sun Microsystems
- 25 Joel Neu, Vitria Technologies
- 26 Sanjay Patil, IONA
- 27 Neal Smith, Chevron
- 28 Nikola Stojanovic, Encoda Systems, Inc.
- 29 Prasad Yendluri, webmethods
- 30 Yutaka Yoshida, Sun Microsystems

# 31 2.1 Contributors

- 32 The following persons contributed to the content of this document, but are not voting members
- 33 of the OASIS/ebXML Registry Technical Committee.
- 34 Len Gallagher, NIST
- 35 Sekhar Vajjhala, Sun Microsystems

# 36 **Table of Contents**

37	1	Statu	is of this Document	
38	2	OASIS/ebXML Registry Technical Committee 4		
39		2.1 Contributors		
40	Tab	le of (	Contents	
41			ligures	
42			Гаbles	
43	3		duction	
44	5	3.1	Summary of Contents of Document	
45		3.2	General Conventions	
46		3.3	Audience	
47	4		gn Objectives	
48	•	4.1	Goals	
49		4.2	Caveats and Assumptions	
50	5		em Overview	
51	J	5.1	What The ebXML Registry Does	
52		5.2	How The ebXML Registry Works	
53			5.2.1 Schema Documents Are Submitted	
54			5.2.2 Business Process Documents Are Submitted	13
55			5.2.3 Seller's Collaboration Protocol Profile Is Submitted	13
56			5.2.4 Buyer Discovers The Seller	13
57			5.2.5 CPA Is Established	14
58		5.3	Registry Users	
59		5.4	Where the Registry Services May Be Implemented	
60		5.5	Implementation Conformance	
61			5.5.1 Conformance as an ebXML Registry	
62			5.5.2 Conformance as an ebXML Registry Client	
63	6		IL Registry Architecture	
64		6.1	Registry Service Described	
65		6.2	Abstract Registry Service	
66		6.3	Concrete Registry Services	
67			6.3.1 SOAP Binding	
68 60		6.1	6.3.2 ebXML Message Service Binding	
69 70		6.4 6.5	LifeCycleManager Interface	
70 71		6.6	QueryManager Interface	
72		0.0	6.6.1 Registry Client Described	
73			6.6.2 Registry Communication Bootstrapping	
74			6.6.3 RegistryClient Interface	
75			6.6.4 Registry Response	
76		6.7	Interoperability Requirements	
77			6.7.1 Client Interoperability	
78			6.7.2 Inter-Registry Cooperation	
79	7	Life	Cycle Management Service	
80		7.1	Life Cycle of a Repository Item	
81		7.2	RegistryObject Attributes	

	7.3	The Submit Objects Protocol	27
83		7.3.1 Universally Unique ID Generation	27
84		7.3.2 ID Attribute And Object References	
85		7.3.3 Audit Trail	
86		7.3.4 Submitting Organization	
87		7.3.5 Error Handling	
88		7.3.6 Sample SubmitObjectsRequest	
89	7.4	The Update Objects Protocol	
90		7.4.1 Audit Trail	
91		7.4.2 Submitting Organization	
92		7.4.3 Error Handling	
93	7.5	The Add Slots Protocol	
94	7.6	The Remove Slots Protocol	
95	7.0	The Approve Objects Protocol	
96	1.1	7.7.1 Audit Trail	
97		7.7.2 Submitting Organization	
98		0 0	
98 99	7.8		
99 100	1.0	The Deprecate Objects Protocol	
		7.8.1 Audit Trail	
101		7.8.2 Submitting Organization	
102	7.0	7.8.3 Error Handling	
103	7.9	The Remove Objects Protocol	
104		7.9.1 Deletion Scope DeleteRepositoryItemOnly	
105		7.9.2 Deletion Scope DeleteAll	
106		7.9.3 Error Handling	
107		ery Management Service	
108	8.1	Ad Hoc Query Request/Response	40
	0.1		
109		8.1.1 Query Response Options	41
109 110	8.2	8.1.1 Query Response Options Filter Query Support	41 42
109 110 111		<ul><li>8.1.1 Query Response Options</li><li>Filter Query Support</li><li>8.2.1 FilterQuery</li></ul>	41 42 44
109 110 111 112		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> </ul>	41 42 44 46
109 110 111 112 113		<ul><li>8.1.1 Query Response Options</li><li>Filter Query Support</li><li>8.2.1 FilterQuery</li></ul>	41 42 44 46
109 110 111 112 113 114		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> </ul>	
109 110 111 112 113		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> <li>8.2.3 RegistryEntryQuery</li> </ul>	
109 110 111 112 113 114		<ul> <li>8.1.1 Query Response Options.</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> <li>8.2.3 RegistryEntryQuery</li> <li>8.2.4 AssociationQuery.</li> </ul>	
109 110 111 112 113 114 115		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	
109 110 111 112 113 114 115 116		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	
109 110 111 112 113 114 115 116 117		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> <li>8.2.3 RegistryEntryQuery</li> <li>8.2.4 AssociationQuery</li> <li>8.2.5 AuditableEventQuery</li> <li>8.2.6 ClassificationQuery</li> <li>8.2.7 ClassificationNodeQuery</li> </ul>	$ \begin{array}{r}     41 \\     42 \\     44 \\     46 \\     59 \\     62 \\     64 \\     67 \\     69 \\     74 \\ \end{array} $
109 110 111 112 113 114 115 116 117 118		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{c}    $
109 110 111 112 113 114 115 116 117 118 119		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> <li>8.2.3 RegistryEntryQuery</li> <li>8.2.4 AssociationQuery</li> <li>8.2.5 AuditableEventQuery</li> <li>8.2.6 ClassificationQuery</li> <li>8.2.7 ClassificationNodeQuery</li> <li>8.2.8 ClassificationSchemeQuery</li> <li>8.2.9 RegistryPackageQuery</li> </ul>	$ \begin{array}{r}     41 \\     42 \\     44 \\     46 \\     59 \\     62 \\     64 \\     67 \\     69 \\     74 \\     75 \\     77 \\ \end{array} $
109 110 111 112 113 114 115 116 117 118 119 120		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{r}     41 \\     42 \\     44 \\     46 \\     59 \\     62 \\     64 \\     67 \\     69 \\     74 \\     75 \\     77 \\     78 \\ \end{array} $
109 110 111 112 113 114 115 116 117 118 119 120 121		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li> <li>8.2.1 FilterQuery</li> <li>8.2.2 RegistryObjectQuery</li> <li>8.2.3 RegistryEntryQuery</li> <li>8.2.4 AssociationQuery</li> <li>8.2.5 AuditableEventQuery</li> <li>8.2.6 ClassificationQuery</li> <li>8.2.7 ClassificationNodeQuery</li> <li>8.2.8 ClassificationSchemeQuery</li> <li>8.2.9 RegistryPackageQuery</li> <li>8.2.10 ExtrinsicObjectQuery</li> <li>8.2.11 OrganizationQuery</li> <li>8.2.12 ServiceQuery</li> </ul>	$ \begin{array}{r}     41 \\     42 \\     44 \\     46 \\     59 \\     62 \\     64 \\     67 \\     69 \\     74 \\     75 \\     77 \\     78 \\     82 \end{array} $
109 110 111 112 113 114 115 116 117 118 119 120 121 122		<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{c}     41 \\     42 \\     44 \\     46 \\     59 \\     62 \\     64 \\     67 \\     69 \\     74 \\     75 \\     77 \\     78 \\     82 \\     84 \\ \end{array} $
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123		<ul> <li>8.1.1 Query Response Options</li></ul>	$ \begin{array}{c}    $
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125	8.2	<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{c}    $
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126	8.2	<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$\begin{array}{c} 41 \\ 42 \\ 44 \\ 46 \\ 59 \\ 62 \\ 64 \\ 67 \\ 69 \\ 74 \\ 75 \\ 77 \\ 78 \\ 82 \\ 84 \\ 88 \\ 92 \\ 92 \\ 92 \end{array}$
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127	8.2	<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{c}    $
109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126	8.2	<ul> <li>8.1.1 Query Response Options</li> <li>Filter Query Support</li></ul>	$ \begin{array}{c}    $

130		8.3.5 RegistryObject Queries	95	
131		8.3.6 RegistryEntry Queries		
132		8.3.7 Classification Queries		
133		8.3.8 Association Queries		
134		8.3.9 Package Queries		
135		8.3.10 ExternalLink Queries	98	
136		8.3.11 Audit Trail Queries		
137	8.4	Content Retrieval	98	
138		8.4.1 Identification Of Content Payloads	98	
139		8.4.2 GetContentResponse Message Structure		
140	9 Regis	stry Security		100
141	9.1	Security Concerns	100	
142	9.2	Integrity of Registry Content	100	
143		9.2.1 Message Payload Signature		
144		9.2.2 Payload Signature Requirements	101	
145	9.3	Authentication	103	
146		9.3.1 Message Header Signature	103	
147	9.4	Key Distribution and KeyInfo Element	105	
148	9.5	Confidentiality	106	
149		9.5.1 On-the-wire Message Confidentiality	106	
150		9.5.2 Confidentiality of Registry Content	106	
151	9.6	Authorization		
152		9.6.1 Actions		
153	9.7	Access Control	107	
154	Appendix			109
155	A.1	Registry Service Abstract Specification		
156	A.2	Registry Service SOAP Binding	109	
157	Appendix	B ebXML Registry Schema Definitions	•••••	110
158	B.1	RIM Schema	110	
159	B.2	Query Schema		
160	B.3	Registry Services Interface Schema		
161	B.4	Examples of Instance Documents	110	
162	Appendix	C Interpretation of UML Diagrams	•••••	111
163	C.1	UML Class Diagram		
164	C.2	UML Sequence Diagram	111	
165	Appendix	D SQL Query	•••••	112
166	D.1	SQL Query Syntax Specification	112	
167	D.2	Non-Normative BNF for Query Syntax Grammar		
168	D.3	Relational Schema For SQL Queries	114	
169	Appendix	E Non-normative Content Based Ad Hoc Queries	•••••	115
170	E.1	Automatic Classification of XML Content	115	
171	E.2	Index Definition	115	
172	E.3	Example Of Index Definition		
173	E.4	Proposed XML Definition		
174	E.5	Example of Automatic Classification	116	
175	Appendix	F Security Implementation Guideline	•••••	117
176	F.1	Security Concerns	117	

177	F.2	Authentication	
178	F.3	Authorization	118
179	F.4	Registry Bootstrap	
180	F.5	Content Submission – Client Responsibility	
181	F.6	Content Submission – Registry Responsibility	119
182	F.7	Content Delete/Deprecate - Client Responsibility	
183	F.8	Content Delete/Deprecate - Registry Responsibility	
184	F.9	Using ds:KeyInfo Field	119
185	Appendix	G Native Language Support (NLS)	
186	G.1	Definitions.	
187		G.1.1 Coded Character Set (CCS):	
188		G.1.2 Character Encoding Scheme (CES):	
189		G.1.3 Character Set (charset):	
190	G.2	NLS And Request / Response Messages	
191	G.3	NLS And Storing of RegistryObject	
192		G.3.1 Character Set of <i>LocalizedString</i>	
193		G.3.2 Language Information of <i>LocalizedString</i>	
194	G.4	NLS And Storing of Repository Items	
195		G.4.1 Character Set of Repository Items	
196		G.4.2 Language information of repository item	
197	Appendix	<b>H</b> Registry Profile	
198	10	References	
199	11	Disclaimer	
200	12	Contact Information	
201	13	Copyright Statement	
202			

# 203 Table of Figures

204	Figure 1: Actor Relationships	
205	Figure 2: ebXML Registry Service Architecture	
206	Figure 3: The Abstract ebXML Registry Service	
207	Figure 4: A Concrete ebXML Registry Service	
208	Figure 5: Registry Architecture Supports Flexible Topologies	
209	Figure 6: Life Cycle of a Repository Item	
210	Figure 7: Submit Objects Sequence Diagram	
211	Figure 8: Update Objects Sequence Diagram	
212	Figure 9: Add Slots Sequence Diagram	
213	Figure 10: Remove Slots Sequence Diagram	
214	Figure 11: Approve Objects Sequence Diagram	
215	Figure 12: Deprecate Objects Sequence Diagram	
216	Figure 13: Remove Objects Sequence Diagram	
217	Figure 14: Submit Ad Hoc Query Sequence Diagram	
218	Figure 15: Example ebRIM Binding	
219	Figure 16: ebRIM Binding for RegistryObjectQuery	
220	Figure 17: ebRIM Binding for RegistryEntryQuery	59
221	Figure 18: ebRIM Binding for AssociationQuery	
222	Figure 19: ebRIM Binding for AuditableEventQuery	
223	Figure 20: ebRIM Binding for ClassificationQuery	
224	Figure 21: ebRIM Binding for ClassificationNodeQuery	
225	Figure 22: ebRIM Binding for ClassificationSchemeQuery	74
226	Figure 23: ebRIM Binding for RegistryPackageQuery	
227	Figure 24: ebRIM Binding for ExtrinsicObjectQuery	77
228	Figure 25: ebRIM Binding for OrganizationQuery	
229	Figure 26: ebRIM Binding for ServiceQuery	
230	Figure 27: The Clause Structure	
231		

# 232 **Table of Tables**

233	Table 1: Registry Users	
234	Table 2: LifeCycle Manager Summary	
235	Table 3: Query Manager	
236	Table 4: RegistryClient Summary	
237	Table 5 Submit Objects Error Handling	
238	Table 6: Update Objects Error Handling	
239	Table 7: Approve Objects Error Handling	
240	Table 8: Deprecate Objects Error Handling	
241	Table 9: Remove Objects Error Handling	
242	Table 10: Path Filter Expressions for Use Cases	
243	Table 11: Default Access Control Policies	
244		

# 245 **3 Introduction**

# **3.1 Summary of Contents of Document**

- This document defines the interface to the ebXML Registry Services as well as interactionprotocols, message definitions and XML schema.
- 249 A separate document, ebXML Registry Information Model [ebRIM], provides information on
- the types of metadata that are stored in the Registry as well as the relationships among the various metadata classes.

# 252 **3.2 General Conventions**

- 253 The following conventions are used throughout this document:
- 254 UML diagrams are used as a way to concisely describe concepts. They are not intended to
- 255 convey any specific Implementation or methodology requirements.
- 256 The term "repository item" is used to refer to an object that has resides in a repository for storage
- and safekeeping (e.g., an XML document or a DTD). Every repository item is described in the
- 258 Registry by a RegistryObject instance.
- The term "RegistryEntry" is used to refer to an object that provides metadata about a repository item.
- 261 Capitalized Italic words are defined in the ebXML Glossary.
- 262 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD
- NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in REC 2110 [Pro07]
- interpreted as described in RFC 2119 [Bra97].

# 265 **3.3 Audience**

- 266 The target audience for this specification is the community of software developers who are:
- Implementers of ebXML Registry Services
- Implementers of ebXML Registry Clients
- 269 Related Documents
- 270 The following specifications provide some background and related information to the reader:
- a) *ebXML Registry Information Model* [ebRIM]
- b) *ebXML Message Service Specification* [ebMS]
- 273 c) *ebXML Business Process Specification Schema* [ebBPSS]
- d) *ebXML Collaboration-Protocol Profile and Agreement Specification* [ebCPP]

# 275 **4 Design Objectives**

### 276 **4.1 Goals**

- 277 The goals of this version of the specification are to:
- Communicate functionality of Registry services to software developers
- Specify the interface for Registry clients and the Registry
- Provide a basis for future support of more complete ebXML Registry requirements
- Be compatible with other ebXML specifications

## 282 **4.2 Caveats and Assumptions**

- 283 This version of the Registry Services Specification is the second in a series of phased
- deliverables. Later versions of the document will include additional capability as deemed
- appropriate by the OASIS/ebXML Registry Technical Committee. It is assumed that:
- Interoperability requirements dictate that at least one of the normative interfaces as referenced inthis specification must be supported.
- All access to the Registry content is exposed via the interfaces defined for the Registry Services.
- The Registry makes use of a Repository for storing and retrieving persistent information required by the Registry Services. This is an implementation detail that will not be discussed further in this specification.

# 293 **5 System Overview**

## 294 5.1 What The ebXML Registry Does

295 The ebXML Registry provides a set of services that enable sharing of information between

interested parties for the purpose of enabling business process integration between such parties

based on the ebXML specifications. The shared information is maintained as objects in a

repository and managed by the ebXML Registry Services defined in this document.

# 299 **5.2 How The ebXML Registry Works**

300 This section describes at a high level some use cases illustrating how Registry clients may make

- 301 use of Registry Services to conduct B2B exchanges. It is meant to be illustrative and not
- 302 prescriptive.

303 The following scenario provides a high level textual example of those use cases in terms of

304 interaction between Registry clients and the Registry. It is not a complete listing of the use cases

that could be envisioned. It assumes for purposes of example, a buyer and a seller who wish to

306 conduct B2B exchanges using the RosettaNet PIP3A4 Purchase Order business protocol. It is

307 assumed that both buyer and seller use the same Registry service provided by a third party. Note

308 that the architecture supports other possibilities (e.g. each party uses its own private Registry).

#### 309 **5.2.1 Schema Documents Are Submitted**

310 A third party such as an industry consortium or standards group submits the necessary schema

- 311 documents required by the RosettaNet PIP3A4 Purchase Order business protocol with the
- 312 Registry using the LifeCycleManager service of the Registry described in Section 7.3.

#### 313 5.2.2 Business Process Documents Are Submitted

- 314 A third party, such as an industry consortium or standards group, submits the necessary business
- 315 process documents required by the RosettaNet PIP3A4 Purchase Order business protocol with

the Registry using the LifeCycleManager service of the Registry described in Section 7.3.

## 317 **5.2.3 Seller's Collaboration Protocol Profile Is Submitted**

318 The seller publishes its Collaboration Protocol Profile or CPP as defined by [ebCPP] to the

Registry. The CPP describes the seller, the role it plays, the services it offers and the technical

320 details on how those services may be accessed. The seller classifies their Collaboration Protocol

321 Profile using the Registry's flexible Classification capabilities.

#### 322 5.2.4 Buyer Discovers The Seller

323 The buyer browses the Registry using Classification schemes defined within the Registry using a

324 Registry Browser GUI tool to discover a suitable seller. For example the buyer may look for all

325 parties that are in the Automotive Industry, play a seller role, support the RosettaNet PIP3A4

- 326 process and sell Car Stereos.
- 327 The buyer discovers the seller's CPP and decides to engage in a partnership with the seller.

#### 328 **5.2.5 CPA is Established**

- 329 The buyer unilaterally creates a Collaboration Protocol Agreement or CPA as defined by
- 330 [ebCPP] with the seller using the seller's CPP and their own CPP as input. The buyer proposes a
- trading relationship to the seller using the unilateral CPA. The seller accepts the proposed CPA
- and the trading relationship is established.
- 333 Once the seller accepts the CPA, the parties may begin to conduct B2B transactions as defined
- 334 by [ebMS].

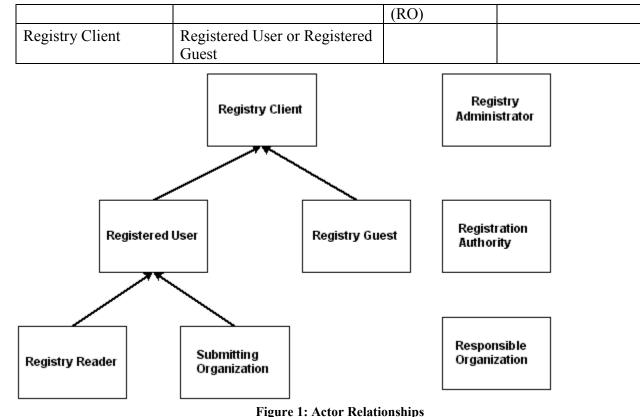
# 335 5.3 Registry Users

336 We describe the actors who use the registry from the point of view of security and analyze the

- 337 security concerns of the registry below. This analysis leads up to the security requirements for
- version 2.0. Some of the actors are defined in Section 9.7. Note that the same entity may
- 339 represent different actors. For example, a Registration Authority and Registry Administrator may
- 340 have the same identity.
- 341

Table 1: Registry Users

Actor	Function	<b>ISO/IEC</b> 11179	Comments
RegistrationAuthority	Hosts the RegistryObjects	Registration Authority (RA)	
Registry Administrator	Evaluates and enforces registry security policy. Facilitates definition of the registry security policy.		MAY have the same identity as Registration Authority
Registered User	Has a contract with the Registration Authority and MUST be authenticated by Registration Authority.		The contract could be a ebXML CPA or some other form of contract.
Registry Guest	Has no contract with Registration Authority. Does not have to be authenticated for Registry access. Cannot change contents of the Registry (MAY be permitted to read some RegistryObjects.)		Note that a Registry Guest is not a Registry Reader.
Submitting Organization	A Registered User who does lifecycle operations on permitted RegistryObjects.	Submitting Organization (SO)	
Registry Reader	A Registered User who has only <i>read</i> access		
Responsible Organization	Creates Registry Objects	Responsible Organization	RO MAY have the same identity as SO



342

- 344 Note:
- 345 In the current version of the specification the following are true.
- 346 A Submitting Organization and a Responsible Organization are the same.
- 347 Registration of a user happens out-of-band, i.e, by means not specified in this specification.
- 348 A Registry Administrator and Registration Authority are the same.

# **5.4 Where the Registry Services May Be Implemented**

The Registry Services may be implemented in several ways including, as a public web site, as a private web site, hosted by an ASP or hosted by a VPN provider.

## 352 **5.5 Implementation Conformance**

- 353 An implementation is a *conforming* ebXML Registry if the implementation meets the conditions
- in Section 5.5.1. An implementation is a conforming ebXML Registry Client if the
- implementation meets the conditions in Section 5.5.2. An implementation is a conforming
- ebXML Registry and a conforming ebXML Registry Client if the implementation conforms to
- 357 the conditions of Section 5.5.1 and Section 5.5.2. An implementation shall be a conforming
- 358 ebXML Registry, a conforming ebXML Registry Client, or a conforming ebXML Registry and
- 359 Registry Client.

#### 360 **5.5.1 Conformance as an ebXML Registry**

- 361 An implementation conforms to this specification as an ebXML Registry if it meets the 362 following conditions:
- 363 1. Conforms to the ebXML Registry Information Model [ebRIM].
- 364 2. Supports the syntax and semantics of the Registry Interfaces and Security Model.
- 365 3. Supports the defined ebXML Registry Schema (Appendix B).
- 366 4. Optionally supports the syntax and semantics of Section 8.3, SQL Query Support.

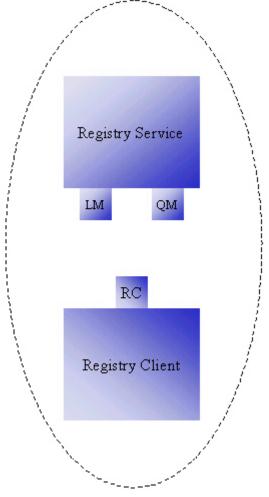
#### 367 **5.5.2 Conformance as an ebXML Registry Client**

- An implementation conforms to this specification, as an ebXML Registry Client if it meets thefollowing conditions:
- 370 1. Supports the ebXML CPA and bootstrapping process.
- 371 2. Supports the syntax and the semantics of the Registry Client Interfaces.
- 372 3. Supports the defined ebXML Error Message DTD.
- 373 4. Supports the defined ebXML Registry Schema (Appendix B).

374

# 375 6 ebXML Registry Architecture

- The ebXML Registry architecture consists of an ebXML Registry Service and ebXML Registry
- 377 Clients. The ebXML Registry Service provides the methods for managing a repository. An
- 378 ebXML Registry Client is an application used to access the Registry.



379 380

Figure 2: ebXML Registry Service Architecture

# 381 6.1 Registry Service Described

- 382 The ebXML Registry Service is comprised of a robust set of interfaces designed to
- fundamentally manage the objects and inquiries associated with the ebXML Registry. The two primary interfaces for the Registry Service consist of:
- A Life Cycle Management interface that provides a collection of methods for managing
   objects within the Registry.
- A Query Management Interface that controls the discovery and retrieval of information from the Registry.
- 389 A registry client program utilizes the services of the registry by invoking methods on one of the
- 390 above interfaces defined by the Registry Service. This specification defines the interfaces
- 391 exposed by the Registry Service (Sections 6.4 and 6.5) as well as the interface for the Registry
- 392 Client (Section 6.6).

## 393 6.2 Abstract Registry Service

- 394 The architecture defines the ebXML Registry as an abstract registry service that is defined as:
- 395 1. A set of interfaces that must be supported by the registry.
- 396 2. The set of methods that must be supported by each interface.
- 397 3. The parameters and responses that must be supported by each method.
- 398 The abstract registry service neither defines any specific implementation for the ebXML
- 399 Registry, nor does it specify any specific protocols used by the registry. Such implementation
- 400 details are described by concrete registry services that realize the abstract registry service.
- 401 The abstract registry service (Figure 3) shows how an abstract ebXML Registry must provide
- 402 two key functional interfaces called QueryManager<sup>1</sup> (QM) and LifeCycleManager<sup>2</sup>
- 403 (LM).



404 405

Figure 3: The Abstract ebXML Registry Service

406 Appendix A provides hyperlinks to the abstract service definition in the Web Service Description
 407 Language (WSDL) syntax.

# 408 6.3 Concrete Registry Services

- 409 The architecture allows the abstract registry service to be mapped to one or more concrete
- 410 registry services defined as:
- Implementations of the interfaces defined by the abstract registry service.
- Bindings of these concrete interfaces to specific communication protocols.
- 413 This specification describes two concrete bindings for the abstract registry service:
- 414 A SOAP binding using the HTTP protocol
- 415 An ebXML Messaging Service (ebMS) binding
- 416 A registry may implement one or both of the concrete bindings for the abstract registry service as
- 417 shown in Figure 4.
- 418

<sup>&</sup>lt;sup>1</sup> Known as ObjectQueryManager in V1.0

<sup>&</sup>lt;sup>2</sup> Known as ObjectManager in V1.0

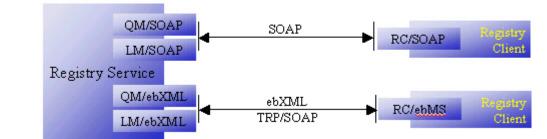


Figure 4: A Concrete ebXML Registry Service

Figure 4 shows a concrete implementation of the abstract ebXML Registry (RegistryService) on 421

422 the left side. The RegistryService provides the QueryManager and LifeCycleManager interfaces

423 available with multiple protocol bindings (SOAP and ebMS).

- 424 Figure 4 also shows two different clients of the ebXML Registry on the right side. The top client
- 425 uses SOAP interface to access the registry while the lower client uses ebMS interface. Clients
- use the appropriate concrete interface within the RegistryService service based upon their 426
- 427 protocol preference.

#### 428 6.3.1 SOAP Binding

#### 429 6.3.1.1 WSDL Terminology Primer

430 This section provides a brief introduction to Web Service Description Language (WSDL) since

the SOAP binding is described using WSDL syntax. WSDL provides the ability to describe a 431

- 432 web service in abstract as well as with concrete bindings to specific protocols. In WSDL, an
- 433 abstract service consists of one or more port types or end-points. Each port type consists
- 434 of a collection of operations. Each operation is defined in terms of messages that define
- 435 what data is exchanged as part of that operation. Each message is typically defined in terms of
- 436 elements within an XML Schema definition.
- 437 An abstract service is not bound to any specific protocol (e.g. SOAP). In WSDL, an abstract
- 438 service may be used to define a concrete service by binding it to a specific protocol. This binding
- 439 is done by providing a binding definition for each abstract port type that defines additional
- 440 protocols specific details. Finally, a concrete service definition is defined as a collection of
- ports, where each port simply adds address information such as a URL for each concrete port. 441

#### 442 6.3.1.2 Concrete Binding for SOAP

- 443 This section assumes that the reader is somewhat familiar with SOAP and WSDL. The SOAP 444 binding to the ebXML Registry is defined as a web service description in WSDL as follows:
- 445 A single service element with name "RegistryService" defines the concrete SOAP binding for the registry service. 446
- 447 The service element includes two port definitions, where each port corresponds with one of 448 the interfaces defined for the abstract registry service. Each port includes an HTTP URL for accessing that port. 449
- 450 Each port definition also references a binding element, one for each interface defined in the WSDL for the abstract registry service. 451 452 453

<service name = "RegistryService">

463 The complete WSDL description for the SOAP binding can be obtained via a hyperlink in464 Appendix A.

#### 465 6.3.2 ebXML Message Service Binding

#### 466 **6.3.2.1** Service and Action Elements

- When using the ebXML Messaging Services Specification, ebXML Registry Service elementscorrespond to Messaging Service elements as follows:
- The value of the Service element in the MessageHeader is an ebXML Registry Service
   interface name (e.g., "LifeCycleManager"). The type attribute of the Service element should
   have a value of "ebXMLRegistry".
- The value of the Action element in the MessageHeader is an ebXML Registry Service method name (e.g., "submitObjects").

```
475 <eb:Service eb:type="ebXMLRegistry">LifeCycleManger</eb:Service>
```

- 476 <eb:Action>submitObjects</eb:Action>
- 477

461 462

- 478 Note that the above allows the Registry Client only one interface/method pair per message. This
- implies that a Registry Client can only invoke one method on a specified interface for a givenrequest to a registry.
- 481 6.3.2.2 Synchronous and Asynchronous Responses
- 482 All methods on interfaces exposed by the registry return a response message.
- 483 Asynchronous response
- When a message is sent asynchronously, the Registry will return two response messages. The
  first message will be an immediate response to the request and does not reflect the actual
  response for the request. This message will contain:
- MessageHeader;
- RegistryResponse element with empty content (e.g., NO AdHocQueryResponse);
- 489 status attribute with value **Unavailable**.
- 490 The Registry delivers the actual Registry response element with non-empty content
- 491 asynchronously at a later time. The delivery is accomplished by the Registry invoking the
- 492 onResponse method on the RegistryClient interface as implemented by the registry client
- 493 application. The onResponse method includes a RegistryResponse element as shown below:
- MessageHeader;
- 495 RegistryResponse element including;
- 496 Status attribute (Success, Failure);

497 – Optional RegistryErrorList.

#### 498 Synchronous response

- 499 When a message is sent synchronously, the Message Service Handler will hold open the
- 500 communication mechanism until the Registry returns a response. This message will contain:
- 501 MessageHeader;
- RegistryResponse element including;
- 503 Status attribute (Success, Failure);
- 504 Optional RegistryErrorList.

#### 505 6.3.2.3 ebXML Registry Collaboration Profiles and Agreements

506 The ebXML CPP specification [ebCPP] defines a Collaboration-Protocol Profile (CPP) and a 507 Collaboration-Protocol Agreement (CPA) as mechanisms for two parties to share information 508 regarding their respective business processes. That specification assumes that a CPA has been 500 agreed to by both parties in order for them to approach in P2P interactions

- agreed to by both parties in order for them to engage in B2B interactions.
- 510 This specification does not mandate the use of a CPA between the Registry and the Registry
- 511 Client. However if the Registry does not use a CPP, the Registry shall provide an alternate

512 mechanism for the Registry Client to discover the services and other information provided by a

- 513 CPP. This alternate mechanism could be a simple URL.
- 514 The CPA between clients and the Registry should describe the interfaces that the Registry and
- 515 the client expose to each other for Registry-specific interactions. The definition of the Registry
- 516 CPP template and a Registry Client CPP template are beyond the scope of this document.

# 517 6.4 LifeCycleManager Interface

518 This is the interface exposed by the Registry Service that implements the object life cycle

519 management functionality of the Registry. Its methods are invoked by the Registry Client. For

520 example, the client may use this interface to submit objects, to classify and associate objects and

521 to deprecate and remove objects. For this specification the semantic meaning of submit, classify,

- 522 associate, deprecate and remove is found in [ebRIM].
- 523

 Table 2: LifeCycle Manager Summary

Method Summary of LifeCycleManager			
RegistryResponse	approveObjects (ApproveObjectsRequest req) Approves one or more previously submitted objects.		
RegistryResponse	deprecateObjects (DeprecateObjectsRequest req) Deprecates one or more previously submitted objects.		
RegistryResponse	<b>removeObjects</b> ( <u>RemoveObjectsRequest</u> req) Removes one or more previously submitted objects from the Registry.		
RegistryResponse	<b>submitObjects</b> (SubmitObjectsRequest req) Submits one or more objects and possibly related metadata such as Associations and Classifications.		
RegistryResponse	updateObiects(UpdateObiectsRequest req)		

	Updates one or more previously submitted objects.	
	addSlots (AddSlotsRequest req) Add slots to one or more registry entries.	
RegistryResponse         removeSlots         req)           Remove specified slots from one or more registry e		

## 524 6.5 QueryManager Interface

525 This is the interface exposed by the Registry that implements the Query management service of

526 the Registry. Its methods are invoked by the Registry Client. For example, the client may use this

527 interface to perform browse and drill down queries or ad hoc queries on registry content.

528

#### Table 3: Query Manager

# Method Summary of QueryManager

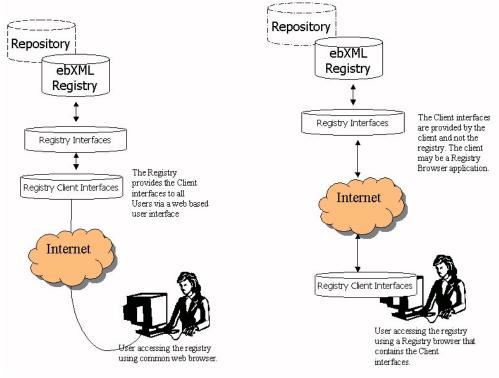
RegistryResponsesubmitAdhocQuery(AdhocQueryRequestreq)Submit an ad hoc query request.

## 529 6.6 Registry Clients

#### 530 6.6.1 Registry Client Described

531 The Registry Client interfaces may be local to the registry or local to the user. Figure 5 depicts 532 the two possible topologies supported by the registry architecture with respect to the Registry 533 and Registry Clients. The picture on the left side shows the scenario where the Registry provides 534 a web based "thin client" application for accessing the Registry that is available to the user using 535 a common web browser. In this scenario the Registry Client interfaces reside across the Internet and are local to the Registry from the user's view. The picture on the right side shows the 536 537 scenario where the user is using a "fat client" Registry Browser application to access the registry. 538 In this scenario the Registry Client interfaces reside within the Registry Browser tool and are 539 local to the Registry from the user's view. The Registry Client interfaces communicate with the 540 Registry over the Internet in this scenario.

- 541 A third topology made possible by the registry architecture is where the Registry Client
- 542 interfaces reside in a server side business component such as a Purchasing business component.
- 543 In this topology there may be no direct user interface or user intervention involved. Instead, the
- 544 Purchasing business component may access the Registry in an automated manner to select
- 545 possible sellers or service providers based on current business needs.



546 547

Figure 5: Registry Architecture Supports Flexible Topologies

## 548 6.6.2 Registry Communication Bootstrapping

549 Before a client can access the services of a Registry, there must be some communication 550 bootstrapping between the client and the registry. The most essential aspect of this bootstrapping 551 process is for the client to discover addressing information (e.g. an HTTP URL) to each of the 552 concrete service interfaces of the Registry. The client may obtain the addressing information by 553 discovering the ebXML Registry in a public registry such as UDDI or within another ebXML 554 Registry.

- In case of SOAP binding, all the info needed by the client (e.g. Registry URLs) is available
   in a WSDL description for the registry. This WSDL conforms to the template WSDL
   description in Appendix A.1. This WSDL description may be discovered in a public registry
   such as UDDI.
- In case of ebMS binding, the information exchange between the client and the registry may be accomplished in a registry specific manner, which may involve establishing a CPA
- between the client and the registry. Once the information exchange has occurred the Registry
- and the client will have addressing information (e.g. URLs) for the other party.

#### **563 6.6.2.1 Communication Bootstrapping for SOAP Binding**

564 Each ebXML Registry must provide a WSDL description for its RegistryService as defined by

- 565 Appendix A.1. A client uses the WSDL description to determine the address information of the
- 566 RegistryService in a protocol specific manner. For example the SOAP/HTTP based ports of the
- 567 RegistryService may be accessed via a URL specified in the WSDL for the registry.
- 568 The use of WSDL enables the client to use automated tools such as a WSDL compiler to
- 569 generate stubs that provide access to the registry in a language specific manner.

- 570 At minimum, any client may access the registry over SOAP/HTTP using the address information
- 571 within the WSDL, with minimal infrastructure requirements other than the ability to make
- 572 synchronous SOAP call to the SOAP based ports on the RegistryService.

#### 573 6.6.2.2 Communication Bootstrapping for ebXML Message Service

574 Since there is no previously established CPA between the Registry and the RegistryClient, the 575 client must know at least one Transport-specific communication address for the Registry. This

576 communication address is typically a URL to the Registry, although it could be some other type

- 577 of address such as an email address. For example, if the communication used by the Registry is
- 578 HTTP, then the communication address is a URL. In this example, the client uses the Registry's
- 579 public URL to create an implicit CPA with the Registry. When the client sends a request to the
- Registry, it provides a URL to itself. The Registry uses the client's URL to form its version of an implicit CPA with the client. At this point a session is established within the Registry. For the
- duration of the client's session with the Registry, messages may be exchanged bidirectionally as
- 582 duration of the client's session with the Registry, messages may be exchanged bidired 583 required by the interaction protocols defined in this specification
- required by the interaction protocols defined in this specification.

### 584 6.6.3 RegistryClient Interface

585 This is the principal interface implemented by a Registry client. The client provides this interface

586 when creating a connection to the Registry. It provides the methods that are used by the Registry

587 to deliver asynchronous responses to the client. Note that a client need not provide a

- 588 RegistryClient interface if the [CPA] between the client and the registry does not support
- asynchronous responses.
- 590 The registry sends all asynchronous responses to operations via the onResponse method.
- 591

#### Table 4: RegistryClient Summary

# Method Summary of RegistryClient

void **onResponse** (<u>RegistryResponse</u> resp) Notifies client of the response sent by registry to previously submitted request.

#### 592 6.6.4 Registry Response

593 The RegistryResponse is a common class defined by the Registry interface that is used by the 594 registry to provide responses to client requests.

# 595 6.7 Interoperability Requirements

#### 596 6.7.1 Client Interoperability

597 The architecture requires that any ebXML compliant registry client can access any ebXML

598 compliant registry service in an interoperable manner. An ebXML Registry may implement any

599 number of protocol bindings from the set of normative bindings (currently ebXML TRP and

600 SOAP/HTTP) defined in this proposal. The support of additional protocol bindings is optional.

#### 601 6.7.2 Inter-Registry Cooperation

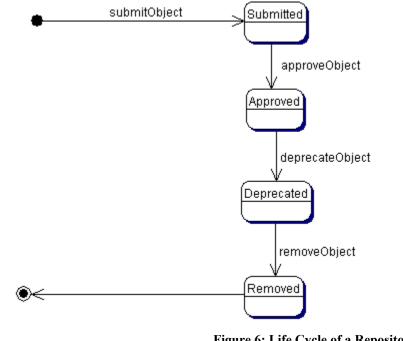
- 602 This version of the specification does not preclude ebXML Registries from cooperating with
- 603 each other to share information, nor does it preclude owners of ebXML Registries from
- registering their ebXML registries with other registry systems, catalogs, or directories.
- 605 Examples include:
- An ebXML Registry that serves as a registry of ebXML Registries.
- A non-ebXML Registry that serves as a registry of ebXML Registries.
- Cooperative ebXML Registries, where multiple ebXML registries register with each other in order to form a federation.

#### Life Cycle Management Service 610 7

- 611 This section defines the LifeCycleManagement service of the Registry. The Life Cycle
- 612 Management Service is a sub-service of the Registry service. It provides the functionality
- 613 required by RegistryClients to manage the life cycle of repository items (e.g. XML documents
- required for ebXML business processes). The Life Cycle Management Service can be used with 614
- 615 all types of repository items as well as the metadata objects specified in [ebRIM] such as
- 616 Classification and Association.
- 617 The minimum-security policy for an ebXML registry is to accept content from any client if a
- 618 certificate issued by a Certificate Authority recognized by the ebXML registry digitally signs the 619 content

#### 7.1 Life Cycle of a Repository Item 620

- 621 The main purpose of the LifeCycleManagement service is to manage the life cycle of repository
- items. Figure 6 shows the typical life cycle of a repository item. Note that the current version of 622
- 623 this specification does not support Object versioning. Object versioning will be added in a future
- 624 version of this specification



#### Figure 6: Life Cycle of a Repository Item

#### 7.2 RegistryObject Attributes 627

625 626

628 A repository item is associated with a set of standard metadata defined as attributes of the

629 RegistryObject class and its sub-classes as described in [ebRIM]. These attributes reside outside

630 of the actual repository item and catalog descriptive information about the repository item. XML

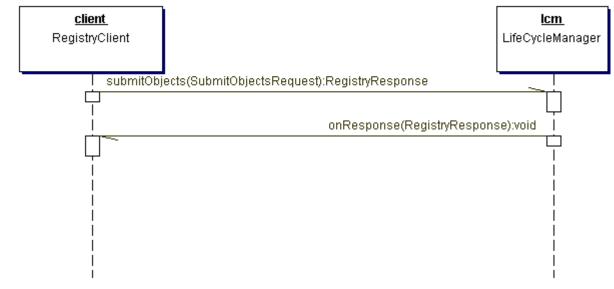
- elements called ExtrinsicObject and other elements (See Appendix B.1 for details) encapsulate 631
- 632 all object metadata attributes defined in [ebRIM] as XML attributes.

# 633 **7.3 The Submit Objects Protocol**

634 This section describes the protocol of the Registry Service that allows a RegistryClient to submit

one or more repository items to the repository using the LifeCycleManager on behalf of a

636 Submitting Organization. It is expressed in UML notation as described in Appendix C.



637 638

Figure 7: Submit Objects Sequence Diagram

- 639 For details on the schema for the Business documents shown in this process refer to Appendix B.
- 640 The SubmitObjectRequest message includes a LeafRegistryObjectList element.
- 641 The LeafRegistryObjectList element specifies one or more ExtrinsicObjects or other
- 642 RegistryEntries such as Classifications, Associations, ExternalLinks, or Packages.

643 An ExtrinsicObject element provides required metadata about the content being submitted to the

Registry as defined by [ebRIM]. Note that these standard ExtrinsicObject attributes are separate

- 645 from the repository item itself, thus allowing the ebXML Registry to catalog objects of any
- 646 object type.

#### 647 **7.3.1 Universally Unique ID Generation**

As specified by [ebRIM], all objects in the registry have a unique id. The id must be a

649 Universally Unique Identifier (UUID) and must conform to the to the format of a URN that 650 specifies a DCE 128 bit UUID as specified in [UUID].

#### 651 (e.g. urn:uuid:a2345678-1234-1234-123456789012)

652 The registry usually generates this id. The client may optionally supply the id attribute for

653 submitted objects. If the client supplies the id and it conforms to the format of a URN that

654 specifies a DCE 128 bit UUID then the registry assumes that the client wishes to specify the id

for the object. In this case, the registry must honour a client-supplied id and use it as the id

attribute of the object in the registry. If the id is found by the registry to not be globally unique,

- the registry must raise the error condition: InvalidIdError.
- If the client does not supply an id for a submitted object then the registry must generate a

universally unique id. Whether the client generates the id or whether the registry generates it, itmust be generated using the DCE 128 bit UUID generation algorithm as specified in [UUID].

#### 661 7.3.2 ID Attribute And Object References

662 The id attribute of an object may be used by other objects to reference the first object. Such 663 references are common both within the SubmitObjectsRequest as well as within the registry. 664 Within a SubmitObjectsRequest, the id attribute may be used to refer to an object within the 665 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the 666 SubmitObjectsRequest that needs to be referred to within the request document may be assigned an id by the submitter so that it can be referenced within the request. The submitter may give the 667 object a proper uuid URN, in which case the id is permanently assigned to the object within the 668 669 registry. Alternatively, the submitter may assign an arbitrary id (not a proper unid URN) as long 670 as the id is unique within the request document. In this case the id serves as a linkage mechanism within the request document but must be ignored by the registry and replaced with a registry 671 672 generated id upon submission.

673 When an object in a SubmitObjectsRequest needs to reference an object that is already in the

registry, the request must contain an ObjectRef element whose id attribute is the id of the object in the registry. This id is by definition a proper unid URN. An ObjectRef may be viewed as a

676 proxy within the reguest for an object that is in the registry.

#### 677 **7.3.3 Audit Trail**

678 The RS must create AuditableEvents object with eventType Created for each RegistryObject 679 created via a SubmitObjects request.

#### 680 **7.3.4 Submitting Organization**

- The RS must create an Association of type SubmitterOf between the submitting organization and
- 682 each RegistryObject created via a SubmitObjects request. (Submitting organization is
- determined from the organization attribute of the User who submits a SubmitObjects request.)

#### 684 7.3.5 Error Handling

- A SubmitObjects request is atomic and either succeeds or fails in total. In the event of success,
- the registry sends a RegistryResponse with a status of "Success" back to the client. In the event
- of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In
- the event of an immediate response for an asynchronous request, the registry sends a
- 689 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
- 690 more Error conditions are raised in the processing of the submitted objects. Warning messages
- do not result in failure of the request. The following business rules apply:
- 692

#### **Table 5 Submit Objects Error Handling**

Business Rule	Applies To	<b>Error/Warning</b>
ID not unique	All Classes	Error
Not authorized	All Classes	Error

Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion are set by the RS, and ignored if supplied.	All Classes	Warning

#### 693 7.3.6 Sample SubmitObjectsRequest

The following example shows several different use cases in a single SubmitObjectsRequest. It 694 695 does not show the complete SOAP or [ebMS] Message with the message header and additional 696 payloads in the message for the repository items.

697 A SubmitObjectsRequest includes a RegistryObjectList which contains any number of objects 698 that are being submitted. It may also contain any number of ObjectRefs to link objects being  $\begin{array}{c} 6990\\ 7001\\ 7023\\ 7070\\ 7030\\ 7000$ submitted to objects already within the registry.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<SubmitObjectsRequest
 xmlns = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
 xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0 file:///C:/osws/ebxmlrr-
spec/misc/schema/rim.xsd urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0
file:///C:/osws/ebxmlrr-spec/misc/schema/rs.xsd"
 xmlns:rim = "urn:oasis:names:tc:ebxml-regrep:rim:xsd:2.0"
 xmlns:rs = "urn:oasis:names:tc:ebxml-regrep:registry:xsd:2.0"
  <rim:LeafRegistryObjectList>
   <!--
   The following 3 objects package specified ExtrinsicObject in specified
     RegistryPackage, where both the RegistryPackage and the ExtrinsicObject are
     being submitted
     -->
   <rim:RegistryPackage id = "acmePackage1" >
     <rim:Name>
       <rim:LocalizedString value = "RegistryPackage #1"/>
     </rim.Name>
     <rim:Description>
       <rim:LocalizedString value = "ACME's package #1"/>
     </rim:Description>
   </rim:RegistryPackage>
   <rim:ExtrinsicObject id = "acmeCPP1"
     <rim:Name>
       <rim:LocalizedString value = "Widget Profile" />
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "ACME's profile for selling widgets" />
     </rim:Description>
   </rim:ExtrinsicObject>
   <rim:Association id = "acmePackage1-acmeCPP1-Assoc" associationType = "Packages" sourceObject</pre>
= "acmePackage1" targetObject = "acmeCPP1" />
   <!--
     The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
     Where the RegistryPackage is being submitted and the ExtrinsicObject is
     already in registry
      -->
```

```
<rim:RegistryPackage id = "acmePackage2" >
     <rim:Name>
       <rim:LocalizedString value = "RegistryPackage #2"/>
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "ACME's package #2"/>
     </rim:Description>
   </rim:RegistryPackage>
   <rim:ObjectRef id = "urn:uuid:a2345678-1234-1234-123456789012"/>
   <rim:Association id = "acmePackage2-alreadySubmittedCPP-Assoc" associationType = "Packages"
sourceObject = "acmePackage2" targetObject = "urn:uuid:a2345678-1234-1234-123456789012"/>
   <!--
     The following 3 objects package specified ExtrinsicObject in specified RegistryPackage,
     where the RegistryPackage and the ExtrinsicObject are already in registry
   <rim:ObjectRef id = "urn:uuid:b2345678-1234-1234-123456789012"/>
   <rim:ObjectRef id = "urn:uuid:c2345678-1234-1234-123456789012"/>
   <!-- id is unspecified implying that registry must create a uuid for this object -->
   <rim:Association associationType = "Packages" sourceObject = "urn:uuid:b2345678-1234-1234-
123456789012" targetObject = "urn:uuid:c2345678-1234-1234-123456789012"/>
   <!--
     The following 3 objects externally link specified ExtrinsicObject using
     specified ExternalLink, where both the ExternalLink and the ExtrinsicObject
     are being submitted
     -->
   <rim:ExternalLink id = "acmeLink1" >
     <rim:Name>
      <rim:LocalizedString value = "Link #1"/>
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "ACME's Link #1"/>
     </rim:Description>
   </rim:ExternalLink>
   <rim:ExtrinsicObject id = "acmeCPP2" >
     <rim:Name>
       <rim:LocalizedString value = "Sprockets Profile" />
     </rim:Name>
    <rim:Description>
       <rim:LocalizedString value = "ACME's profile for selling sprockets"/>
     </rim:Description>
   </rim:ExtrinsicObject>
   <rim:Association id = "acmeLink1-acmeCPP2-Assoc" associationType = "ExternallyLinks"</pre>
sourceObject = "acmeLink1" targetObject = "acmeCPP2"/>
   <!--
     The following 2 objects externally link specified ExtrinsicObject using specified
     ExternalLink, where the ExternalLink is being submitted and the ExtrinsicObject
     is already in registry. Note that the targetObject points to an ObjectRef in a
    previous line
   <rim:ExternalLink id = "acmeLink2">
     <rim:Name>
      <rim:LocalizedString value = "Link #2"/>
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "ACME's Link #2"/>
     </rim:Description>
   </rim:ExternalLink>
```

84490 8885123345567 8855567788855677888556778885567788661

866 867 868

884 885 885

```
<rim:Association id = "acmeLink2-alreadySubmittedCPP-Assoc" associationType =
"ExternallyLinks" sourceObject = "acmeLink2" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>
    <!--
     The following 3 objects externally identify specified ExtrinsicObject using specified
     ExternalIdentifier, where the ExternalIdentifier is being submitted and the
     ExtrinsicObject is already in registry. Note that the targetObject points to an
     ObjectRef in a previous line
      -->
   <rim:ClassificationScheme id = "DUNS-id" isInternal="false" nodeType="UniqueCode" >
     <rim:Name>
       <rim:LocalizedString value = "DUNS"/>
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "This is the DUNS scheme"/>
     </rim:Description>
   </rim:ClassificationScheme>
   <rim:ExternalIdentifier id = "acmeDUNSId" identificationScheme="DUNS-id" value =</pre>
"13456789012">
    <rim:Name>
       <rim:LocalizedString value = "DUNS" />
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "DUNS ID for ACME"/>
     </rim:Description>
   </rim:ExternalIdentifier>
   <rim:Association id = "acmeDUNSId-alreadySubmittedCPP-Assoc" associationType =</pre>
"ExternallyIdentifies" sourceObject = "acmeDUNSId" targetObject = "urn:uuid:a2345678-1234-1234-
123456789012"/>
   <!--
     The following show submission of a brand new classification scheme in its entirety
   <rim:ClassificationScheme id = "Geography-id" isInternal="true" nodeType="UniqueCode" >
     <rim:Name>
       <rim:LocalizedString value = "Geography"/>
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "This is a sample Geography scheme"/>
     </rim:Description>
     <rim:ClassificationNode id = "NorthAmerica-id" parent = "Geography-id" code =</pre>
"NorthAmerica" >
       <rim:ClassificationNode id = "UnitedStates-id" parent = "NorthAmerica-id" code =</pre>
"UnitedStates" />
       <rim:ClassificationNode id = "Canada-id" parent = "NorthAmerica-id" code = "Canada" />
     </rim:ClassificationNode>
     <rim:ClassificationNode id = "Asia-id" parent = "Geography-id" code = "Asia" >
       <rim:ClassificationNode id = "Japan-id" parent = "Asia-id" code = "Japan" >
         <rim:ClassificationNode id = "Tokyo-id" parent = "Japan-id" code = "Tokyo" />
       </rim:ClassificationNode>
     </rim:ClassificationNode>
   </rim:ClassificationScheme>
   <!--
     The following show submission of a Automotive sub-tree of ClassificationNodes that
     gets added to an existing classification scheme named 'Industry'
     that is already in the registry
      -->
   <rim:ObjectRef id = "urn:uuid:d2345678-1234-1234-123456789012"/>
    <rim:ClassificationNode id = "automotiveNode" parent = "urn:uuid:d2345678-1234-1234-
123456789012">
     <rim:Name>
       <rim:LocalizedString value = "Automotive" />
```

```
</rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "The Automotive sub-tree under Industry scheme"/>
     </rim:Description>
   </rim:ClassificationNode>
   <rim:ClassificationNode id = "partSuppliersNode" parent = "automotiveNode">
     <rim:Name>
       <rim:LocalizedString value = "Parts Supplier" />
     </rim:Name>
     <rim:Description>
       <rim:LocalizedString value = "The Parts Supplier node under the Automotive node" />
     </rim:Description>
   </rim:ClassificationNode>
   <rim:ClassificationNode id = "engineSuppliersNode" parent = "automotiveNode">
     <rim:Name>
       <rim:LocalizedString value = "Engine Supplier" />
     </rim:Name>
    <rim:Description>
       <rim:LocalizedString value = "The Engine Supplier node under the Automotive node" />
     </rim:Description>
   </rim:ClassificationNode>
   <!--
     The following show submission of 2 Classifications of an object that is already in
     the registry using 2 ClassificationNodes. One ClassificationNode
     is being submitted in this request (Japan) while the other is already in the registry.
      -->
    <rim:Classification id = "japanClassification" classifiedObject = "urn:uuid:a2345678-1234-</pre>
1234-123456789012" classificationNode = "Japan-id">
     <rim:Description>
       <rim:LocalizedString value = "Classifies object by /Geography/Asia/Japan node"/>
     </rim:Description>
    </rim:Classification>
   <rim:Classification id = "classificationUsingExistingNode" classifiedObject =</pre>
"urn:uuid:a2345678-1234-1234-123456789012" classificationNode = "urn:uuid:e2345678-1234-1234-
123456789012">
     <rim:Description>
       <rim:LocalizedString value = "Classifies object using a node in the registry" />
     </rim:Description>
   </rim:Classification>
   <rim:ObjectRef id = "urn:uuid:e2345678-1234-1234-123456789012"/>
  </rim:LeafRegistryObjectList>
</SubmitObjectsRequest>
```

# 939 7.4 The Update Objects Protocol

940 This section describes the protocol of the Registry Service that allows a Registry Client to update 941 one or more existing Registry Items in the registry on behalf of a Submitting Organization. It is 942 expressed in UML notation as described in Appendix C.

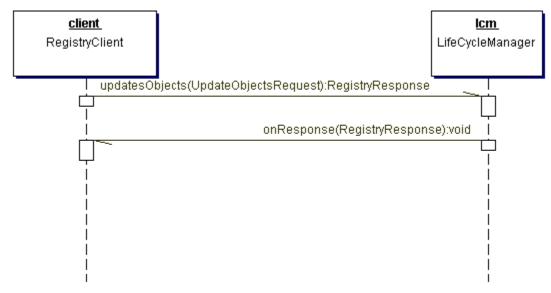


Figure 8: Update Objects Sequence Diagram

For details on the schema for the Business documents shown in this process refer to Appendix B.

946 The UpdateObjectsRequest message includes a LeafRegistryObjectList element. The

947 LeafRegistryObjectList element specifies one or more RegistryObjects. Each object in the list

948 must be a current RegistryObject. RegistryObjects must include all attributes, even those the

949 user does not intend to change. A missing attribute is interpreted as a request to set that attribute 950 to NULL.

#### 951 **7.4.1 Audit Trail**

952 The RS must create AuditableEvents object with eventType Updated for each RegistryObject953 updated via an UpdateObjects request.

#### 954 **7.4.2 Submitting Organization**

The RS must maintain an Association of type SubmitterOf between the submitting organization

and each RegistryObject updated via an UpdateObjects request. If an UpdateObjects request is

accepted from a different submitting organization, then the RS must delete the original

- association object and create a new one. Of course, the AccessControlPolicy may prohibit this
- sort of update in the first place. (Submitting organization is determined from the organization
- 960 attribute of the User who submits an UpdateObjects request.)

#### 961 7.4.3 Error Handling

962 An UpdateObjects request is atomic and either succeeds or fails in total. In the event of success,

963 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event

of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In

965 the event of an immediate response for an asynchronous request, the registry sends a

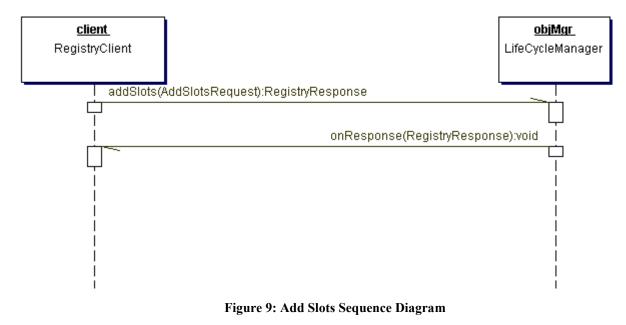
- RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or
- 967 more Error conditions are raised in the processing of the updated objects. Warning messages do
- not result in failure of the request. The following business rules apply:

<b>Business Rule</b>	<b>Applies</b> To	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	All Classes	Error
Referenced object not found.	Association, Classification, ClassificationNode, Organization	Error
Associations not allowed to connect to deprecated objects.	Association	Error
Object status, majorVersion and minorVersion cannot be changed via the UpdateObjects protocol, ignored if supplied.	All Classes	Warning
RegistryEntries with stability = "Stable" should not be updated.	All Classes	Warning

#### Table 6: Update Objects Error Handling

# 970 7.5 The Add Slots Protocol

- 971 This section describes the protocol of the Registry Service that allows a client to add slots to a
- 972 previously submitted registry entry using the LifeCycleManager. Slots provide a dynamic
- 973 mechanism for extending registry entries as defined by [ebRIM].

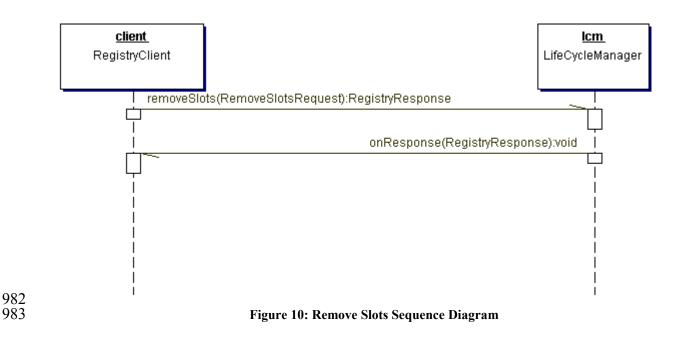


- 976 In the event of success, the registry sends a RegistryResponse with a status of "success" back to
- 977 the client. In the event of failure, the registry sends a RegistryResponse with a status of "failure"
- back to the client.

974 975

# 979 **7.6 The Remove Slots Protocol**

This section describes the protocol of the Registry Service that allows a client to remove slots toa previously submitted registry entry using the LifeCycleManager.

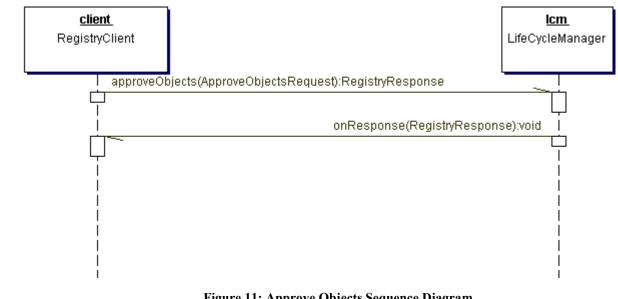


#### 7.7 The Approve Objects Protocol 984

985 This section describes the protocol of the Registry Service that allows a client to approve one or more previously submitted repository items using the LifeCycleManager. Once a repository item 986

is approved it will become available for use by business parties (e.g. during the assembly of new 987

988 CPAs and Collaboration Protocol Profiles).



989 990

**Figure 11: Approve Objects Sequence Diagram** 

991 For details on the schema for the business documents shown in this process refer to Appendix B.

#### 7.7.1 Audit Trail 992

993 The RS must create AuditableEvents object with eventType Approved for each RegistryObject 994 approved via an Approve Objects request.

#### 995 **7.7.2 Submitting Organization**

996 The RS must maintain an Association of type SubmitterOf between the submitting organization

997 and each RegistryObject updated via an ApproveObjects request. If an ApproveObjects request

998 is accepted from a different submitting organization, then the RS must delete the original

association object and create a new one. Of course, the AccessControlPolicy may prohibit this

1000 sort of ApproveObjects request in the first place. (Submitting organization is determined from

1001 the organization attribute of the User who submits an ApproveObjects request.)

#### 1002 7.7.3 Error Handling

1003 An ApproveObjects request is atomic and either succeeds or fails in total. In the event of success,

the registry sends a RegistryResponse with a status of "Success" back to the client. In the event

1005 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In

1006 the event of an immediate response for an asynchronous request, the registry sends a

1007 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or

1008 more Error conditions are raised in the processing of the object reference list. Warning messages

- 1009 do not result in failure of the request. The following business rules apply:
- 1010

 Table 7: Approve Objects Error Handling

Business Rule	Applies To	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	RegistryEntry Classes	Error
Only RegistryEntries may be "approved".	All Classes other than RegistryEntry classes	Error
Object status is already "Approved".	RegistryEntry Classes	Warning

# 1011 **7.8 The Deprecate Objects Protocol**

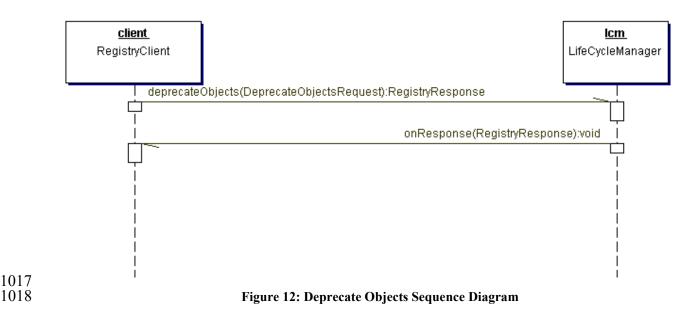
1012 This section describes the protocol of the Registry Service that allows a client to deprecate one or

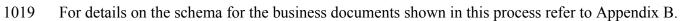
1013 more previously submitted repository items using the LifeCycleManager. Once an object is

1014 deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that

1015 object can be submitted. However, existing references to a deprecated object continue to function

1016 normally.





## 1020 7.8.1 Audit Trail

1021 The RS must create AuditableEvents object with eventType Deprecated for each RegistryObject 1022 deprecated via a Deprecate Objects request.

## 1023 7.8.2 Submitting Organization

1024 The RS must maintain an Association of type SubmitterOf between the submitting organization 1025 and each RegistryObject updated via a Deprecate Objects request. If a Deprecate Objects request 1026 is accepted from a different submitting organization, then the RS must delete the original 1027 association object and create a new one. Of course, the AccessControlPolicy may prohibit this 1028 sort of Deprecate Objects request in the first place. (Submitting organization is determined from

1029 the organization attribute of the User who submits a Deprecate Objects request.)

## 1030 7.8.3 Error Handling

A DeprecateObjects request is atomic and either succeeds or fails in total. In the event of success, the registry sends a RegistryResponse with a status of "Success" back to the client. In the event of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In the event of an immediate response for an asynchronous request, the registry sends a RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or more Error conditions are raised in the processing of the object reference list. Warning messages

- 1037 do not result in failure of the request. The following business rules apply:
- 1038

#### **Table 8: Deprecate Objects Error Handling**

Business Rule	Applies To	Error/Warning
Object not found	All Classes	Error
Not authorized	RegistrvEntrv	Error

	Classes	
Only RegistryEntries may be "deprecated".	All Classes other than RegistryEntry classes	Error
Object status is already "Deprecated".	RegistryEntry Classes	Warning

## 1039 **7.9 The Remove Objects Protocol**

1040 This section describes the protocol of the Registry Service that allows a client to remove one or 1041 more RegistryObject instances and/or repository items using the LifeCycleManager.

1042 The RemoveObjectsRequest message is sent by a client to remove RegistryObject instances

1043 and/or repository items. The RemoveObjectsRequest element includes an XML attribute called

1044 deletionScope which is an enumeration that can have the values as defined by the following

1045 sections.

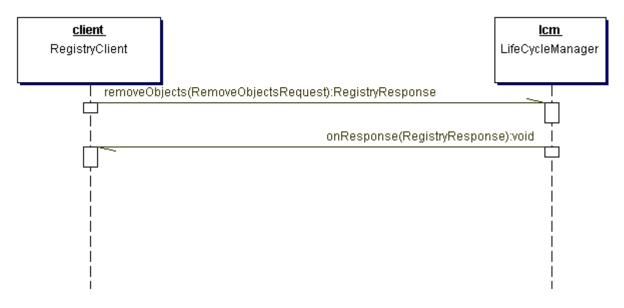
## 1046 **7.9.1 Deletion Scope DeleteRepositoryItemOnly**

1047 This deletionScope specifies that the request should delete the repository items for the specified 1048 registry entries but not delete the specified registry entries. This is useful in keeping references to

1049 the registry entries valid.

## 1050 7.9.2 Deletion Scope DeleteAll

- 1051 This deletionScope specifies that the request should delete both the RegistryObject and the
- 1052 repository item for the specified registry entries. Only if all references (e.g. Associations,
- 1053 Classifications, ExternalLinks) to a RegistryObject have been removed, can that RegistryObject
- 1054 then be removed using a RemoveObjectsRequest with deletionScope DeleteAll. Attempts to
- 1055 remove a RegistryObject while it still has references raises an error condition:
- 1056 InvalidRequestError.
- 1057 The remove object protocol is expressed in UML notation as described in Appendix C.



1059

Figure 13: Remove Objects Sequence Diagram

1060 For details on the schema for the business documents shown in this process refer to Appendix B.

## 1061 7.9.3 Error Handling

1062 A Remove Objects request is atomic and either succeeds or fails in total. In the event of success,

1063 the registry sends a RegistryResponse with a status of "Success" back to the client. In the event

1064 of failure, the registry sends a RegistryResponse with a status of "Failure" back to the client. In

1065 the event of an immediate response for an asynchronous request, the registry sends a

1066 RegistryResponse with a status of "Uavailable" back to the client. Failure occurs when one or

1067 more Error conditions are raised in the processing of the object reference list. Warning messages

- 1068 do not result in failure of the request. The following business rules apply:
- 1069

### Table 9: Remove Objects Error Handling

<b>Business Rule</b>	<b>Applies</b> To	<b>Error/Warning</b>
Object not found	All Classes	Error
Not authorized	RegistryObject Classes	Error

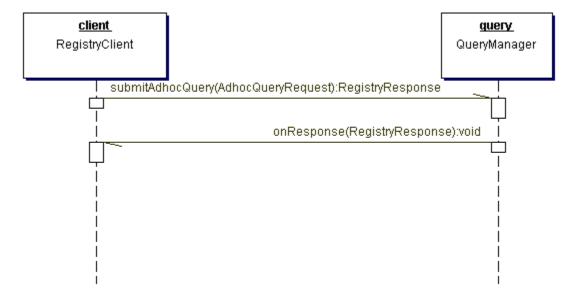
1070

# 1071 8 Query Management Service

- 1072 This section describes the capabilities of the Registry Service that allow a client
- 1073 (QueryManagerClient) to search for or query different kind of registry objects in the ebXML
- 1074 Registry using the QueryManager interface of the Registry. The Registry supports the following
- 1075 query capabilities:
- 1076 Filter Query
- 1077 SQL Query
- 1078 The Filter Query mechanism in Section 8.2 SHALL be supported by every Registry
- 1079 implementation. The SQL Query mechanism is an optional feature and MAY be provided by a
- 1080 registry implementation. However, if a vendor provides an SQL query capability to an ebXML
- 1081 Registry it SHALL conform to this document. As such this capability is a normative yet optional 1082 capability.
- In a future version of this specification, the W3C XQuery syntax may be considered as anotherquery syntax.
- 1085 The Registry will hold a self-describing capability profile that identifies all supported
- 1086 AdhocQuery options. This profile is described in Appendix H.

## 1087 8.1 Ad Hoc Query Request/Response

- 1088 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
- AdhocQueryRequest contains a subelement that defines a query in one of the supported Registryquery mechanisms.
- 1091 The QueryManager sends an AdhocQueryResponse either synchronously or asynchronously
- 1092 back to the client. The AdhocQueryResponse returns a collection of objects whose element type
- 1093 depends upon the responseOption attribute of the AdhocQueryRequest. These may be objects
- representing leaf classes in [ebRIM], references to objects in the registry as well as intermediate
- 1095 classes in [ebRIM] such as RegistryObject and RegistryEntry.
- Any errors in the query request messages are indicated in the corresponding query responsemessage.



1098 1099

#### Figure 14: Submit Ad Hoc Query Sequence Diagram

For details on the schema for the business documents shown in this process refer to AppendixB.2.

1103	
1104	<pre><element name="AdhocQueryRequest"></element></pre>
1105	<complextype></complextype>
1106	<sequence></sequence>
1107	<pre><element maxoccurs="1" minoccurs="1" ref="tns:ResponseOption"></element></pre>
1108	<pre><choice maxoccurs="1" minoccurs="1"></choice></pre>
1109	<element ref="tns:FilterQuery"></element>
1110	<pre><element ref="tns:SQLQuery"></element></pre>
1111	
1112	
1113	
1114	
1115	
1116	<pre><element name="AdhocQueryResponse"></element></pre>
1117	<complextype></complextype>
1118	<choice maxoccurs="1" minoccurs="1"></choice>
1119	<pre><element ref="tns:FilterQueryResult"></element></pre>
1120	<pre><element ref="tns:SQLQueryResult"></element></pre>
1121	
1122	
1123	
1124	

- 1125 8.1.1 Query Response Options
- 1126 Purpose
- 1127 A QueryManagerClient may specify what an ad hoc query must return within an
- 1128 AdhocQueryResponse using the ResponseOption element of the AdHocQueryRequest.
- 1129 ResponseOption element has an attribute "returnType" and its values are:

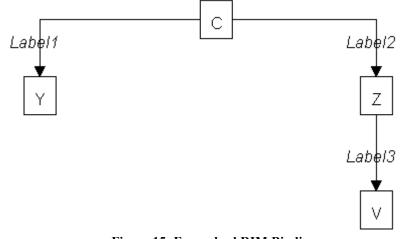
- ObjectRef This option specifies that the AdhocQueryResponse may contain a collection of
   ObjectRef XML elements as defined in [ebRIM Schema]. Purpose of this option is to return
   just the identifiers of the registry objects.
- RegistryObject This option specifies that the AdhocQueryResponse may contain a collection of RegistryObject XML elements as defined in [ebRIM Schema]. In this case all attributes of the registry objects are returned (objectType, name, description, ...) in addition to id attribute.
- RegistryEntry This option specifies that the AdhocQueryResponse may contain a collection of RegistryEntry or RegistryObject XML elements as defined in [ebRIM Schema], which correspond to RegistryEntry or RegistryObject attributes.
- LeafClass This option specifies that the AdhocQueryResponse may contain a collection of XML elements that correspond to leaf classes as defined in [ebRIM Schema].
- LeafClassWithRepositoryItem This option specifies that the AdhocQueryResponse may contain a collection of ExtrinsicObject XML elements as defined in [ebRIM Schema] accompanied with their repository items or RegistryEntry or RegistryObject and their attributes. Linking of ExtrinsicObject and its repository item is done via contentURI as explained in Section 8.4 -Content Retrieval.
- 1147 ResponseOption element also has an attribute "returnComposedObjects". It specifies whether or 1148 not the whole hierarchy of composed objects are returned with the registry objects.
- 1149 If "returnType" is higher then the RegistryObject option, then the highest option that satisfies the
- 1150 query is returned. This can be illustrated with a case when OrganizationQuery is asked to return
- 1151 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass
- option instead. If OrganizationQuery is asked to retrieve a RegistryEntry as a return type then
- 1153 RegistryObject metadata will be returned.

#### 1154 Definition 1155 1156 <complexType name="ResponseOptionType"> 1157 <attribute name="returnType" default="RegistryObject"> 1158 <simpleType> 1159 <restriction base="NMTOKEN"> 1160 <enumeration value="ObjectRef" /> 1161 <enumeration value="RegistryObject" /> 1162 <enumeration value="RegistryEntry" /> 1163 <enumeration value="LeafClass" /> 1164 <enumeration value="LeafClassWithRepositoryItem" /> 1165 </restriction> 1166 </simpleType> 1167 </attribute> 1168 <attribute name="returnComposedObjects" type="boolean" default="false" /> 1169 </complexType> 1170 <element name="ResponseOption" type="tns:ResponseOptionType" /> 1171

## 1172 8.2 Filter Query Support

- 1173 FilterQuery is an XML syntax that provides simple query capabilities for any ebXML
- 1174 conforming Registry implementation. Each query alternative is directed against a single class
- defined by the ebXML Registry Information Model (ebRIM). There are two types of filter
- 1176 queries depending on which classes are queried on.

- Firstly, there are RegistryObjectQuery and RegistryEntryQuery. They allow for generic
- 1178 queries that might return different subclasses of the class that is queried on. The result of
- such a query is a set of XML elements that correspond to instances of any class that satisfies
- 1180 the responseOption defined previously in Section 8.1.1. An example might be that
- 1181 RegistryObjectQuery with responseOption LeafClass will return all attributes of all instances
- that satisfy the query. This implies that response might return XML elements that correspond
- 1183 to classes like ClassificationScheme, RegistryPackage, Organization and Service.
- Secondly, FilterQuery supports queries on selected ebRIM classes in order to define the exact traversals of these classes. Responses to these queries are accordingly constrained.
- 1186 A client submits a FilterQuery as part of an AdhocQueryRequest. The QueryManager sends an
- 1187 AdhocQueryResponse back to the client, enclosing the appropriate FilterQueryResult specified
- herein. The sequence diagrams for AdhocQueryRequest and AdhocQueryResponse are specifiedin Section 8.1.
- 1190 Each FilterQuery alternative is associated with an ebRIM Binding that identifies a hierarchy of
- 1191 classes derived from a single class and its associations with other classes as defined by ebRIM.
- 1192 Each choice of a class pre-determines a virtual XML document that can be queried as a tree. For
- 1193 example, let C be a class, let Y and Z be classes that have direct associations to C, and let V be a
- 1194 class that is associated with Z. The ebRIM Binding for C might be as in Figure 15



1195 1196

Figure 15: Example ebRIM Binding

1197 Label1 identifies an association from C to Y, Label2 identifies an association from C to Z, and

1198 Label3 identifies an association from Z to V. Labels can be omitted if there is no ambiguity as to

- 1199 which ebRIM association is intended. The name of the query is determined by the root class, i.e.
- this is an ebRIM Binding for a CQuery. The Y node in the tree is limited to the set of Y instances
- 1201 that are linked to C by the association identified by Label1. Similarly, the Z and V nodes are
- 1202 limited to instances that are linked to their parent node by the identified association.

1203 Each FilterQuery alternative depends upon one or more class filters, where a class filter is a

1204 restricted predicate clause over the attributes of a single class. Class methods that are defined in

ebRIM and that return simple types constitute "visible attributes" that are valid choices for

1206 predicate clauses. Names of those attributes will be same as name of the corresponding method

- just without the prefix 'get'. For example, in case of "getLevelNumber" method the
- 1208 corresponding visible attribute is "levelNumber". The supported class filters are specified in
- 1209 Section 8.2.13 and the supported predicate clauses are defined in Section 8.2.14. A FilterQuery

- 1210 will be composed of elements that traverse the tree to determine which branches satisfy the
- 1211 designated class filters, and the query result will be the set of instances that support such a 1212 branch.
- 1213 In the above example, the CQuery element will have three subelements, one a CFilter on the C
- 1214 class to eliminate C instances that do not satisfy the predicate of the CFilter, another a YFilter on
- 1215 the Y class to eliminate branches from C to Y where the target of the association does not satisfy
- 1216 the YFilter, and a third to eliminate branches along a path from C through Z to V. The third
- 1217 element is called a branch element because it allows class filters on each class along the path
- from C to V. In general, a branch element will have subelements that are themselves class filters,
- 1219 other branch elements, or a full-blown query on the class in the path.
- 1220 If an association from a class C to a class Y is one-to-zero or one-to-one, then at most one
- branch, filter or query element on Y is allowed. However, if the association is one-to-many, then
- 1222 multiple branch, filter or query elements are allowed. This allows one to specify that an instance
- 1223 of C must have associations with multiple instances of Y before the instance of C is said to
- 1224 satisfy the branch element.
- 1225 The FilterQuery syntax is tied to the structures defined in ebRIM. Since ebRIM is intended to be
- stable, the FilterQuery syntax is stable. However, if new structures are added to the ebRIM, then
- 1227 the FilterQuery syntax and semantics can be extended at the same time. Also, FilterQuery syntax
- 1228 follows the inheritance hierarchy of ebRIM, which means that subclass queries inherit from their
- 1229 respective superclass queries. Structures of XML elements that match the ebRIM classes are
- 1230 explained in [ebRIM Schema]. Names of Filters, Queries and Branches correspond to names in
- 1231 ebRIM whenever possible.
- 1232 The ebRIM Binding paragraphs in Sections 8.2.2 through 8.2.12 below identify the virtual
- 1233 hierarchy for each FilterQuery alternative. The Semantic Rules for each query alternative specify
- 1234 the effect of that binding on query semantics.

## 1235 8.2.1 FilterQuery

## 1236 Purpose

To identify a set of queries that traverse specific registry class. Each alternative assumes a
specific binding to ebRIM. The status is a success indication or a collection of warnings and/or
exceptions.

<pre><element name="FilterQuery"></element></pre>
<complextype></complextype>
<choice maxoccurs="1" minoccurs="1"></choice>
<pre><element ref="tns:RegistryObjectQuery"></element></pre>
<pre><element ref="tns:RegistryEntryQuery"></element></pre>
<pre><element ref="tns:AssociationQuery"></element></pre>
<pre><element ref="tns:AuditableEventQuery"></element></pre>
<pre><element ref="tns:ClassificationQuery"></element></pre>
<pre><element ref="tns:ClassificationNodeQuery"></element></pre>
<pre><element ref="tns:ClassificationSchemeQuery"></element></pre>
<pre><element ref="tns:RegistryPackageQuery"></element></pre>
<pre><element ref="tns:ExtrinsicObjectQuery"></element></pre>
<pre><element ref="tns:OrganizationQuery"></element></pre>
<pre><element ref="tns:ServiceQuery"></element></pre>

1256	
1257	
1258	
1259	
1260	<pre><element name="FilterQueryResult"></element></pre>
1261	<complextype></complextype>
1262	<pre><choice maxoccurs="1" minoccurs="1"></choice></pre>
1263	<pre><element ref="tns:RegistryObjectQueryResult"></element></pre>
1264	<pre><element ref="tns:RegistryEntryQueryResult"></element></pre>
1265	<pre><element ref="tns:AssociationQueryResult"></element></pre>
1266	<pre><element ref="tns:AuditableEventQueryResult"></element></pre>
1267	<pre><element ref="tns:ClassificationQueryResult"></element></pre>
1268	<pre><element ref="tns:ClassificationNodeQueryResult"></element></pre>
1269	<pre><element ref="tns:ClassificationSchemeQueryResult"></element></pre>
1270	<pre><element ref="tns:RegistryPackageQueryResult"></element></pre>
1271	<pre><element ref="tns:ExtrinsicObjectQueryResult"></element></pre>
1272	<pre><element ref="tns:OrganizationQueryResult"></element></pre>
1273	<pre><element ref="tns:ServiceQueryResult"></element></pre>
1274	
1275	
1276	
1277	

#### 1278 Semantic Rules

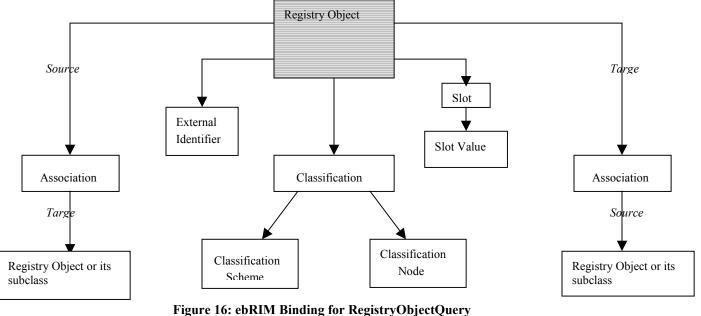
- 1279 1. The semantic rules for each FilterQuery alternative are specified in subsequent subsections.
- Semantic rules specify the procedure for implementing the evaluation of Filter Queries.
   Implementations do not necessarily have to follow the same procedure provided that the same effect is achieved.
- Each FilterQueryResult is a set of XML elements to identify each instance of the result set.
   Each XML attribute carries a value derived from the value of an attribute specified in the
   Registry Information Model [ebRIM Schema].
- 4. For each FilterQuery subelement there is only one corresponding FilterQueryResult
  subelement that must be returned as a response. Class name of the FilterQueryResult
  subelement has to match the class name of the FilterQuery subelement.
- 1289 5. If a Filter, Branch or Query element for a class has no sub-elements then every persistent1290 instance of that class satisfies the Filter, Branch or Query.
- 6. If an error condition is raised during any part of the execution of a FilterQuery, then the status attribute of the XML RegistryResult is set to "failure" and no AdHocQueryResult element is returned; instead, a RegistryErrorList element must be returned with its highestSeverity element set to "error". At least one of the RegistryError elements in the RegistryErrorList will have its severity attribute set to "error".
- 1296 7. If no error conditions are raised during execution of a FilterQuery, then the status attribute of
  1297 the XML RegistryResult is set to "success" and an appropriate FilterQueryResult element
  1298 must be included. If a RegistryErrorList is also returned, then the highestSeverity attribute of
  1299 the RegistryErrorList is set to "warning" and the serverity attribute of each RegistryError is
  1300 set to "warning".

## 1301 8.2.2 RegistryObjectQuery

## 1302 Purpose

- 1303 To identify a set of registry object instances as the result of a query over selected registry
- 1304 metadata.

### 1305 ebRIM Binding



#### 1306

1307	Demition
1308	<complextype name="RegistryObjectQueryType"></complextype>
1309	<sequence></sequence>
1310	<element maxoccurs="1" minoccurs="0" ref="tns:RegistryObjectFilter"></element>
1311	<element maxoccurs="unbounded" minoccurs="0" ref="tns:ExternalIdentifierFilter"></element>
1312	<element maxoccurs="unbounded" minoccurs="0" ref="tns:AuditableEventQuery"></element>
1313	<element maxoccurs="1" minoccurs="0" ref="tns:NameBranch"></element>
1314	<element maxoccurs="1" minoccurs="0" ref="tns:DescriptionBranch"></element>
1315	<element maxoccurs="unbounded" minoccurs="0" ref="tns:ClassifiedByBranch"></element>
1316	<element maxoccurs="unbounded" minoccurs="0" ref="tns:SlotBranch"></element>
1317	<element maxoccurs="unbounded" minoccurs="0" ref="tns:SourceAssociationBranch"></element>
1318	<element maxoccurs="unbounded" minoccurs="0" ref="tns:TargetAssociationBranch"></element>
1319	
1320	
1321	<element name="RegistryObjectQuery" type="tns:RegistryObjectQueryType"></element>
1322	
1323	<complextype name="LeafRegistryObjectListType"></complextype>
1324	<choice maxoccurs="unbounded" minoccurs="0"></choice>
1325	<element ref="tns:ObjectRef"></element>
1326	<element ref="tns:Association"></element>
1327	<element ref="tns:AuditableEvent"></element>
1328	<element ref="tns:Classification"></element>
1329	<element ref="tns:ClassificationNode"></element>
1330	<element ref="tns:ClassificationScheme"></element>
1331	<element ref="tns:ExternalIdentifier"></element>
1332	<element ref="tns:ExternalLink"></element>
1333	<element ref="tns:ExtrinsicObject"></element>

1334	<element ref="tns:Organization"></element>
1335	<element ref="tns:RegistryPackage"></element>
1336	<element ref="tns:Service"></element>
1337	<element ref="tns:ServiceBinding"></element>
1338	<element ref="tns:SpecificationLink"></element>
1339	<element ref="tns:User"></element>
1340	
1341	
1342	complex 1 ype
1343	<complextype name="RegistryObjectListType"></complextype>
1343	<complex hand="RegistryObjectListType" type=""> <complexcontent></complexcontent></complex>
1344	1
	<extension base="tns:LeafRegistryObjectListType"></extension>
1346	<choice maxoccurs="unbounded" minoccurs="0"></choice>
1347	<element ref="tns:RegistryEntry"></element>
1348	<pre><element ref="tns:RegistryObject"></element></pre>
1349	
1350	
1351	
1352	
1353	<element name="RegistryObjectQueryResult" type="rim:RegistryObjectListType"></element>
1354	
1355	<complextype name="InternationalStringBranchType"></complextype>
1356	<sequence></sequence>
1357	<element maxoccurs="unbounded" minoccurs="0" ref="tns:LocalizedStringFilter"></element>
1358	
1359	
1360	
1361	<complextype name="AssociationBranchType"></complextype>
1362	<sequence></sequence>
1363	<pre><element maxoccurs="1" minoccurs="0" ref="tns:AssociationFilter"></element></pre>
1364	<choice maxoccurs="1" minoccurs="0"></choice>
1365	<element maxoccurs="1" minoccurs="0" ref="tns:ExternalLinkFilter"></element>
1366	<element maxoccurs="1" minoccurs="0" ref="tns:ExternalIdentifierFilter"></element>
1367	<element maxoccurs="1" minoccurs="0" ref="tns:RegistryObjectQuery"></element>
1368	<element maxoccurs="1" minoccurs="0" ref="tns:RegistryEntryQuery"></element>
1369	<pre><element maxoccurs="1" minoccurs="0" ref="tns:AssociationQuery"></element></pre>
1370	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ClassificationQuery"></element></pre>
1371	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ClassificationSchemeQuery"></element></pre>
1372	<pre><li>element ref="tns:ClassificationNodeQuery" minOccurs="0" maxOccurs="1" /&gt;</li></pre>
1373	<pre><li><li><li><li><li><li><li><li><li><li< td=""></li<></li></li></li></li></li></li></li></li></li></pre>
1374	<pre><element maxoccurs="1" minoccurs="0" ref="tns:AuditableEventQuery"></element></pre>
1375	<pre><li><li><li><li><li><li><li><li><li><li< td=""></li<></li></li></li></li></li></li></li></li></li></pre>
1376	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ExtrinsicObjectQuery"></element></pre>
1377	<pre><li><li><li><li><li><li><li><li><li><li< td=""></li<></li></li></li></li></li></li></li></li></li></pre>
1378	<pre><element maxoccurs="1" minoccurs="0" ref="tns:UserBranch"></element></pre>
1378	
1379	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ServiceBindingBranch"></element> <clement maxoccurs="1" minoccurs="0" ref="tms:ServiceBindingBranch"></clement></pre>
	<pre><element maxoccurs="1" minoccurs="0" ref="tns:SpecificationLinkBranch"></element></pre>
1381	
1382	
1383	
1384	<element name="SourceAssociationBranch" type="tns:AssociationBranchType"></element>
1385	<element name="TargetAssociationBranch" type="tns:AssociationBranchType"></element>
1386	
1387	<element name="ClassifiedByBranch"></element>
1388	<complextype></complextype>
1389	<sequence></sequence>
1390	<element maxoccurs="1" minoccurs="0" ref="tns:ClassificationFilter"></element>

1391	<element maxoccurs="1" minoccurs="0" ref="tns:ClassificationSchemeQuery"></element>
1392	<element maxoccurs="1" minoccurs="0" ref="tns:ClassificationNodeQuery"></element>
1393	sequence>
1394	
1395	
1396	
1397	<element name="SlotBranch"></element>
1398	<complextype></complextype>
1399	<sequence></sequence>
1400	<pre><element maxoccurs="1" minoccurs="0" ref="tns:SlotFilter"></element></pre>
1401	<pre><element maxoccurs="unbounded" minoccurs="0" ref="tns:SlotValueFilter"></element></pre>
1401	
1403	
1404	
1405	
1406	<element name="UserBranch"></element>
1407	<complextype></complextype>
1408	<sequence></sequence>
1409	<pre><element maxoccurs="1" minoccurs="0" ref="tns:UserFilter"></element></pre>
1410	
	<pre><element maxoccurs="1" minoccurs="0" ref="tns:PostalAddressFilter"></element></pre>
1411	<pre><element maxoccurs="unbounded" minoccurs="0" ref="tns:TelephoneNumberFilter"></element></pre>
1412	<element maxoccurs="unbounded" minoccurs="0" ref="tns:EmailAddressFilter"></element>
1413	<element maxoccurs="1" minoccurs="0" ref="tns:OrganizationQuery"></element>
1414	
1415	
1416	
1417	
1418	<complextype name="ServiceBindingBranchType"></complextype>
1419	
1419	<sequence></sequence>
	<element maxoccurs="1" minoccurs="0" ref="tns:ServiceBindingFilter"></element>
1421	<element maxoccurs="unbounded" minoccurs="0" ref="tns:SpecificationLinkBranch"></element>
1422	<element maxoccurs="1" minoccurs="0" ref="tns:ServiceBindingTargetBranch"></element>
1423	
1424	
1425	<pre><element name="ServiceBindingBranch" type="tns:ServiceBindingBranchType"></element></pre>
1426	<element name="ServiceBindingTargetBranch" type="tns:ServiceBindingBranchType"></element>
1427	
1428	<element name="SpecificationLinkBranch"></element>
1429	
	<complex type=""></complex>
1430	<sequence></sequence>
1431	<element maxoccurs="1" minoccurs="0" ref="tns:SpecificationLinkFilter"></element>
1432	<element maxoccurs="1" minoccurs="0" ref="tns:RegistryObjectQuery"></element>
1433	<element maxoccurs="1" minoccurs="0" ref="tns:RegistryEntryQuery"></element>
1434	
1435	
1436	
1437	
1121	

## 1438 Semantic Rules

- Let RO denote the set of all persistent RegistryObject instances in the Registry. The
   following steps will eliminate instances in RO that do not satisfy the conditions of the
   specified filters.
- a) If RO is empty then go to number 2 below.

1443 1444 1445	b)	If a RegistryObjectFilter is not specified then go to the next step; otherwise, let x be a registry object in RO. If x does not satisfy the RegistryObjectFilter, then remove x from RO. If RO is empty then continue to the next numbered rule.
1446 1447 1448 1449 1450 1451 1452	c)	If an ExternalIdentifierFilter element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one ExternalIdentifier instance, then remove x from RO; otherwise, treat each ExternalIdentifierFilter element separately as follows: Let EI be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are linked to x. If EI is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1453 1454 1455 1456	d)	If an AuditableEventQuery is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x doesn't have an auditable event that satisfy AuditableEventQuery as specified in Section 8.2.5 then remove x from RO. If RO is empty then continue to the next numbered rule.
1457 1458 1459 1460 1461 1462	e)	If a NameBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a name then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise treat NameBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by at least one of the LocalizedStrings that constitute the name of the registry object then remove x from RO. If RO is empty then continue to the next numbered rule.
1463 1464 1465 1466 1467 1468	f)	If a DescriptionBranch is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x does not have a name then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise treat DescriptionBranch as follows: If any LocalizedStringFilter that is specified is not satisfied by some of the LocalizedStrings that constitute the description of the registry object then remove x from RO. If RO is empty then continue to the next numbered rule.
1469 1470 1471 1472 1473 1474 1475 1476 1477 1478 1479 1480 1481 1482 1483 1484	g)	If a ClassifiedByBranch element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the classifiedObject of at least one Classification instance, then remove x from RO; otherwise, treat each ClassifiedByBranch element separately as follows: If no ClassificationFilter is specified within the ClassifiedByBranch, then let CL be the set of all Classification instances that have x as the classifiedObject; otherwise, let CL be the set of Classification instances that satisfy the ClassificationFilter and have x as the classifiedObject. If CL is empty, then remove x from RO and continue to the next numbered rule. Otherwise, if CL is not empty, and if a Classification instances in CL whose defining classification scheme satisfies the ClassificationSchemeQuery. If the new CL is empty, then remove x from RO and continue to the next numbered classification scheme satisfies the ClassificationNodeQuery is specified, then replace CL by the set of remaining Classification instances in CL for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery is specified, then replace CL by the set of remaining Classification instances in CL for which a classification node exists and for which that classification node satisfies the ClassificationNodeQuery. If the new CL is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1485 1486 1487 1488 1489 1490 1491 1492 1493 1494	h)	If a SlotBranch element is not specified, then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not linked to at least one Slot instance, then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise, treat each SlotBranch element separately as follows: If a SlotFilter is not specified within the SlotBranch, then let SL be the set of all Slot instances for x; otherwise, let SL be the set of Slot instances that satisfy the SlotFilter and are Slot instances for x. If SL is empty, then remove x from RO and continue to the next numbered rule. Otherwise, if SL remains not empty, and if a SlotValueFilter is specified SlotValueFilter is valid. If SL is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1495 1496 1497 1498 1499	i)	If a SourceAssociationBranch element is not specified then go to the next step; otherwise, let x be a remaining registry object in RO. If x is not the source object of at least one Association instance, then remove x from RO. If RO is empty then continue to the next numbered rule; otherwise, treat each SourceAssociationBranch element separately as follows:
1500 1501 1502 1503 1504		If no AssociationFilter is specified within the SourceAssociationBranch, then let AF be the set of all Association instances that have x as a source object; otherwise, let AF be the set of Association instances that satisfy the AssociationFilter and have x as the source object. If AF is empty, then remove x from RO.
1505 1506		If RO is empty then continue to the next numbered rule.
1507 1508 1509 1510 1511		If an ExternalLinkFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1512 1513 1514 1515		If an ExternalIdentifierFilter is specified within the SourceAssociationBranch, then let ROT be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1516 1517 1518 1519 1520 1521		If a RegistryObjectQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1522 1523 1524 1525 1526		If a RegistryEntryQuery is specified within the SourceAssociationBranch, then let ROT be the set of RegistryEntry instances that satisfy the RegistryEntryQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1520 1527 1528 1529 1530		If a ClassificationSchemeQuery is specified within the SourceAssociationBranch, then let ROT be the set of ClassificationScheme instances that satisfy the ClassificationSchemeQuery and are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule.

1531	
1532	If a ClassificationNodeQuery is specified within the SourceAssociationBranch, then let
1532	ROT be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1534	and are the target object of some element of AF. If ROT is empty, then remove x from
1535	RO. If RO is empty then continue to the next numbered rule.
1536	1 5
1537	If an OrganizationQuery is specified within the SourceAssociationBranch, then let ROT
1538	be the set of Organization instances that satisfy the OrganizationQuery and are the target
1539	object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty
1540	then continue to the next numbered rule.
1541	
1542	If an AuditableEventQuery is specified within the SourceAssociationBranch, then let
1543	ROT be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1544	the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1545	is empty then continue to the next numbered rule.
1546	
1547	If a RegistryPackageQuery is specified within the SourceAssociationBranch, then let
1548	ROT be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1549	are the target object of some element of AF. If ROT is empty, then remove x from RO. If
1550	RO is empty then continue to the next numbered rule.
1551	
1552	If an ExtrinsicObjectQuery is specified within the SourceAssociationBranch, then let
1553	ROT be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1554	the target object of some element of AF. If ROT is empty, then remove x from RO. If RO
1555	is empty then continue to the next numbered rule.
1556	
1557	If a ServiceQuery is specified within the SourceAssociationBranch, then let ROT be the
1558	set of Service instances that satisfy the ServiceQuery and are the target object of some
1559	element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue
1560	to the next numbered rule.
1561	

1562 If a UserBranch is specified within the SourceAssociationBranch then let ROT be the set 1563 of User instances that are the target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the 1564 1565 member of ROT. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove u from ROT. If ROT is empty, then remove x from 1566 RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter 1567 1568 element is specified within the UserBranch, and if the postal address of u does not satisfy 1569 that filter, then remove u from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are 1570 1571 specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by at least one of the telephone numbers of u then remove u from ROT. If ROT is empty, 1572 1573 then remove x from RO. If RO is empty then continue to the next numbered rule. If an 1574 OrganizationQuery element is specified within the UserBranch, then let o be the 1575 Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROT. 1576 1577 If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. 1578 1579 1580 If a ClassificationQuery is specified within the SourceAssociationBranch, then let ROT 1581 be the set of Classification instances that satisfy the ClassificationOuery and are the 1582 target object of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2). 1583 1584 1585 If a ServiceBindingBranch is specified within the SourceAssociationBranch, then let 1586 ROT be the set of ServiceBinding instances that are the target object of some element of 1587 AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule. Let sb be the member of ROT. If a ServiceBindingFilter element is 1588 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then 1589 1590 remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then 1591 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the 1592 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as 1593 follows:

1594 Let sb be a remaining service binding in ROT. Let SL be the set of all specification link instances sl that describe specification links of sb. If a SpecificationLinkFilter element is 1595 1596 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then 1597 remove sl from SL. If SL is empty then remove sb from ROT. If ROT is empty then 1598 remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl 1599 1600 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: 1601 Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for at least one registry object in RO, then remove sl from SL. If 1602 1603 SL is empty then remove sb from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryOuery element is 1604 1605 specified within the SpecificationLinkBranch then let sl be a remaining specification link 1606 in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the 1607 RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for at least one registry entry in RE, then remove sl from SL. If SL is empty then remove sb from 1608 1609 ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a ServiceBindingTargetBranch is specified within the 1610 1611 ServiceBindingBranch, then let SBT be the set of ServiceBinding instances that satisfy 1612 the ServiceBindingTargetBranch and are the target service binding of some element of 1613 ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from 1614 RO. If RO is empty then continue to the next numbered rule. 1615 1616 If a SpecificationLinkBranch is specified within the SourceAssociationBranch, then let ROT be the set of SpecificationLink instances that are the target object of some element 1617 of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the 1618 next numbered rule. Let sl be the member of ROT. If a SpecificationLinkFilter element is 1619 1620 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then 1621 continue to the next numbered rule. If a RegistryObjectQuery element is specified within 1622 the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat 1623 RegistryObjectQuery element as follows: Let RO be the result set of the 1624 1625 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROT. If ROT is empty then remove x from 1626 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery 1627 1628 element is specified within the SpecificationLinkBranch then let sl be a remaining specification link in ROT. Treat RegistryEntryQuery element as follows: Let RE be the 1629 1630 result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification 1631 link for at least one registry entry in RE, then remove sl from ROT. If ROT is empty then remove x from RO. If RO is empty then continue to the next numbered rule. 1632 1633 1634 If an AssociationQuery is specified within the SourceAssociationBranch, then let ROT be 1635 the set of Association instances that satisfy the AssociationQuery and are the target object 1636 of some element of AF. If ROT is empty, then remove x from RO. If RO is empty then continue to the next numbered rule (Rule 2). 1637 1638

1639	j)	If a TargetAssociationBranch element is not specified then go to the next step; otherwise,
1640		let x be a remaining registry object in RO. If x is not the target object of some
1641		Association instance, then remove x from RO. If RO is empty then continue to the next
1642		numbered rule; otherwise, treat each TargetAssociationBranch element separately as
1643		follows:
1644		
1645		If no AssociationFilter is specified within the TargetAssociationBranch, then let AF be
1646		the set of all Association instances that have x as a target object; otherwise, let AF be the
1647		set of Association instances that satisfy the AssociationFilter and have x as the target
1648		object. If AF is empty, then remove x from RO. If RO is empty then continue to the next
1649		numbered rule.
1650		
1651		If an ExternalLinkFilter is specified within the TargetAssociationBranch, then let ROS be
1652		the set of ExternalLink instances that satisfy the ExternalLinkFilter and are the source
1653		object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1654		then continue to the next numbered rule.
1655		
1656		If an ExternalIdentifierFilter is specified within the TargetAssociationBranch, then let
1657		ROS be the set of ExternalIdentifier instances that satisfy the ExternalIdentifierFilter and
1658		are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1659		RO is empty then continue to the next numbered rule.
1660		
1661		If a RegistryObjectQuery is specified within the TargetAssociationBranch, then let ROS
1662		be the set of RegistryObject instances that satisfy the RegistryObjectQuery and are the
1663		source object of some element of AF. If ROS is empty, then remove x from RO. If RO is
1664		empty then continue to the next numbered rule.
1665		
1666		If a RegistryEntryQuery is specified within the TargetAssociationBranch, then let ROS
1667		be the set of
1668		RegistryEntry instances that satisfy the RegistryEntryQuery and are the source object of
1669		some element of AF. If ROS is empty, then remove x from RO. If RO is empty then
1670		continue to the next numbered rule.
1671		
1672		If a ClassificationSchemeQuery is specified within the TargetAssociationBranch, then let
1673		ROS be the set of ClassificationScheme instances that satisfy the
1674		ClassificationSchemeQuery and are the source object of some element of AF. If ROS is
1675		empty, then remove x from RO. If RO is empty then continue to the next numbered rule.
1676		
1677		If a ClassificationNodeQuery is specified within the TargetAssociationBranch, then let
1678		ROS be the set of ClassificationNode instances that satisfy the ClassificationNodeQuery
1679		and are the source object of some element of AF. If ROS is empty, then remove x from
1680		RO. If RO is empty then continue to the next numbered rule.
1681		

1682	If an OrganizationQuery is specified within the TargetAssociationBranch, then let ROS
1683	be the set of Organization instances that satisfy the OrganizationQuery and are the source
1684	object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty
1685	then continue to the next numbered rule.
1686	
1687	If an AuditableEventQuery is specified within the TargetAssociationBranch, then let
1688	ROS be the set of AuditableEvent instances that satisfy the AuditableEventQuery and are
1689	the source object of some element of AF. If ROS is empty, then remove x from RO. If
1690	RO is empty then continue to the next numbered rule.
1691	
1692	If a RegistryPackageQuery is specified within the TargetAssociationBranch, then let
1693	ROS be the set of RegistryPackage instances that satisfy the RegistryPackageQuery and
1694	are the source object of some element of AF. If ROS is empty, then remove x from RO. If
1695	RO is empty then continue to the next numbered rule.
1696	
1697	If an ExtrinsicObjectQuery is specified within the TargetAssociationBranch, then let
1698	ROS be the set of ExtrinsicObject instances that satisfy the ExtrinsicObjectQuery and are
1699	the source object of some element of AF. If ROS is empty, then remove x from RO. If
1700	RO is empty then continue to the next numbered rule.
1701	No is empty then continue to the next numbered rule.
1702	If a ServiceQuery is specified within the TargetAssociationBranch, then let ROS be the
1702	set of Service instances that satisfy the ServiceQuery and are the source object of some
1703	element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue
1704	to the next numbered rule.
	to the next numbered rule.
1706	
1707	If a UserBranch is specified within the TargetAssociationBranch then let ROS be the set
1708	of User instances that are the source object of some element of AF. If ROS is empty, then
1709	remove x from RO. If RO is empty then continue to the next numbered rule. Let u be the
1710	member of ROS. If a UserFilter element is specified within the UserBranch, and if u does
1711	not satisfy that filter, then remove u from ROS. If ROS is empty, then remove x from
1712	RO. If RO is empty then continue to the next numbered rule. If a PostalAddressFilter
1713	element is specified within the UserBranch, and if the postal address of u does not satisfy
1714	that filter, then remove u from ROS. If ROS is empty, then remove x from RO. If RO is
1715	empty then continue to the next numbered rule. If TelephoneNumberFilter(s) are
1716	specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied
1717	by some of the telephone numbers of u then remove u from ROS. If ROS is empty, then
1718	remove x from RO. If RO is empty then continue to the next numbered rule. If an
1719	OrganizationQuery element is specified within the UserBranch, then let o be the
1720	Organization instance that is identified by the organization that u is affiliated with. If o
1721	doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove u from ROS.
1722	If ROS is empty, then remove x from RO. If RO is empty then continue to the next
1723	numbered rule.
1724	

1725 If a ClassificationOuery is specified within the TargetAssociationBranch, then let ROS be the set of Classification instances that satisfy the ClassificationQuery and are the source 1726 object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty 1727 1728 then continue to the next numbered rule (Rule 2). 1729 1730 If a ServiceBindingBranch is specified within the TargetAssociationBranch, then let ROS 1731 be the set of ServiceBinding instances that are the source object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the next 1732 1733 numbered rule. Let sb be the member of ROS. If a ServiceBindingFilter element is 1734 specified within the ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then 1735 continue to the next numbered rule. If a SpecificationLinkBranch is specified within the 1736 1737 ServiceBindingBranch then consider each SpecificationLinkBranch element separately as 1738 follows: 1739 Let sb be a remaining service binding in ROS. Let SL be the set of all specification link 1740 instances sl that describe specification links of sb. If a SpecificationLinkFilter element is 1741 specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then 1742 remove sl from SL. If SL is empty then remove sb from ROS. If ROS is empty then 1743 remove x from RO. If RO is empty then continue to the next numbered rule. If a 1744 RegistryObjectQuery element is specified within the SpecificationLinkBranch then let sl 1745 be a remaining specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is 1746

not a specification link for some registry object in RO, then remove sl from SL. If SL is

specified within the SpecificationLinkBranch then let sl be a remaining specification link

RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some

registry entry in RE, then remove sl from SL. If SL is empty then remove sb from ROS.

empty then remove sb from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery element is

in SL. Treat RegistryEntryQuery element as follows: Let RE be the result set of the

If ROS is empty then remove x from RO. If RO is empty then continue to the next

1755 1756

1747 1748

1749

1750

1751

1752

1753 1754

numbered rule.

1757 If a SpecificationLinkBranch is specified within the TargetAssociationBranch, then let ROS be the set of SpecificationLink instances that are the source object of some element 1758 of AF. If ROS is empty, then remove x from RO. If RO is empty then continue to the 1759 1760 next numbered rule. Let sl be the member of ROS. If a SpecificationLinkFilter element is specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then 1761 remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then 1762 1763 continue to the next numbered rule. If a RegistryObjectQuery element is specified within 1764 the SpecificationLinkBranch then let sl be a remaining specification link in ROS. Treat RegistryObjectOuery element as follows: Let RO be the result set of the 1765 1766 RegistryObjectQuery as defined in Section 8.2.2. If sl is not a specification link for some registry object in RO, then remove sl from ROS. If ROS is empty then remove x from 1767 1768 RO. If RO is empty then continue to the next numbered rule. If a RegistryEntryQuery 1769 element is specified within the SpecificationLinkBranch then let sl be a remaining 1770 specification link in ROS. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification 1771 1772 link for some registry entry in RE, then remove sl from ROS. If ROS is empty then remove x from RO. If RO is empty then continue to the next numbered rule. If a 1773 1774 ServiceBindingTargetBranch is specified within the ServiceBindingBranch, then let SBT 1775 be the set of ServiceBinding instances that satisfy the ServiceBindingTargetBranch and 1776 are the target service binding of some element of ROT. If SBT is empty then remove sb from ROT. If ROT is empty, then remove x from RO. If RO is empty then continue to the 1777 1778 next numbered rule.

1779

1780If an AssociationQuery is specified within the TargetAssociationBranch, then let ROS be1781the set of Association instances that satisfy the AssociationQuery and are the source1782object of some element of AF. If ROS is empty, then remove x from RO. If RO is empty1783then continue to the next numbered rule (Rule 2).

- If RO is empty, then raise the warning: *registry object query result is empty*; otherwise, set
   RO to be the result of the RegistryObjectQuery.
- 1786
  1787
  3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

### 1788 Examples

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

1794 "Asia/Japan".

```
1795
1796 <AdhocQueryRequest>
1796 <AdhocQueryRequest>
1797 <ResponseOption returnType = "RegistryEntry"/>
1798 <FilterQuery>
1799 <RegistryObjectQuery>
1800 <ClassifiedByBranch>
1801 <ClassificationFilter>
1802 <Clause>
```

1803	<simpleclause leftargument="path"></simpleclause>
1804	<pre><stringclause stringpredicate="Equal">//Automotive</stringclause></pre>
1805	
1806	
1807	
1808	<classificationschemequery></classificationschemequery>
1809	<namebranch></namebranch>
1810	<localizedstringfilter></localizedstringfilter>
1811	<clause></clause>
1812	<simpleclause leftargument="value"></simpleclause>
1813	<stringclause stringpredicate="Equal">urn:ebxml:cs:industry</stringclause>
1814	
1815	
1816	
1817	
1818	
1819	
1820	<classifiedbybranch></classifiedbybranch>
1821	<classificationfilter></classificationfilter>
1822	<clause></clause>
1823	<simpleclause leftargument="path"></simpleclause>
1824	<stringclause stringpredicate="StartsWith">/Geography-id/Asia/Japan</stringclause>
1825	
1826	
1827	
1828	<classificationschemequery></classificationschemequery>
1829	<namebranch></namebranch>
1830	<localizedstringfilter></localizedstringfilter>
1831	<clause></clause>
1832	<simpleclause leftargument="value"></simpleclause>
1833	<stringclause stringpredicate="Equal">urn:ebxml:cs:geography</stringclause>
1834	
1835	
1836	
1837	
1838	
1839	
1840	
1841	
1842	
1843	

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

1850	
1851	<adhocqueryrequest></adhocqueryrequest>
1852	<responseoption returntype="RegistryObject"></responseoption>
1853	<filterquery></filterquery>
1854	<registryobjectquery></registryobjectquery>
1855	<classifiedbybranch></classifiedbybranch>
1856	<classificationnodequery></classificationnodequery>
1857	<descriptionbranch></descriptionbranch>
1858	<localizedstringfilter></localizedstringfilter>
1859	<clause></clause>
1860	<simpleclause leftargument="value"></simpleclause>
1861	<stringclause stringpredicate="Equal">transistor</stringclause>
1862	
1863	
1864	
1865	
1866	
1867	
1868	
1869	
1870	
1871	

## 1872 8.2.3 RegistryEntryQuery

### 1873 Purpose

1874 To identify a set of registry entry instances as the result of a query over selected registry

- 1875 metadata.
- 1876



### 1877 ebRIM Binding

1878

### Figure 17: ebRIM Binding for RegistryEntryQuery

1880	
1881	<complextype name="RegistryEntryQueryType"></complextype>
1882	<complexcontent></complexcontent>
1883	<extension base="tns:RegistryObjectQueryType"></extension>
1884	<sequence></sequence>
1885	<pre><element maxoccurs="1" minoccurs="0" ref="tns:RegistryEntryFilter"></element></pre>
1886	

1887	
1888 1889	 
1890	<pre>   </pre>
1890	<pre><comment name="RegistryEntryQuery" type="ths.RegistryEntryQueryType"></comment></pre>
1892	<element name="RegistryEntryQueryResult"></element>
1893	<complextype></complextype>
1894	<choice maxoccurs="unbounded" minoccurs="0"></choice>
1895	<pre><element ref="rim:ObjectRef"></element></pre>
1896	<element ref="rim:ClassificationScheme"></element>
1897	<element ref="rim:ExtrinsicObject"></element>
1898	<element ref="rim:RegistryEntry"></element>
1899	<element ref="rim:RegistryObject"></element>
1900	<pre><element ref="rim:RegistryPackage"></element></pre>
1901	
1902	
1903 1904	
1904	
1905	Semantic Rules
1906	1. Let RE denote the set of all persistent RegistryEntry instances in the Registry. The following
1907	steps will eliminate instances in RE that do not satisfy the conditions of the specified filters.
1908	a) If RE is empty then continue to the next numbered rule.
1909	b) If a RegistryEntryFilter is not specified then go to the next step; otherwise, let x be a
1910	registry entry in RE. If x does not satisfy the RegistryEntryFilter, then remove x from RE.
1911	If RE is empty then continue to the next numbered rule.
1912	c) Let RE be the set of remaining RegistryEntry instances. Evaluate inherited
1913	RegistryObjectQuery over RE as explained in Section 8.2.2.
1914	2. If RE is empty, then raise the warning: <i>registry entry query result is empty</i> ; otherwise, set RE
1915	to be the result of the RegistryEntryQuery.
1916	3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)
1917	within the RegistryResponse.
1010	
1918	Examples
1919	A client wishes to establish a trading relationship with XYZ Corporation and wants to know if
1920	they have registered any of their business documents in the Registry. The following query
1921	returns a set of registry entry identifiers for currently registered items submitted by any
1922	organization whose name includes the string "XYZ". It does not return any registry entry
1923	identifiers for superseded, replaced, deprecated, or withdrawn items.
1924	
1925	<adhocqueryrequest></adhocqueryrequest>
1926 1927	<responseoption returntype="ObjectRef"></responseoption> <filterquery></filterquery>
1927	<registryentryquery></registryentryquery>
1929	<targetassociationbranch></targetassociationbranch>
1930	<associationfilter></associationfilter>
1931	<clause></clause>
1932	<simpleclause leftargument="associationType"></simpleclause>
1933	<pre><stringclause stringpredicate="Equal">SubmitterOf</stringclause></pre>

1934	
1935	
1936	
1937	<organizationquery></organizationquery>
1938	<namebranch></namebranch>
1939	<localizedstringfilter></localizedstringfilter>
1940	<clause></clause>
1941	<simpleclause leftargument="value"></simpleclause>
1942	<stringclause stringpredicate="Contains">XYZ</stringclause>
1943	SimpleClause>
1944	
1944	
1945	
1947	
1948	
1949	<registryentryfilter></registryentryfilter>
1950	<clause></clause>
1951	<simpleclause leftargument="status"></simpleclause>
1952	<stringclause stringpredicate="Equal">Approved</stringclause>
1953	
1954	
1955	
1956	
1957	
1958	
1959	
1960	A client is using the United Nations Standard Product and Services Classification (UNSPSC)
1061	scheme and wants to identify all companies that deal with products classified as "Integrated
1961	scheme and wants to identify all companies that deal with products classified as "Integrated
1962	circuit components", i.e. UNSPSC code "321118". The client knows that companies have
	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
1962	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each
1962 1963 1964	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with.
1962 1963 1964 1965	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to
1962 1963 1964 1965 1966	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of
1962 1963 1964 1965 1966 1967	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to
1962 1963 1964 1965 1966 1967 1968	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.
1962 1963 1964 1965 1966 1967 1968 1969	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. <adhocqueryrequest></adhocqueryrequest>
1962 1963 1964 1965 1966 1967 1968 1969 1970	circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. <adhocqueryrequest> <responseoption returntype="RegistryEntry"></responseoption></adhocqueryrequest>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components.</pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre> <a href="#database"></a> <a< th=""></a<>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre> <b>ClassificationFilter ClassificationFilter ClassificationSchemeQuery ClassificationSchemeQuery ClassificationSchemeQuery ClassificationSchemeQuery ClassificationSchemeQuery ClassificationSchemeQuery ClassificationSchemeQuery Clause Clause</b>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1977 1978 1979 1980 1981 1982 1983 1984 1985	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>
1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1977 1978 1979 1980 1981 1982 1983 1984 1985	<pre>circuit components", i.e. UNSPSC code "321118". The client knows that companies have registered their Collaboration Protocol Profile (CPP) documents in the Registry, and that each such profile has been classified by UNSPSC according to the products the company deals with. However, the client does not know if the UNSPSC classification scheme is internal or external to this registry. The following query returns a set of approved registry entry instances for CPP's of companies that deal with integrated circuit components. </pre>

1989	
1990	
1991	
1992	
1993	<registryentryfilter></registryentryfilter>
1994	<clause></clause>
1995	<compoundclause connectivepredicate="And"></compoundclause>
1996	<clause></clause>
1997	<simpleclause leftargument="objectType"></simpleclause>
1998	<stringclause stringpredicate="Equal">CPP</stringclause>
1999	
2000	
2001	<clause></clause>
2002	<simpleclause leftargument="status"></simpleclause>
2003	<stringclause stringpredicate="Equal">Approved</stringclause>
2004	
2005	
2006	
2007	
2008	
2009	
2010	
2011	
2012	

#### 2013 8.2.4 AssociationQuery

- 2014 Purpose
- 2015 To identify a set of association instances as the result of a query over selected registry metadata.
- 2016 2017 ebRIM Binding



2018

Figure 18: ebRIM Binding for AssociationQuery

2019	Definition
2020	
2021	<complextype name="AssociationQueryType"></complextype>
2022	<complexcontent></complexcontent>
2023	<extension base="tns:RegistryObjectQueryType"></extension>
2024	<sequence></sequence>
2025	<pre><element maxoccurs="1" minoccurs="0" ref="tns:AssociationFilter"></element></pre>
2026	
2027	
2028	
2029	
2030	<pre><element name="AssociationQuery" type="tns:AssociationQueryType"></element></pre>
2031	
2032	<pre><element name="AssociationQueryResult"></element></pre>
2033	<complextype></complextype>

2034 2035 2036 2037 2038 2039 2040 2041	<choice maxoccurs="unbounded" minoccurs="0"> <element ref="rim:ObjectRef"></element> <element ref="rim:RegistryObject"></element> <element ref="rim:Association"></element> </choice>  
2042	Semantic Rules
2043 2044	1. Let A denote the set of all persistent Association instances in the Registry. The following steps will eliminate instances in A that do not satisfy the conditions of the specified filters.
2045	a) If A is empty then continue to the next numbered rule.
2046 2047 2048 2049	<ul><li>b) If an AssociationFilter element is not directly contained in the AssociationQuery element, then go to the next step; otherwise let x be an association instance in A. If x does not satisfy the AssociationFilter then remove x from A. If A is empty then continue to the next numbered rule.</li></ul>
2050 2051	c) Let A be the set of remaining Association instances. Evaluate inherited RegistryObjectQuery over A as explained in Section 8.2.2.
2052 2053	2. If A is empty, then raise the warning: <i>association query result is empty</i> ; otherwise, set A to be the result of the AssociationQuery.
2054 2055	<b>3</b> . Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.
2056	Examples
2057 2058	A client application wishes to identify a set of associations that are 'equivalentTo' a set of other associations.
2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081	<adhocqueryrequest"> <responseoption returntype="LeafClass"></responseoption> <filterquery> <associationbranch> <associationfilter> <clause> <simpleclause leftargument="associationType"> <stringclause leftargument="associationType"> <stringclause stringpredicate="Equal">EquivalentTo</stringclause> </stringclause></simpleclause></clause>   <simpleclause leftargument="associationType"> <stringclause stringpredicate="Equal">EquivalentTo</stringclause>    </simpleclause> <stringclause stringpredicate="StartsWith">Sin</stringclause>                </associationfilter></associationbranch></filterquery></adhocqueryrequest">
2081 2082	 <associationfilter></associationfilter>

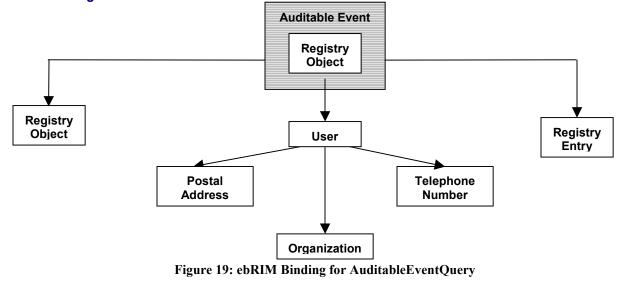
2083	<clause></clause>
2084	SimpleClause leftArgument="associationType">
2085	<pre><stringclause stringpredicate="StartsWith">Son</stringclause></pre>
2086	
2087	
2088	
2089	
2090	
2091	
2092	

## 2093 8.2.5 AuditableEventQuery

### 2094 Purpose

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

## 2097 ebRIM Binding



### 2098

2117 2118 2119 2120 2121 2122 2123 2124		choice minOccurs="0" maxOccurs="unbounded"> <element ref="rim:ObjectRef"></element> <element ref="rim:RegistryObject"></element> <element ref="rim:AuditableEvent"></element> /choice> omplexType> nent>	
2125	Se	ntic Rules	
2126 2127 2128	1.	et AE denote the set of all persistent AuditableEvent instances in the Registry. The ollowing steps will eliminate instances in AE that do not satisfy the conditions of the pecified filters.	
2129		If AE is empty then continue to the next numbered rule.	
2130 2131 2132		) If an AuditableEventFilter is not specified then go to the next step; otherwise, let x be an auditable event in AE. If x does not satisfy the AuditableEventFilter, then remove x from AE. If AE is empty then continue to the next numbered rule.	
2133 2134 2135 2136 2137		If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not an auditable event for some registry object in RO, then remove x from AE. If AE is empty then continue to the next numbered rule.	
2138 2139 2140 2141 2142		) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not an auditable event for some registry entry in RE, then remove x from AE. If AE is empty then continue to the next numbered rule.	
2143 2144 2145 2146 2147 2148 2149 2150 2151 2152 2153 2154 2155		If a UserBranch element is not specified then go to the next step; otherwise, let x be a remaining auditable event in AE. Let u be the user instance that invokes x. If a UserFilte element is specified within the UserBranch, and if u does not satisfy that filter, then remove x from AE. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove x from AE. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of u then remove x from AE. If EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilters isn't satisfied by some of the email addresses of u then remove x from AE. If an OrganizationQuery element is specified within the UserBranch then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x from AE. If AE is empty then continue to the next numbered rule.	
2156 2157		Let AE be the set of remaining AuditableEvent instances. Evaluate inherited RegistryObjectQuery over AE as explained in Section 8.2.2.	
2158 2159	2.	AE is empty, then raise the warning: <i>auditable event query result is empty</i> ; otherwise set E to be the result of the AuditableEventQuery.	
2160	3.	eturn the result and any accumulated warnings or exceptions (in the RegistryErrorList)	

2161 within the RegistryResponse.

	Examples			
2163 2164	A Registry client has registered an item and it has been assigned a name "urn:path:myitem". The client is now interested in all events since the beginning of the year that have impacted that item.			
2165	The following query will return a set of AuditableEvent instances for all such events.			
2166	The following query will retain a set of Auditable Lvent instances for an such events.			
2167	<a dhaaquarydaquaat=""></a>			
2168	<adhocqueryrequest></adhocqueryrequest>			
	<responseoption returntype="LeafClass"></responseoption>			
2169	<filterquery></filterquery>			
2170	<auditableeventquery></auditableeventquery>			
2171	<auditableeventfilter></auditableeventfilter>			
2172	<clause></clause>			
2173	<simpleclause leftargument="timestamp"></simpleclause>			
2174	<rationalclause logicalpredicate="GE"></rationalclause>			
2175	DateTimeClause>2000-01-01T00:00:00-05:00			
2176				
2177				
2178				
2179				
2180	<registryentryquery></registryentryquery>			
2181	<namebranch></namebranch>			
2181				
2182	<localizedstringfilter></localizedstringfilter>			
2183	< <u>Clause&gt;</u>			
2184	<simpleclause leftargument="value"></simpleclause>			
	<stringclause stringpredicate="Equal">urn:path:myitem</stringclause>			
2186				
2187				
2188				
2189				
2190				
2191				
2192				
2193				
2194				
2195	A alight company has many registered objects in the Desigtry. The Desigtry allows events			
	A client company has many registered objects in the Registry. The Registry allows events			
	submitted by other organizations to have an impact on your registered items, e.g. new			
2196				
2196 2197	classifications and new associations. The following query will return a set of identifiers for all			
2197	classifications and new associations. The following query will return a set of identifiers for all			
2197 2198	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by			
2197 2198 2199	classifications and new associations. The following query will return a set of identifiers for all			
2197 2198 2199 2200	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". <adhocqueryrequest></adhocqueryrequest>			
2197 2198 2199 2200 2201 2202	<pre>classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". </pre> <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption></adhocqueryrequest>			
2197 2198 2199 2200 2201 2202 2203	<pre>classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". <adhocqueryrequest></adhocqueryrequest></pre>			
2197 2198 2199 2200 2201 2202 2203 2203 2204	<pre>classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". </pre> <pre> </pre>			
2197 2198 2199 2200 2201 2202 2203 2204 2205	<pre>classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". </pre> <pre> </pre> <pre>    <pre>   <pre>    <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>  <pre>   <pre>   <pre>   <pre>  <pre>   <pre>  <pre>   <pre>   <pre>  <pre>   <pre>  <pre>   <pre>   <pre>  <pre>   <pre>   <pre>  <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>  <pre>   <pre>   <pre>   <pre>    <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>  <pre>   <pre>   <pre>   <pre>  <pre>   <pre>   <pre>   <pre>   <pre>  <pre>   <pre>  <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>    <pre>   <pre>    <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>   <pre>    <pre>    <pre>    <pre>   <pre>    <pre>   <pre>     <pre>     </pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206	<pre>classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". </pre> <pre> </pre>			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <auditableeventquery> <registryentryquery> <targetassociationbranch> <associationfilter> <clause> <simpleclause leftargument="associationType"> <stringclause> </stringclause></simpleclause></clause></associationfilter></targetassociationbranch></registryentryquery></auditableeventquery></filterquery></adhocqueryrequest>			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <auditableeventquery> <registryentryquery> <targetassociationbranch> <associationfilter> <clause> <stimpleclause leftargument="associationType"> <stimpleclause "equal"="" =="">SubmitterOf </stimpleclause></stimpleclause></clause> </associationfilter></targetassociationbranch></registryentryquery></auditableeventquery></filterquery></adhocqueryrequest>			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg". <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <auditableeventquery> <registryentryquery> <targetassociationbranch> <associationfilter> <clause> <stringclause leftargument="associationType"> <stringclause "equal"="" =="">SubmitterOf</stringclause> </stringclause></clause>    </associationfilter></targetassociationbranch></registryentryquery></auditableeventquery></filterquery></adhocqueryrequest>			
2197 2198 2199 2200 2201 2202 2203 2204 2205 2206 2207 2208 2209 2210 2211 2212 2213 2214	classifications and new associations. The following query will return a set of identifiers for all auditable events, invoked by some other party, that had an impact on an item submitted by "myorg".			

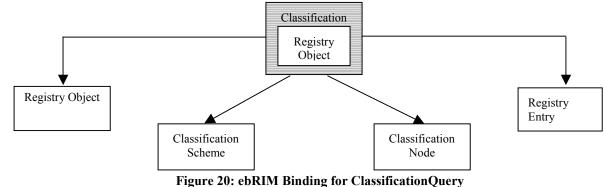
2217		
2217	<clause></clause>	
2218	<simpleclause leftargument="value"></simpleclause>	
2219	<stringclause stringpredicate="Equal">myorg</stringclause>	
2220		
2221		
2222		
2223		
2224		
2225		
2226		
2227	<userbranch></userbranch>	
2228	<organizationquery></organizationquery>	
2229	<namebranch></namebranch>	
2230	<localizedstringfilter></localizedstringfilter>	
2231	<clause></clause>	
2232	<simpleclause leftargument="value"></simpleclause>	
2233	<stringclause stringpredicate="-Equal">myorg</stringclause>	
2234		
2235		
2236		
2237		
2238		
2239		
2240		
2241		
2242		
2243		

#### 8.2.6 ClassificationQuery 2244

#### 2245 **Purpose**

To identify a set of classification instances as the result of a query over selected registry 2246 2247 metadata.

#### 2248 ebRIM Binding



## 2249

2250	Definition
2251	
2252	<complextype name="ClassificationQueryType"></complextype>
2253	<complexcontent></complexcontent>
2254	<extension base="tns:RegistryObjectQueryType"></extension>
2255	<sequence></sequence>
2256	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ClassificationFilter"></element></pre>

2257 2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271 2272 2273 2274 2275 2276	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ClassificationSchemeQuery"></element> <element maxoccurs="1" minoccurs="0" ref="tns:ClassificationNodeQuery"></element> <element maxoccurs="1" minoccurs="0" ref="tns:RegistryObjectQuery"></element> <element maxoccurs="1" minoccurs="0" ref="tns:RegistryEntryQuery"></element>   </pre>
2277	Semantic Rules
2278 2279	1. Let C denote the set of all persistent Classification instances in the Registry. The following steps will eliminate instances in C that do not satisfy the conditions of the specified filters.
2280	a) If C is empty then continue to the next numbered rule.
2281 2282 2283 2284	b) If a ClassificationFilter element is not directly contained in the ClassificationQuery element, then go to the next step; otherwise let x be an classification instance in C. If x does not satisfy the ClassificationFilter then remove x from C. If C is empty then continue to the next numbered rule.
2285 2286 2287 2288	c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from C. If C is empty then continue to the next numbered rule.
2289 2290 2291 2292	d) If a ClassificationNodeQuery is not specified then go to the next step; otherwise, let x be a remaining classification in C. If the classification node of x does not satisfy the ClassificationNodeQuery as defined in Section 8.2.7, then remove x from C. If C is empty then continue to the next numbered rule.
2293 2294 2295 2296 2297	e) If a RegistryObjectQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryObjectQuery element as follows: Let RO be the result set of the RegistryObjectQuery as defined in Section 8.2.2. If x is not a classification of at least one registry object in RO, then remove x from C. If C is empty then continue to the next numbered rule.
2298 2299 2300 2301 2302	f) If a RegistryEntryQuery element is not specified then go to the next step; otherwise, let x be a remaining classification in C. Treat RegistryEntryQuery element as follows: Let RE be the result set of the RegistryEntryQuery as defined in Section 8.2.3. If x is not a classification of at least one registry entry in RE, then remove x from C. If C is empty then continue to the next numbered rule.

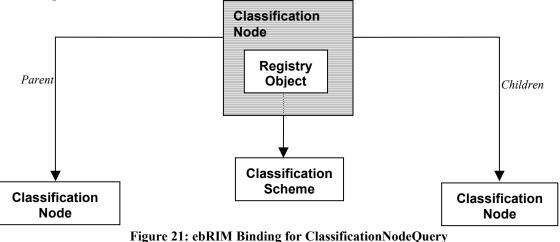
- 23032. If C is empty, then raise the warning: *classification query result is empty*; otherwise2304 otherwise, set C to be the result of the ClassificationQuery.
- 2305 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)2306 within the RegistryResponse.

## 2307 8.2.7 ClassificationNodeQuery

#### 2308 Purpose

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

#### 2311 ebRIM Binding



#### 2312

2212	Definition		
2313	Definition		
2314 2315	<		
2313	<complextype name="ClassificationNodeQueryType"></complextype>		
2310	<complexcontent></complexcontent>		
2317	<extension base="tns:RegistryObjectQueryType"></extension>		
2318	<sequence></sequence>		
	<element maxoccurs="1" minoccurs="0" ref="tns:ClassificationNodeFilter"></element>		
2320	<pre><element maxoccurs="1" minoccurs="0" ref="tns:ClassificationSchemeQuery"></element></pre>		
2321	<pre><element <="" minoccurs="0" name="ClassificationNodeParentBranch" pre="" type="ClassificationNodeQueryType"></element></pre>		
2322	maxOccurs="1" />		
2323	<element <="" name="ClassificationNodeChildrenBranch" p="" type="ClassificationNodeQueryType"></element>		
2324	minOccurs="0" maxOccurs="unbounded" />		
2325			
2326			
2327			
2328			
2329	<element name="ClassificationNodeQuery" type="tns:ClassificationNodeQueryType"></element>		
2330			
2331	<element name="ClassificationNodeQueryResult"></element>		
2332	<complextype></complextype>		
2333	<choice maxoccurs="unbounded" minoccurs="0"></choice>		
2334	<element ref="rim:ObjectRef"></element>		
2335	<element ref="rim:RegistryObject"></element>		
2336	<element ref="rim:ClassificationNode"></element>		
2337			

2338 2339 2340	 		
2341	1 Semantic Rules		
2342 2343 2344	1. Let CN denote the set of all persistent ClassificationNode instances in the Registry. The following steps will eliminate instances in CN that do not satisfy the conditions of the specified filters.		
2345	a) If CN is empty then continue to the next numbered rule.		
2346 2347 2348	b) If a ClassificationNodeFilter is not specified then go to the next step; otherwise, let x be a classification node in CN. If x does not satisfy the ClassificationNodeFilter then remove x from CN. If CN is empty then continue to the next numbered rule.		
2349 2350 2351 2352	c) If a ClassificationSchemeQuery is not specified then go to the next step; otherwise, let x be a remaining classification node in CN. If the defining classification scheme of x does not satisfy the ClassificationSchemeQuery as defined in Section 8.2.8, then remove x from CN. If CN is empty then continue to the next numbered rule.		
2353 2354 2355 2356 2357 2358 2359 2360 2361 2362 2363 2364 2365 2366	<ul> <li>d) If a ClassificationNodeParentBranch element is not specified, then go to the next step; otherwise, let x be a remaining classification node in CN and execute the following paragraph with n=x.</li> <li>Let n be a classification node instance. If n does not have a parent node (i.e. if n is a base level node), then remove x from CN and go to the next step; otherwise, let p be the parent node of n. If a ClassificationNodeFilter element is directly contained in the ClassificationNodeParentBranch and if p does not satisfy the ClassificationNodeFilter, then remove x from CN. If CN is empty then continue to the next numbered rule. If a ClassificationNodeParentBranch and if defining classification scheme of p does not satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue to the next numbered rule. If a nother ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue to the next numbered rule. If another ClassificationNodeParentBranch and if defining classification scheme of p does not satisfy the ClassificationSchemeQuery, then remove x from CN. If CN is empty then continue to the next numbered rule.</li> <li>If another ClassificationNodeParentBranch element is directly contained within this ClassificationNodeParentBranch element, then repeat the previous paragraph with n=p.</li> </ul>		
2367 2368 2369 2370 2371 2372	e) If a ClassificationNodeChildrenBranch element is not specified, then continue to the next numbered rule; otherwise, let x be a remaining classification node in CN. If x is not the parent node of some ClassificationNode instance, then remove x from CN and if CN is empty continue to the next numbered rule; otherwise, treat each ClassificationNodeChildrenBranch element separately and execute the following paragraph with $n = x$ .		

2373	Let n be a classification node instance. If a ClassificationNodeFilter element is not	
2374	specified within the ClassificationNodeChildrenBranch element then let CNC be the set	
2375	of all classification nodes that have n as their parent node; otherwise, let CNC be the set	
2376	of all classification nodes that satisfy the ClassificationNodeFilter and have n as their	
2377	parent node. If CNC is empty, then remove x from CN and if CN is empty continue to the	
2378	next numbered rule; otherwise, let c be any member of CNC. If a	
2379	ClassificationSchemeQuery element is directly contained in the	
2380	ClassificationNodeChildrenBranch and if the defining classification scheme of c does not	
2381	satisfy the ClassificationSchemeQuery then remove c from CNC. If CNC is empty then	
2382	remove x from CN. If CN is empty then continue to the next numbered rule; otherwise,	
2383	let y be an element of CNC and continue with the next paragraph.	
2384	If the ClassificationNodeChildrenBranch element is terminal, i.e. if it does not directly	
2385	contain another ClassificationNodeChildrenBranch element, then continue to the next	
2386	numbered rule; otherwise, repeat the previous paragraph with the new	
2387	ClassificationNodeChildrenBranch element and with $n = y$ .	
2388	f) Let CN be the set of remaining ClassificationNode instances. Evaluate inherited	
2389	RegistryObjectQuery over CN as explained in Section 8.2.2.	
2390	2. If CN is empty, then raise the warning: <i>classification node query result is empty</i> ; otherwise	
2390	set CN to be the result of the ClassificationNodeQuery.	
2392	3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList)	
2393	within the RegistryResponse.	
2394	Path Filter Expression usage in ClassificationNodeFilter	
2395	The path filter expression is used to match classification nodes in ClassificationNodeFilter	
2396	elements involving the path attribute of the ClassificationNode class as defied by the getPath	
2397	method in [ebRIM].	
2398	The path filter expressions are based on a very small and proper sub-set of location path syntax	
2399	of XPath.	
2400		
2400	The path filter expression syntax includes support for matching multiple nodes by using wild	
	card syntax as follows:	
2402	• Use of '*' as a wildcard in place of any path element in the pathFilter	
2403	• Use of '//' syntax to denote any descendent of a node in the pathFilter	
2404 2405 2406 2407 2408 2409 2410 2411	It is defined by the following BNF grammar:	
2406	<pre>pathFilter :== '/' schemeId nodePath</pre>	
2407	nodePath ::= slashes nodeCode   slashes `*'	
2409	slashes nodeCode ( nodePath )?	
2411	Slashes ::= `/'   `//'	
2412	In the above grammer, schemeId is the id attribute of the ClassificationScheme instance. In the	

- above grammar nodeCode is defined by NCName production as defined by
- 2414 <u>http://www.w3.org/TR/REC-xml-names/#NT-NCName</u>.
- 2415 The semantic rules for the ClassificationNodeFilter element allow the use of path attribute as a
- filter that is based on the EQUAL clause. The pattern specified for matching the EQUAL clause is a PATH Filter expression.

- 2418 This is illustrated in the following example that matches all second level nodes in
- 2419 2420 2421 2422 2423 2423 2425 2425 2426 2427 2428 ClassificationScheme with id 'Geography-id' and with code 'Japan':

```
<ClassificationNodeQuery>
          <ClassificationNodeFilter>
            <Clause>
              <SimpleClause leftArgument = "path">
                <StringClause stringPredicate = "Equal">//Geography-id/*/Japan</StringClause>
               </SimpleClause>
            </Clause>
          </ClassificationNodeFilter>
  ΰğ
        </ClassificationNodeQuery>
\bar{2}430
```

#### 2431 **Use Cases and Examples of Path Filter Expressions**

2432 The following table lists various use cases and examples using the sample Geography scheme 2433 2434 2435 2436 2437 2438 2439 below:

```
<ClassificationScheme id='Geography-id' name="Geography"/>
<ClassificationNode id="NorthAmerica-id" parent="Geography-id" code=NorthAmerica" />
<ClassificationNode id="UnitedStates-id" parent="NorthAmerica-id" code="UnitedStates" />
<ClassificationNode id="Asia-id" parent="Geography-id" code="Asia" />
<ClassificationNode id="Japan-id" parent="Asia-id" code="Japan" />
<ClassificationNode id="Tokyo-id" parent="Japan-id" code="Tokyo" />
```

2444

2440

**Table 10: Path Filter Expressions for Use Cases** 

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

#### 2445 **Examples**

2446 A client application wishes to identify all of the classification nodes in the first three levels of a 2447 classification scheme hierarchy. The client knows that the name of the underlying classification

2449	scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three levels.	
2450		
2451	<adhocqueryrequest></adhocqueryrequest>	
2452	<responseoption returntype="LeafClass"></responseoption>	
2453	<filterquery></filterquery>	
2454	<classificationnodequery></classificationnodequery>	
2455	<classificationnodefilter></classificationnodefilter>	
2456	<clause></clause>	
2457	<simpleclause leftargument="levelNumber"></simpleclause>	
2458	<rationalclause logicalpredicate="LE"></rationalclause>	
2459	<intclause>3</intclause>	
2460		
2461		
2462		
2463		
2464	<classificationschemequery></classificationschemequery>	
2465	<namebranch></namebranch>	
2466	<localizedstringfilter></localizedstringfilter>	
2467	<clause></clause>	
2468	<simpleclause leftargument="value"></simpleclause>	
2469	<stringclause stringpredicate="Equal">urn:ebxml:cs:myscheme</stringclause>	
2470		
2471		
2472		
2473		
2474		
2475		
2476		
2477		
2478		
2479	If instead the client wishes all levels returned, they could simply delete the	
2479	If, instead, the client wishes all levels returned, they could simply delete the ClassificationNedeFilter element from the guery.	
2479 2480	If, instead, the client wishes all levels returned, they could simply delete the ClassificationNodeFilter element from the query.	
2480	ClassificationNodeFilter element from the query.	
2480 2481		
2480 2481 2482	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica.	
2480 2481 2482 2483	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest></adhocqueryrequest>	
2480 2481 2482 2483 2483 2484	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption></adhocqueryrequest>	
2480 2481 2482 2483 2484 2484 2485	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2483 2484 2485 2486	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2483 2484 2485 2486 2487	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2483 2484 2485 2486 2486 2487 2488	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause></clause></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2483 2484 2485 2486 2487 2488 2488 2489	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"></simpleclause></clause></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2485 2487 2488 2487 2488 2489 2490	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause></simpleclause></clause></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2485 2486 2487 2488 2489 2490 2491	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </simpleclause></clause></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <classificationnodefilter> <simpleclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </simpleclause></classificationnodefilter></classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </stringclause></simpleclause></clause>  </classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </simpleclause></clause></classificationnodefilter>    </classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <stringclause leftargument="path"> <stringclause leftargument="path"> <clause> </clause></stringclause></stringclause></clause></classificationnodefilter>      </classificationnodequery> </filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </simpleclause></clause></classificationnodefilter>    </classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause leftargument="path"> <clause> </clause> </stringclause></simpleclause></clause>  </classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2495 2496 2497 2498	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <stringclause leftargument="path"> <stringclause leftargument="path"> <clause> </clause></stringclause></stringclause></clause></classificationnodefilter>      </classificationnodequery> </filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause leftargument="path"> <clause> </clause> </stringclause></simpleclause></clause>  </classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2495 2496 2497 2498	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <simpleclause leftargument="path"> <stringclause leftargument="path"> <clause> </clause> </stringclause></simpleclause></clause>  </classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <stringclause leftargument="path"> <stringclause leftargument="path"> <stringclause "equal"="" =="">/Geography-id/NorthAmerica/*</stringclause> </stringclause></stringclause></clause> </classificationnodefilter>    </classificationnodequery> </filterquery> </adhocqueryrequest> The following query finds all third level nodes with code of Tokyo.	
2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2494 2495 2496 2497 2498 2499 2500	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <adhocqueryrequest> <responseoption returntype="LeafClass"></responseoption> <filterquery> <classificationnodequery> <classificationnodefilter> <clause> <stringclause stringpredicate="Equal">/Geography-id/NorthAmerica/*</stringclause> </clause>   </classificationnodefilter></classificationnodequery></filterquery></adhocqueryrequest>	
2480 2481 2482 2483 2484 2485 2486 2487 2488 2489 2490 2491 2492 2493 2494 2495 2496 2497 2498 2499 2500 2501	ClassificationNodeFilter element from the query. The following query finds all children nodes of a first level node whose code is NorthAmerica. <pre></pre> <pre></pre> <pre></pre> <pre></pre> <pre></pre> <pre>ClassificationNodeQuery&gt; <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>	

scheme is "urn:ebxml:cs:myscheme". The following query identifies all nodes at the first three 2448

2503	<classificationnodequery></classificationnodequery>
2504	<classificationnodefilter></classificationnodefilter>
2505	<clause></clause>
2506	<simpleclause leftargument="path"></simpleclause>
2507	<pre><stringclause stringpredicate="Equal">/Geography-id/*/*/Tokyo</stringclause></pre>
2508	
2509	
2510	
2511	
2512	
2513	
2514	

## 2515 8.2.8 ClassificationSchemeQuery

### 2516 Purpose

- To identify a set of classification scheme instances as the result of a query over selected registry metadata.
- 2518 metadata.

### 2519 ebRIM Binding



2520

Figure 22: ebRIM Binding for ClassificationSchemeQuery

#### 2521 Definition 2522 2523 <complexType name="ClassificationSchemeQueryType"> 2524 <complexContent> 2525 <extension base="tns:RegistryEntryQueryType"> 2526 <sequence> 2527 <element ref="tns:ClassificationSchemeFilter" minOccurs="0" maxOccurs="1" /> 2528 </sequence> 2529 </extension> 2530 </complexContent>

- 2531 </complexType>
- 2532 <element name="ClassificationSchemeQuery" type="tns:ClassificationSchemeQueryType" />
  2533

## 2534 Semantic Rules

- Let CS denote the set of all persistent ClassificationScheme instances in the Registry. The
   following steps will eliminate instances in CS that do not satisfy the conditions of the
   specified filters.
- a) If CS is empty then continue to the next numbered rule.
- b) If a ClassificationSchemeFilter is not specified then go to the next step; otherwise, let x
  be a classification scheme in CS. If x does not satisfy the ClassificationSchemeFilter,
  then remove x from CS. If CS is empty then continue to the next numbered rule.

2542 2543	<ul> <li>c) Let CS be the set of remaining ClassificationScheme instances. Evaluate inherited RegistryEntryQuery over CS as explained in Section 8.2.3.</li> </ul>
2544 2545	2. If CS is empty, then raise the warning: <i>classification scheme query result is empty</i> ; otherwise, set CS to be the result of the ClassificationSchemeQuery.
2546 2547	3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.
2548	Examples
2549	A client application wishes to identify all classification scheme instances in the Registry.
2550	<adhocqueryrequest></adhocqueryrequest>
2551	<responseoption returntype="LeafClass"></responseoption>
2552	<filterquery></filterquery>
2553	<classificationschemequery></classificationschemequery>
2554	
2555	

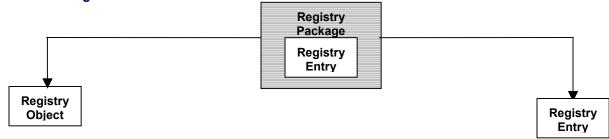
2556

## 2557 8.2.9 RegistryPackageQuery

## 2558 Purpose

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

## 2561 ebRIM Binding



### 2562

Figure 23: ebRIM Binding for RegistryPackageQuery

## 2563 **Definition**

2564	
2565	<complextype name="RegistryPackageQueryType"></complextype>
2566	<complexcontent></complexcontent>
2567	<extension base="tns:RegistryEntryQueryType"></extension>
2568	<sequence></sequence>
2569	<pre><element maxoccurs="1" minoccurs="0" ref="tns:RegistryPackageFilter"></element></pre>
2570	<pre><element maxoccurs="unbounded" minoccurs="0" ref="tns:RegistryObjectQuery"></element></pre>
2571	<element maxoccurs="unbounded" minoccurs="0" ref="tns:RegistryEntryQuery"></element>
2572	
2573	
2574	
2575	
2576	<element name="RegistryPackageQuery" type="tns:RegistryPackageQueryType"></element>
2577	
2578	<element name="RegistryPackageQueryResult"></element>

2579 2580 2581 2582 2583 2584 2585 2586 2586 2587 2588	<complextype> <choice maxoccurs="unbounded" minoccurs="0"> <element ref="rim:ObjectRef"></element> <element ref="rim:RegistryEntry"></element> <element ref="rim:RegistryObject"></element> <element ref="rim:RegistryPackage"></element> </choice> </complextype> 		
2589	Semantic Rules		
2590 2591 2592	<ol> <li>Let RP denote the set of all persistent RegistryPackage instances in the Registry. The following steps will eliminate instances in RP that do not satisfy the conditions of the specified filters.</li> </ol>		
2593	a) If RP is empty then continue to the next numbered rule.		
2594 2595 2596 2597	b) If a RegistryPackageFilter is not specified, then continue to the next numbered rule; otherwise, let x be a registry package instance in RP. If x does not satisfy the RegistryPackageFilter then remove x from RP. If RP is empty then continue to the next numbered rule.		
2598 2599 2600 2601 2602 2603 2604 2605 2606 2607	c) If a RegistryObjectQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryObjectQuery as follows: let RO be the set of RegistryObject instances returned by the RegistryObjectQuery as defined in Section 8.2.2 and let PO be the subset of RO that are members of the package x. If PO is empty, then remove x from RP. If RP is empty then continue to the next numbered rule. If a RegistryEntryQuery element is directly contained in the RegistryPackageQuery element then treat each RegistryEntryQuery as follows: let RE be the set of RegistryEntry instances returned by the RegistryEntryQuery as defined in Section 8.2.3 and let PE be the subset of RE that are members of the package x. If PE is empty, then remove x from RP. If RP is empty then continue to the next numbered rule.		
2608 2609	d) Let RP be the set of remaining RegistryPackage instances. Evaluate inherited RegistryEntryQuery over RP as explained in Section 8.2.3.		
2610 2611	2. If RP is empty, then raise the warning: <i>registry package query result is empty</i> ; otherwise set RP to be the result of the RegistryPackageQuery.		
2612 2613	<ol> <li>Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.</li> </ol>		
2614	Examples		
2615 2616 2617 2618 2619 2620 2621 2622 2623 2623 2624	A client application wishes to identify all package instances in the Registry that contain an Invoice extrinsic object as a member of the package. AdhocQueryRequest> ResponseOption returnType = "LeafClass"/>		

2625 2626 2627 2628 2629 2630 2631 2632	<simpleclause leftargument="objectType"></simpleclause>
2633 2634 2625	
2635 2636 2637	A client application wishes to identify all package instances in the Registry that are not empty. <adhocqueryrequest></adhocqueryrequest>
2638 2639	<responseoption returntype="LeafClass"></responseoption> <filterquery></filterquery>
2640 2641 2642	<registrypackagequery> <registryobjectquery></registryobjectquery></registrypackagequery>
2642 2643 2644	  
2645	~/AutocQueryRequest

A client application wishes to identify all package instances in the Registry that are empty. Since the RegistryPackageQuery is not set up to do negations, clients will have to do two separate

- 2648 RegistryPackageQuery requests, one to find all packages and another to find all non-empty
- 2649 packages, and then do the set difference themselves. Alternatively, they could do a more
- 2650 complex RegistryEntryQuery and check that the packaging association between the package and 2651 its members is non-existent.
- 2652 <u>Note</u>: A registry package is an intrinsic RegistryEntry instance that is completely determined by 2653 its associations with its members. Thus a RegistryPackageQuery can always be re-specified as an 2654 equivalent RegistryEntryQuery using appropriate "Source" and "Target" associations. However, 2655 the equivalent RegistryEntryQuery is often more complicated to write
- the equivalent RegistryEntryQuery is often more complicated to write.

## 2656 8.2.10 ExtrinsicObjectQuery

### 2657 Purpose

- 2658 To identify a set of extrinsic object instances as the result of a query over selected registry
- 2659 metadata.

Extrinsic Object		
	Registry Entry	

## 2660 ebRIM Binding

### 2661

### Figure 24: ebRIM Binding for ExtrinsicObjectQuery

# 2662 **Definition** 2663

2001			
2664	<complextype name="ExtrinsicObjectQueryType"></complextype>		
2665	<complexcontent></complexcontent>		
2666	<extension base="tns:RegistryEntryQueryType"></extension>		
2667	<sequence></sequence>		
2668	<element maxoccurs="1" minoccurs="0" ref="tns:ExtrinsicObjectFilter"></element>		
2669			
2670			
2671			
2672			
2673	<element name="ExtrinsicObjectQuery" type="tns:ExtrinsicObjectQueryType"></element>		
2674			
2675	<element name="ExtrinsicObjectQueryResult"></element>		
2676	<complextype></complextype>		
2677	<choice maxoccurs="unbounded" minoccurs="0"></choice>		
2678	<element ref="rim:ObjectRef"></element>		
2679	<element ref="rim:RegistryEntry"></element>		
2680	<element ref="rim:RegistryObject"></element>		
2681	<pre><element ref="rim:ExtrinsicObject"></element></pre>		
2682			
2683			
2684			
2685			
2000			
2686	Semantic Rules		
2687 2688 2689	1. Let EO denote the set of all persistent ExtrinsicObject instances in the Registry. The following steps will eliminate instances in EO that do not satisfy the conditions of the specified filters.		

- a) If EO is empty then continue to the next numbered rule.
- b) If a ExtrinsicObjectFilter is not specified then go to the next step; otherwise, let x be an
  extrinsic object in EO. If x does not satisfy the ExtrinsicObjectFilter then remove x from
  EO. If EO is empty then continue to the next numbered rule.
- 2694 c) Let EO be the set of remaining ExtrinsicObject instances. Evaluate inherited
   2695 RegistryEntryQuery over EO as explained in Section 8.2.3.
- 2696 2. If EO is empty, then raise the warning: *extrinsic object query result is empty*; otherwise, set
  2697 EO to be the result of the ExtrinsicObjectQuery.
- 2698
  2699
  3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) within the RegistryResponse.

## 2700 8.2.11 OrganizationQuery

- 2701 Purpose
- To identify a set of organization instances as the result of a query over selected registrymetadata.
- 2704 ebRIM Binding

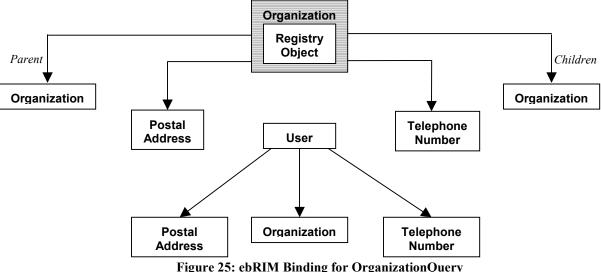


Figure 25: ebRIM Binding for OrganizationQuery	
Definition	
<complextype name="OrganizationQueryType"></complextype>	
<complexcontent></complexcontent>	
<extension base="tns:RegistryObjectQueryType"></extension>	
<sequence></sequence>	
<pre><element maxoccurs="1" minoccurs="0" ref="tns:OrganizationFilter"></element></pre>	
<pre><element maxoccurs="1" minoccurs="0" ref="tns:PostalAddressFilter"></element></pre>	
<pre><element maxoccurs="unbounded" minoccurs="0" ref="tns:TelephoneNumberFilter"></element></pre>	
<element maxoccurs="1" minoccurs="0" ref="tns:UserBranch"></element>	
<pre><element maxoccurs="1" minoccurs="0&lt;/pre&gt;&lt;/td&gt;&lt;/tr&gt;&lt;tr&gt;&lt;td&gt;" name="OrganizationParentBranch" type="tns:OrganizationQueryType"></element></pre>	
<pre><element <="" minoccurs="0" name="OrganizationChildrenBranch" pre="" type="tns:OrganizationQueryType"></element></pre>	
maxOccurs="unbounded" />	
<pre><element name="OrganizationQuery" type="tns:OrganizationQueryType"></element></pre>	
erement mante ergannianten Querj rijpe indiergannianten Querj rijpe i	
<element name="OrganizationQueryResult"></element>	
<complextype></complextype>	
<choice maxoccurs="unbounded" minoccurs="0"></choice>	
<pre><elote and="" himoceurs="" of="" ounded=""> </elote></pre>	
<pre><element ref="rim:RegistryObject"></element></pre>	
<pre><element ref="rim:Organization"></element></pre>	

#### 2736 **Semantic Rules**

- 1. Let ORG denote the set of all persistent Organization instances in the Registry. The 2737 following steps will eliminate instances in ORG that do not satisfy the conditions of the 2738 specified filters. 2739
- a) If ORG is empty then continue to the next numbered rule. 2740

2741 H 2742 2743 2744	b)	If an OrganizationFilter element is not directly contained in the OrganizationQuery element, then go to the next step; otherwise let x be an organization instance in ORG. If x does not satisfy the OrganizationFilter then remove x from ORG. If ORG is empty then continue to the next numbered rule.
2745 o 2746 2747 2748	c)	If a PostalAddressFilter element is not directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. If postal address of x does not satisfy the PostalAddressFilter then remove x from ORG. If ORG is empty then continue to the next numbered rule.
2749 o 2750 2751 2752		If no TelephoneNumberFilter element is directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. If any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then remove x from ORG. If ORG is empty then continue to the next numbered rule.
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766	e)	If a UserBranch element is not directly contained in the OrganizationQuery element then go to the next step; otherwise, let x be an extrinsic object in ORG. Let u be the user instance that is affiliated with x. If a UserFilter element is specified within the UserBranch, and if u does not satisfy that filter, then remove x from ORG. If a PostalAddressFilter element is specified within the UserBranch, and if the postal address of u does not satisfy that filter, then remove x from ORG. If TelephoneNumberFilter(s) are specified within the UserBranch and if any of the TelephoneNumberFilters isn't satisfied by some of the telephone numbers of x then remove x from ORG. If EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilter(s) are specified within the UserBranch and if any of the EmailAddressFilter(s) are specified by some of the email addresses of x then remove x from ORG. If an OrganizationQuery element is specified within the UserBranch, then let o be the Organization instance that is identified by the organization that u is affiliated with. If o doesn't satisfy OrganizationQuery as defined in Section 8.2.11 then remove x from ORG. If ORG is empty then continue to the next numbered rule.
2767 f 2768 2769 2770 2771 2772 2773 2774 2775 2776		If a OrganizationParentBranch element is not specified within the OrganizationQuery, then go to the next step; otherwise, let x be an extrinsic object in ORG. Execute the following paragraph with $o = x$ : Let o be an organization instance. If an OrganizationFilter is not specified within the OrganizationParentBranch and if o has no parent (i.e. if o is a root organization in the Organization hierarchy), then remove x from ORG; otherwise, let p be the parent organization of o. If p does not satisfy the OrganizationFilter, then remove x from ORG. If ORG is empty then continue to the next numbered rule. If another OrganizationParentBranch element is directly contained within this OrganizationParentBranch element, then repeat the previous paragraph with $o = p$ .
2777 g 2778 2779 2780 2781		If a OrganizationChildrenBranch element is not specified, then continue to the next numbered rule; otherwise, let x be a remaining organization in ORG. If x is not the parent node of some organization instance, then remove x from ORG and if ORG is empty continue to the next numbered rule; otherwise, treat each OrganizationChildrenBranch element separately and execute the following paragraph with $n = x$ .

2782 Let n be an organization instance. If an OrganizationFilter element is not specified within the OrganizationChildrenBranch element then let ORGC be the set of all organizations 2783 that have n as their parent node; otherwise, let ORGC be the set of all organizations that 2784 2785 satisfy the OrganizationFilter and have n as their parent node. If ORGC is empty, then remove x from ORG and if ORG is empty continue to the next numbered rule: otherwise. 2786 2787 let c be any member of ORGC. If a PostalAddressFilter element is directly contained in 2788 the OrganizationChildrenBranch and if the postal address of c does not satisfy the 2789 PostalAddressFilter then remove c from ORGC. If ORGC is empty then remove x from 2790 ORG. If ORG is empty then continue to the next numbered rule. If no 2791 TelephoneNumberFilter element is directly contained in the OrganizationChildrenBranch and if If any of the TelephoneNumberFilters isn't satisfied by some of the telephone 2792 2793 numbers of c then remove c from ORGC. If ORGC is empty then remove x from ORG. If 2794 ORG is empty then continue to the next numbered rule; otherwise, let y be an element of 2795 ORGC and continue with the next paragraph. 2796 If the OrganizationChildrenBranch element is terminal, i.e. if it does not directly contain 2797 another OrganizationChildrenBranch element, then continue to the next numbered rule; 2798 otherwise, repeat the previous paragraph with the new OrganizationChildrenBranch 2799 element and with n = y. 2800 h) Let ORG be the set of remaining Organization instances. Evaluate inherited 2801 RegistryObjectQuery over ORG as explained in Section 8.2.2.

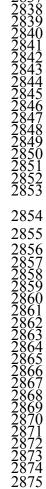
- 2802 2. If ORG is empty, then raise the warning: organization query result is empty; otherwise set 2803 ORG to be the result of the OrganizationQuery.
- 2804 3. Return the result and any accumulated warnings or exceptions (in the RegistryErrorList) 2805 within the RegistryResponse.

#### 2806 **Examples**

2836

2807 A client application wishes to identify a set of organizations, based in France, that have 2808 submitted a PartyProfile extrinsic object this year. 

```
<AdhocQueryRequest>
 <ResponseOption returnType = "LeafClass" returnComposedObjects = "True"/>
 <FilterQuery>
       <OrganizationOuerv>
           <SourceAssociationBranch>
               <AssociationFilter>
                  < Clause>
                      <SimpleClause leftArgument = "associationType">
                          <StringClause stringPredicate = "Equal">SubmitterOf</StringClause>
                      </SimpleClause>
                   </Clause>
               </AssociationFilter>
               <RegistryObjectQuery>
                   <RegistryObjectFilter>
                      <Clause>
                          <SimpleClause leftArgument = "objectType">
                              <StringClause stringPredicate = "Equal">CPP</StringClause>
                          </SimpleClause>
                      </Clause>
                   </RegistryObjectFilter>
                   <AuditableEventQuery>
                      <AuditableEventFilter>
                          <Clause>
                              <SimpleClause leftArgument = "timestamp">
                                <RationalClause logicalPredicate = "GE">
                                  <DateTimeClause>2000-01-01T00:00:00-05:00</DateTimeClause>
                                </RationalClause>
```



<postaladdressfilter></postaladdressfilter>
<clause></clause>
<simpleclause leftargument="country"></simpleclause>
<pre><stringclause stringpredicate="Equal">France</stringclause></pre>

A client application wishes to identify all organizations that have Corporation named XYZ as a

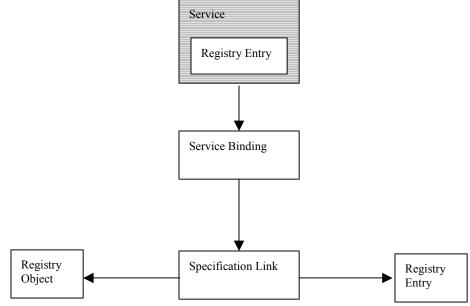
parent.

```
<AdhocQueryRequest>
 <ResponseOption returnType = "LeafClass"/>
 <FilterQuery>
      <OrganizationQuery>
           <OrganizationParentBranch>
              <NameBranch>
                  <LocalizedStringFilter>
                      <Clause>
                          <SimpleClause leftArgument = "value">
                              <StringClause stringPredicate = "Equal">XYZ</StringClause>
                          </SimpleClause>
                      </Clause>
                  </LocalizedStringFilter>
               </NameBranch>
           </OrganizationParentBranch>
       </OrganizationQuery>
 </FilterQuery>
</AdhocQueryRequest>
```

## 2876 8.2.12 ServiceQuery

2877 Purpose

- 2878
- 2879 To identify a set of service instances as the result of a query over selected registry metadata.
- 2880 ebRIM Binding



2881

2882

Figure 26: ebRIM Binding for ServiceQuery

### 2883 2884 <complexType name="ServiceQueryType"> 2885 <complexContent> 2886 <extension base="tns:RegistryEn 2887 <sequence>

Definition

2005	<comptexconcenc></comptexconcenc>
2886	<pre><extension base="tns:RegistryEntryQueryType"></extension></pre>
2887	<sequence></sequence>
2888	<pre><element <="" minoccurs="0" pre="" ref="tns:ServiceFilter"></element></pre>
2889	<pre>maxOccurs="1" /&gt;</pre>
2890	<pre><element <="" minoccurs="0" pre="" ref="tns:ServiceBindingBranch"></element></pre>
2891	<pre>maxOccurs="unbounded" /&gt;</pre>
2892	
2893	
2894	
2895	
2896	<pre><element name="ServiceQuery" type="tns:ServiceQueryType"></element></pre>
2897	
2898	<pre><element name="ServiceQueryResult"></element></pre>
2899	<complextype></complextype>
2900	<choice maxoccurs="unbounded" minoccurs="0"></choice>
2901	<element ref="rim:ObjectRef"></element>
2902	<element ref="rim:RegistryObject"></element>
2903	<element ref="rim:Service"></element>
2904	
2905	
2906	
2907	

### 2908 Semantic Rules

- Let S denote the set of all persistent Service instances in the Registry. The following steps
   will eliminate instances in S that do not satisfy the conditions of the specified filters.
- a) If S is empty then continue to the next numbered rule.

2912 2913 2914	b)	If a ServicetFilter is not specified then go to the next step; otherwise, let x be a service in S. If x does not satisfy the ServiceFilter, then remove x from S. If S is empty then continue to the next numbered rule.
2915 2916	c)	If a ServiceBindingBranch is not specified then continue to the next numbered rule; otherwise, consider each ServiceBindingBranch element separately as follows:
2917		Let SB be the set of all ServiceBinding instances that describe binding of x. Let sb be the
2918		member of SB. If a ServiceBindingFilter element is specified within the
2919		ServiceBindingBranch, and if sb does not satisfy that filter, then remove sb from SB. If
2920		SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
2921		If a SpecificationLinkBranch is not specified within the ServiceBindingBranch then
2922		continue to the next numbered rule; otherwise, consider each SpecificationLinkBranch
2923		element separately as follows:
2924		Let sb be a remaining service binding in SB. Let SL be the set of all specification link
2925		instances sl that describe specification links of sb. If a SpecificationLinkFilter element is
2926		specified within the SpecificationLinkBranch, and if sl does not satisfy that filter, then
2927		remove sl from SL. If SL is empty then remove sb from SB. If SB is empty then remove
2928		x from S. If S is empty then continue to the next numbered rule. If a RegistryObjectQuery
2929		element is specified within the SpecificationLinkBranch then let sl be a remaining
2930		specification link in SL. Treat RegistryObjectQuery element as follows: Let RO be the
2931		result set of the RegistryObjectQuery as defined in Section 8.2.2. If sl is not a
2932		specification link for some registry object in RO, then remove sl from SL. If SL is empty
2933		then remove sb from SB. If SB is empty then remove x from S. If S is empty then
2934		continue to the next numbered rule. If a RegistryEntryQuery element is specified within
2935		the SpecificationLinkBranch then let sl be a remaining specification link in SL. Treat
2936		RegistryEntryQuery element as follows: Let RE be the result set of the
2937 2938		RegistryEntryQuery as defined in Section 8.2.3. If sl is not a specification link for some registry entry in RE, then remove sl from SL. If SL is empty then remove sb from SB. If
2938 2939		SB is empty then remove x from S. If S is empty then continue to the next numbered rule.
2940 2941	d)	Let S be the set of remaining Service instances. Evaluate inherited RegistryEntryQuery over AE as explained in Section 8.2.3.
2942 2943		S is empty, then raise the warning: <i>service query result is empty</i> ; otherwise set S to be the sult of the ServiceQuery.
2944 2945		eturn the result and any accumulated warnings or exceptions (in the RegistryErrorList) thin the RegistryResponse.
2946	Examp	
2947	Examp	
2J+/		
2948	8.2.13	B Registry Filters

## 2949 Purpose

2950 To identify a subset of the set of all persistent instances of a given registry class.

2951	Definition		
2952 2953	<complextype name="FilterType"></complextype>		
_,			

2954	<sequence></sequence>
2955	<pre><element ref="tns:Clause"></element></pre>
2956	
2957	
2958	<pre><element name="RegistryObjectFilter" type="tns:FilterType"></element></pre>
2959	<pre><element name="RegistryEntryFilter" type="tns:FilterType"></element></pre>
2960	<pre><element name="ExtrinsicObjectFilter" type="tns:FilterType"></element></pre>
2961	<pre><element name="RegistryPackageFilter" type="tns:FilterType"></element></pre>
2962	<pre><element name="OrganizationFilter" type="tns:FilterType"></element></pre>
2963	<pre><element name="ClassificationNodeFilter" type="tns:FilterType"></element></pre>
2964	<pre><element name="AssociationFilter" type="tns:FilterType"></element></pre>
2965	<pre><element name="ClassificationFilter" type="tns:FilterType"></element></pre>
2966	<pre><element name="ClassificationSchemeFilter" type="tns:FilterType"></element></pre>
2967	<pre><element name="ExternalLinkFilter" type="tns:FilterType"></element></pre>
2968	<pre><element name="ExternalIdentifierFilter" type="tns:FilterType"></element></pre>
2969	<pre><element name="SlotFilter" type="tns:FilterType"></element></pre>
2970	<pre><element name="AuditableEventFilter" type="tns:FilterType"></element></pre>
2971	<pre><element name="UserFilter" type="tns:FilterType"></element></pre>
2972	<pre><element name="SlotValueFilter" type="tns:FilterType"></element></pre>
2973	<pre><element name="PostalAddressFilter" type="tns:FilterType"></element></pre>
2974	<pre><element name="TelephoneNumberFilter" type="tns:FilterType"></element></pre>
2975	<pre><element name="ServiceFilter" type="tns:FilterType"></element></pre>
2976	<pre><element name="ServiceBindingFilter" type="tns:FilterType"></element></pre>
2977	<pre><element name="SpecificationLinkFilter" type="tns:FilterType"></element></pre>
2978	<pre><element name="LocalizedStringFilter" type="tns:FilterType"></element></pre>
2979	

2980 Semantic Rules

2981 1. The Clause element is defined in Section 8.2.14.

- For every RegistryObjectFilter XML element, the leftArgument attribute of any containing
   SimpleClause shall identify a public attribute of the RegistryObject UML class defined in
   [ebRIM]. If not, raise exception: *object attribute error*. The RegistryObjectFilter returns a set
   of identifiers for RegistryObject instances whose attribute values evaluate to *True* for the
   Clause predicate.
- 3. For every RegistryEntryFilter XML element, the leftArgument attribute of any containing
  SimpleClause shall identify a public attribute of the RegistryEntry UML class defined in
  [ebRIM]. If not, raise exception: *registry entry attribute error*. The RegistryEntryFilter
  returns a set of identifiers for RegistryEntry instances whose attribute values evaluate to *True*for the Clause predicate.
- 4. For every ExtrinsicObjectFilter XML element, the leftArgument attribute of any containing
  SimpleClause shall identify a public attribute of the ExtrinsicObject UML class defined in
  [ebRIM]. If not, raise exception: *extrinsic object attribute error*. The ExtrinsicObjectFilter
  returns a set of identifiers for ExtrinsicObject instances whose attribute values evaluate to *True* for the Clause predicate.
- 5. For every RegistryPackageFilter XML element, the leftArgument attribute of any containing
  SimpleClause shall identify a public attribute of the RegistryPackage UML class defined in
  [ebRIM]. If not, raise exception: *package attribute error*. The RegistryPackageFilter returns
  a set of identifiers for RegistryPackage instances whose attribute values evaluate to *True* for
  the Clause predicate.

- For every OrganizationFilter XML element, the leftArgument attribute of any containing
  SimpleClause shall identify a public attribute of the Organization or PostalAddress UML
  classes defined in [ebRIM]. If not, raise exception: *organization attribute error*. The
  OrganizationFilter returns a set of identifiers for Organization instances whose attribute
  values evaluate to *True* for the Clause predicate.
- For every ClassificationNodeFilter XML element, the leftArgument attribute of any containing SimpleClause shall identify a public attribute of the ClassificationNode UML class defined in [ebRIM]. If not, raise exception: *classification node attribute error*. If the leftAttribute is the visible attribute "path" then if stringPredicate of the StringClause is not "Equal" then raise exception: *classification node path attribute error*. The
- 3012 ClassificationNodeFilter returns a set of identifiers for ClassificationNode instances whose 3013 attribute values evaluate to *True* for the Clause predicate.
- 8. For every AssociationFilter XML element, the leftArgument attribute of any containing
  SimpleClause shall identify a public attribute of the Association UML class defined in
  [ebRIM]. If not, raise exception: *association attribute error*. The AssociationFilter returns a
  set of identifiers for Association instances whose attribute values evaluate to *True* for the
  Clause predicate.
- 3019
  9. For every ClassificationFilter XML element, the leftArgument attribute of any containing
  3020 SimpleClause shall identify a public attribute of the Classification UML class defined in
  3021 [ebRIM]. If not, raise exception: *classification attribute error*. The ClassificationFilter
  3022 returns a set of identifiers for Classification instances whose attribute values evaluate to *True*3023 for the Clause predicate.
- 3024 10. For every ClassificationSchemeFilter XML element, the leftArgument attribute of any
  3025 containing SimpleClause shall identify a public attribute of the ClassificationNode UML
  3026 class defined in [ebRIM]. If not, raise exception: *classification scheme attribute error*. The
  3027 ClassificationSchemeFilter returns a set of identifiers for ClassificationScheme instances
  3028 whose attribute values evaluate to *True* for the Clause predicate.
- 3029 11. For every ExternalLinkFilter XML element, the leftArgument attribute of any containing
  3030 SimpleClause shall identify a public attribute of the ExternalLink UML class defined in
  3031 [ebRIM]. If not, raise exception: *external link attribute error*. The ExternalLinkFilter returns
  3032 a set of identifiers for ExternalLink instances whose attribute values evaluate to *True* for the
  3033 Clause predicate.
- 3034
   12. For every ExternalIdentiferFilter XML element, the leftArgument attribute of any containing
   3035
   SimpleClause shall identify a public attribute of the ExternalIdentifier UML class defined in
   3036
   [ebRIM]. If not, raise exception: *external identifier attribute error*. The
- ExternalIdentifierFilter returns a set of identifiers for ExternalIdentifier instances whose
   attribute values evaluate to *True* for the Clause predicate.
- 3039 13. For every SlotFilter XML element, the leftArgument attribute of any containing
  3040 SimpleClause shall identify a public attribute of the Slot UML class defined in [ebRIM]. If
  3041 not, raise exception: *slot attribute error*. The SlotFilter returns a set of identifiers for Slot
  3042 instances whose attribute values evaluate to *True* for the Clause predicate.

- 3043
  14. For every AuditableEventFilter XML element, the leftArgument attribute of any containing
  3044
  3045
  3045
  3046
  3046
  3046
  3047
  3047
  3047
- 3048
  3048 15. For every UserFilter XML element, the leftArgument attribute of any containing
  3049 SimpleClause shall identify a public attribute of the User UML class defined in [ebRIM]. If
  3050 not, raise exception: *user attribute error*. The UserFilter returns a set of identifiers for User
  3051 instances whose attribute values evaluate to *True* for the Clause predicate.
- 3052 16. SlotValue is a derived, non-persistent class based on the Slot class from ebRIM. There is one SlotValue instance for each "value" in the "values" list of a Slot instance. The visible 3053 3054 attribute of SlotValue is "value". It is a character string. The dynamic instances of SlotValue 3055 are derived from the "values" attribute defined in ebRIM for a Slot instance. For every 3056 SlotValueFilter XML element, the leftArgument attribute of any containing SimpleClause 3057 shall identify the "value" attribute of the SlotValue class just defined. If not, raise exception: 3058 slot element attribute error. The SlotValueFilter returns a set of Slot instances whose "value" 3059 attribute evaluates to *True* for the Clause predicate.
- 3060 17. For every PostalAddressFilter XML element, the leftArgument attribute of any containing
  3061 SimpleClause shall identify a public attribute of the PostalAddress UML class defined in
  3062 [ebRIM]. If not, raise exception: *postal address attribute error*. The PostalAddressFilter
  3063 returns a set of identifiers for PostalAddress instances whose attribute values evaluate to *True*3064 for the Clause predicate.
- 3065 18. For every TelephoneNumberFilter XML element, the leftArgument attribute of any
  3066 containing SimpleClause shall identify a public attribute of the TelephoneNumber UML
  3067 class defined in [ebRIM]. If not, raise exception: *telephone number identity attribute error*.
  3068 The TelephoneNumberFilter returns a set of identifiers for TelephoneNumber instances
  3069 whose attribute values evaluate to *True* for the Clause predicate.
- 3070 19. For every ServiceFilter XML element, the leftArgument attribute of any containing
  3071 SimpleClause shall identify a public attribute of the Service UML class defined in [ebRIM].
  3072 If not, raise exception: *service attribute error*. The ServiceFilter returns a set of identifiers for
  3073 Service instances whose attribute values evaluate to *True* for the Clause predicate.
- 3074
   20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
   3075
   3076
   3076
   3076
   3077
   3077
   3078
   3078
   20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
   3076
   3078
   3078
   20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
   3076
   3078
   3078
   20. For every ServiceBindingFilter XML element, the leftArgument attribute of any containing
   3078
   3078
- 3079 21. For every SpecificationLinkFilter XML element, the leftArgument attribute of any
  3080 containing SimpleClause shall identify a public attribute of the SpecificationLink UML class
  3081 defined in [ebRIM]. If not, raise exception: *specification link attribute error*. The
  3082 SpecificationLinkFilter returns a set of identifiers for SpecificationLink instances whose
  3083 attribute values evaluate to *True* for the Clause predicate.

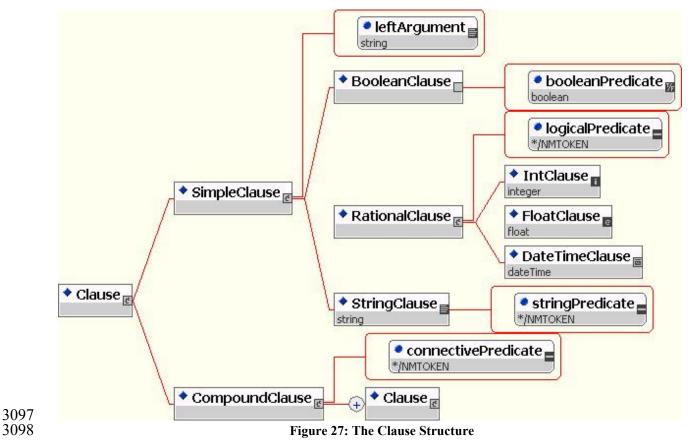
3084
 22. For every LocalizedStringFilter XML element, the leftArgument attribute of any containing
 3085
 3086
 SimpleClause shall identify a public attribute of the LocalizedString UML class defined in
 3086
 [ebRIM]. If not, raise exception: *localized string attribute error*. The LocalizedStringFilter
 3087
 returns a set of identifiers for LocalizedString instances whose attribute values evaluate to
 3088

## 3089 8.2.14 XML Clause Constraint Representation

- 3090 Purpose
- 3091 The simple XML FilterQuery utilizes a formal XML structure based on Predicate Clauses.
- 3092 Predicate Clauses are utilized to formally define the constraint mechanism, and are referred to 3093 simply as Clauses in this specification.

### 3094 Conceptual Diagram

- 3095 The following is a conceptual diagram outlining the Clause structure.
- 3096



### 3099 Semantic Rules

- 3100 Predicates and Arguments are combined into a "LeftArgument Predicate RightArgument"
- 3101 format to form a Clause. There are two types of Clauses: SimpleClauses and CompoundClauses.
- 3102 <u>SimpleClauses</u>
- 3103 A SimpleClause always defines the leftArgument as a text string, sometimes referred to as the

- 3104 Subject of the Clause. SimpleClause itself is incomplete (abstract) and must be extended.
- 3105 SimpleClause is extended to support BooleanClause, StringClause, and RationalClause 3106 (abstract).
- 3107 BooleanClause implicitly defines the predicate as 'equal to', with the right argument as a
- 3108 boolean. StringClause defines the predicate as an enumerated attribute of appropriate string-
- 3109 compare operations and a right argument as the element's text data. Rational number support is
- 3110 provided through a common RationalClause providing an enumeration of appropriate rational
- 3111 number compare operations, which is further extended to IntClause and FloatClause, each with
- 3112 appropriate signatures for the right argument.
- 3113 <u>CompoundClauses</u>
- 3114 A CompoundClause contains two or more Clauses (Simple or Compound) and a connective
- 3115 predicate. This provides for arbitrarily complex Clauses to be formed.

### 3116 Definition

```
3117
3118
             <element name = "Clause">
3119
                <annotation>
3120
                   <documentation xml:lang = "en">
3121
          The following lines define the XML syntax for Clause.
3122
3123
                   </documentation>
3124
                </annotation>
3125
                <complexType>
3126
                   <choice>
3127
                      <element ref = "tns:SimpleClause"/>
3128
                      <element ref = "tns:CompoundClause"/>
3129
                   </choice>
3130
                </complexType>
3131
             </element>
3132
             <element name = "SimpleClause">
3133
                <complexType>
3134
                   <choice>
3135
                      <element ref = "tns:BooleanClause"/>
3136
                      <element ref = "tns:RationalClause"/>
3137
                      <element ref = "tns:StringClause"/>
3138
                   </choice>
3139
                   <attribute name = "leftArgument" use = "required" type =
3140
          "string"/>
3141
                </complexType>
3142
             </element>
3143
             <element name = "CompoundClause">
3144
                <complexType>
3145
                   <sequence>
3146
                      <element ref = "tns:Clause" maxOccurs = "unbounded"/>
3147
                   </sequence>
3148
                   <attribute name = "connectivePredicate" use = "required">
3149
                      <simpleType>
3150
                          <restriction base = "NMTOKEN">
3151
                             <enumeration value = "And"/>
3152
                             <enumeration value = "Or"/>
3153
                          </restriction>
3154
                      </simpleType>
3155
                   </attribute>
3156
                </complexType>
3157
             </element>
```

```
3158
             <element name = "BooleanClause">
3159
                <complexType>
3160
                    <attribute name = "booleanPredicate" use = "required" type =</pre>
3161
          "boolean"/>
3162
                </complexType>
3163
             </element>
3164
             <element name = "RationalClause">
3165
                <complexType>
3166
                   <choice>
3167
                       <element ref = "tns:IntClause"/>
3168
                       <element ref = "tns:FloatClause"/>
3169
                       <element ref = "tns:DateTimeClause"/>
3170
                   </choice>
3171
                   <attribute name = "logicalPredicate" use = "required">
3172
                       <simpleType>
3173
                          <restriction base = "NMTOKEN">
3174
                             <enumeration value = "LE"/>
3175
                             <enumeration value = "LT"/>
3176
                             <enumeration value = "GE"/>
3177
                             <enumeration value = "GT"/>
3178
                             <enumeration value = "EQ"/>
3179
                             <enumeration value = "NE"/>
3180
                          </restriction>
3181
                       </simpleType>
3182
                    </attribute>
3183
                </complexType>
3184
             </element>
3185
             <element name = "IntClause" type = "integer"/>
3186
             <element name = "FloatClause" type = "float"/>
3187
             <element name = "DateTimeClause" type = "dateTime"/>
3188
3189
             <element name = "StringClause">
3190
                <complexType>
3191
                    <simpleContent>
3192
                       <extension base = "string">
3193
                          <attribute name = "stringPredicate" use = "required">
3194
                             <simpleType>
3195
                                <restriction base = "NMTOKEN">
3196
                                      <enumeration value = "Contains"/>
3197
                                      <enumeration value = "-Contains"/>
3198
                                      <enumeration value = "StartsWith"/>
3199
                                      <enumeration value = "-StartsWith"/>
3200
                                      <enumeration value = "Equal"/>
3201
                                      <enumeration value = "-Equal"/>
3202
                                      <enumeration value = "EndsWith"/>
3203
                                      <enumeration value = "-EndsWith"/>
3204
                                </restriction>
3205
                             </simpleType>
3206
                          </attribute>
3207
                       </extension>
3208
                    </simpleContent>
3209
                </complexType>
3210
             </element>
3211
```

### 3212 Examples

3213 Simple BooleanClause: "Smoker" = True

```
3214
3215
          <Clause>
3216
             <SimpleClause leftArgument="Smoker">
3217
                 <BooleanClause booleanPredicate="True"/>
3218
             </SimpleClause>
3219
          </Clause>
3220
       Simple StringClause: "Smoker" contains "mo"
3221
3222
3223
          <Clause>
3224
             <SimpleClause leftArgument = "Smoker">
3225
                 <StringClause stringPredicate = "Contains">mo</StringClause>
3226
             </SimpleClause>
3227
          <Clause>
       Simple IntClause: "Age" >= 7
3228
3229
3230
          <Clause>
3231
             <SimpleClause leftArgument="Age">
3232
                 <RationalClause logicalPredicate="GE">
3233
                    <IntClause>7</IntClause>
3234
                 </RationalClause>
3235
             </SimpleClause>
3236
          </Clause>
3237
       Simple FloatClause: "Size" = 4.3
3238
3239
3240
          <Clause>
3241
             <SimpleClause leftArgument="Size">
3242
                 <RationalClause logicalPredicate="Equal">
3243
                    <FloatClause>4.3</FloatClause>
3244
                 </RationalClause>
3245
             </SimpleClause>
3246
          </Clause>
3247
       Compound with two Simples (("Smoker" = False)AND("Age" =< 45))
3248
3249
3250
          <Clause>
3251
             <CompoundClause connectivePredicate="And">
3252
                <Clause>
3253
                    <SimpleClause leftArgument="Smoker">
3254
                       <BooleanClause booleanPredicate="False"/>
3255
                    </SimpleClause>
3256
                 </Clause>
3257
                 <Clause>
3258
                    <SimpleClause leftArgument="Age">
3259
                       <RationalClause logicalPredicate="LE">
3260
                          <IntClause>45</IntClause>
3261
                       </RationalClause>
3262
                    </SimpleClause>
3263
                 </Clause>
3264
             </CompoundClause>
3265
          </Clause>
3266
```

3267 Coumpound with one Simple and one Compound

```
3268 (("Smoker" = False)And(("Age" =< 45)Or("American"=True)))
```

```
3269
3270
          <Clause>
3271
             <CompoundClause connectivePredicate="And">
3272
                <Clause>
3273
                    <SimpleClause leftArgument="Smoker">
3274
                       <BooleanClause booleanPredicate="False"/>
3275
                    </SimpleClause>
3276
                </Clause>
3277
                <Clause>
3278
                    <CompoundClause connectivePredicate="Or">
3279
                       <Clause>
3280
                          <SimpleClause leftArgument="Age">
3281
                             <RationalClause logicalPredicate="LE">
3282
                                <IntClause>45</IntClause>
3283
                             </RationalClause>
3284
                          </SimpleClause>
3285
                       </Clause>
3286
                       <Clause>
3287
                          <SimpleClause leftArgument="American">
3288
                             <BooleanClause booleanPredicate="True"/>
3289
                          </SimpleClause>
3290
                       </Clause>
3291
                   </CompoundClause>
3292
                </Clause>
3293
             </CompoundClause>
3294
          <Clause>
3295
```

# 3296 8.3 SQL Query Support

The Registry may optionally support an SQL based query capability that is designed for Registry
clients that demand more advanced query capability. The optional SQLQuery element in the
AdhocQueryRequest allows a client to submit complex SQL queries using a declarative query
language.

3301 The syntax for the SQLQuery of the Registry is defined by a stylized use of a proper subset of

the "SELECT" statement of Entry level SQL defined by ISO/IEC 9075:1992, Database

3303 Language SQL [SQL], extended to include <sql invoked routines> (also known as

3304 stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM] and pre-defined routines defined

- in template form in Appendix D.3. The syntax of the Registry query language is defined by the
- BNF grammar in D.1.
- Note that the use of a subset of SQL syntax for SQLQuery does not imply a requirement to userelational databases in a Registry implementation.

## 3309 8.3.1 SQL Query Syntax Binding To [ebRIM]

3310 SQL Queries are defined based upon the query syntax in in Appendix D.1 and a fixed relational

3311 schema defined in Appendix D.3. The relational schema is an algorithmic binding to [ebRIM] as

3312 described in the following sections.

### 3313 8.3.1.1 Class Binding

- A subset of the class names defined in [ebRIM] map to table names that may be queried by an
- 3315 SQL query. Appendix D.3 defines the names of the ebRIM classes that may be queried by an3316 SQL query.
- The algorithm used to define the binding of [ebRIM] classes to table definitions in Appendix D.3is as follows:
- Classes that have concrete instances are mapped to relational tables. In addition entity classes
   (e.g. PostalAddress and TelephoneNumber) are also mapped to relational tables.
- The intermediate classes in the inheritance hierarchy, namely RegistryObject and RegistryEntry, map to relational views.
- The names of relational tables and views are the same as the corresponding [ebRIM] class name. However, the name binding is case insensitive.
- Each [ebRIM] class that maps to a table in Appendix D.3 includes column definitions in Appendix D.3 where the column definitions are based on a subset of attributes defined for that class in [ebRIM]. The attributes that map to columns include the inherited attributes for the [ebRIM] class. Comments in Appendix D.3 indicate which ancestor class contributed which column definitions.
- An SQLQuery against a table not defined in Appendix D.3 may raise an error condition:InvalidQueryException.
- The following sections describe the algorithm for mapping attributes of [ebRIM] to SQLcolumndefinitions.

## 3334 8.3.1.2 Primitive Attributes Binding

- Attributes defined by [ebRIM] that are of primitive types (e.g. String) may be used in the same
- 3336 way as column names in SQL. Again the exact attribute names are defined in the class
- definitions in [ebRIM]. Note that while names are in mixed case, SQL-92 is case insensitive. It is
- 3338 therefore valid for a query to contain attribute names that do not exactly match the case defined 3339 in [ebRIM].
- 3340 8.3.1.3 Reference Attribute Binding
- A few of the [ebRIM] class attributes are of type UUID and are a reference to an instance of a
- class defined by [ebRIM]. For example, the accessControlPolicy attribute of the RegistryObject
   class returns a reference to an instance of an AccessControlPolicy object.
- 5343 class returns a reference to an instance of an AccessControlPolicy object.
- 3344 In such cases the reference maps to the id attribute for the referenced object. The name of the
- resulting column is the same as the attribute name in [ebRIM] as defined by 8.3.1.2. The data
- type for the column is VARCHAR(64) as defined in Appendix D.3.
- 3347 When a reference attribute value holds a null reference, it maps to a null value in the SQL
- binding and may be tested with the <null specification> ("IS [NOT] NULL" syntax) as definedby [SQL].
- 3350 Reference attribute binding is a special case of a primitive attribute mapping.

## 3351 8.3.1.4 Complex Attribute Binding

A few of the [ebRIM] interfaces define attributes that are not primitive types. Instead they are of

- a complex type as defined by an entity class in [ebRIM]. Examples include attributes of type
- 3354 TelephoneNumber, Contact, PersonName etc. in class Organization and class User.
- 3355 The SQL query schema does not map complex attributes as columns in the table for the class for
- which the attribute is defined. Instead the complex attributes are mapped to columns in the table
- for the domain class that represents the data type for the complex attribute (e.g.
- 3358 TelephoneNumber). A column links the row in the domain table to the row in the parent table
- 3359 (e.g. User). An additional column named 'attribute\_name' identifies the attribute name in the
- parent class, in case there are multiple attributes with the same complex attribute type.
- This mapping also easily allows for attributes that are a collection of a complex type. For
- 3362 example, a User may have a collection of TelephoneNumbers. This maps to multiple rows in the
- 3363 TelephoneNumber table (one for each TelephoneNumber) where each row has a parent identifier
- and an attribute\_name.
- 3365 8.3.1.5 Binding of Methods Returning Collections
- 3366 Several of the [ebRIM] classes define methods in addition to attributes, where these methods
- return collections of references to instances of classes defined by [ebRIM]. For example, the
- 3368 getPackages method of the ManagedObject class returns a Collection of references to instances
- 3369 of Packages that the object is a member of.
- 3370 Such collection returning methods in [ebRIM] classes have been mapped to stored procedures in
- Appendix D.3 such that these stored procedures return a collection of id attribute values. The
- returned value of these stored procedures can be treated as the result of a table sub-query in SQL.
- These stored procedures may be used as the right-hand-side of an SQL IN clause to test for membership of an object in such collections of references.

## 3375 8.3.2 Semantic Constraints On Query Syntax

- This section defines simplifying constraints on the query syntax that cannot be expressed in the BNF for the query syntax. These constraints must be applied in the semantic analysis of the query.
- 1. Class names and attribute names must be processed in a case insensitive manner.
- 3380
  3381
  2. The syntax used for stored procedure invocation must be consistent with the syntax of an SQL procedure invocation as specified by ISO/IEC 9075-4 [SQL/PSM].
- 3382 3. For this version of the specification, the SQL select column list consists of exactly one
  column, and must always be t.id, where t is a table reference in the FROM clause.
- Join operations must be restricted to simple joins involving only those columns that have an
  index defined within the normative SQL schema. This constraint is to prevent queries that
  may be computationally too expensive.

## 3387 8.3.3 SQL Query Results

- 3388 The result of an SQL query resolves to a collection of objects within the registry. It never
- resolves to partial attributes. The objects related to the result set may be returned as an
- 3390 ObjectRef, RegistryObject, RegistryEntry or leaf ebRIM class depending upon the
- 3391 responseOption parameter specified by the client on the AdHocQueryRequest. The entire result

3392 set is returned as a SQLQueryResult as defined by the AdHocQueryResponse in Section 8.1.

#### 3393 8.3.4 Simple Metadata Based Queries

3394 The simplest form of an SQL query is based upon metadata attributes specified for a single class 3395 within [ebRIM]. This section gives some examples of simple metadata based queries.

3396 For example, to get the collection of ExtrinsicObjects whose name contains the word 'Acme' 3397 and that have a version greater than 1.3, the following query must be submitted:

```
3398
3399
3400
3401
         SELECT eo.id from ExtrinsicObject eo, Name nm where nm.value LIKE '%Acme%' AND
                  eo.id = nm.parent AND
                  eo.majorVersion >= 1 AND
 402
                  (eo.majorVersion >= 2 OR eo.minorVersion > 3);
340<del>3</del>
```

3404 Note that the query syntax allows for conjugation of simpler predicates into more complex queries as shown in the simple example above. 3405

#### 3406 8.3.5 RegistryObject Queries

The schema for the SQL query defines a special view called RegistryObject that allows doing a 3407 3408 polymorphic query against all RegistryObject instances regardless of their actual concrete type or 3409 table name.

3410 The following example is the similar to that in Section 8.3.4 except that it is applied against all 3411 RegistryObject instances rather than just ExtrinsicObject instances. The result set will include id 3412 for all qualifying RegistryObject instances whose name contains the word 'Acme' and whose description contains the word "bicycle". 3413 3414 3415 3416

```
SELECT ro.id from RegistryObject ro, Name nm, Description d where nm.value LIKE '%Acme%' AND
       d.value LIKE '%bicycle%' AND
       ro.id = nm.parent AND ro.id = d.parent;
```

#### 3419 8.3.6 RegistryEntry Queries

3417 3418

3420 The schema for the SQL query defines a special view called RegistryEntry that allows doing a 3421 polymorphic query against all RegistryEntry instances regardless of their actual concrete type or 3422 table name.

3423 The following example is the same as Section 8.3.4 except that it is applied against all

- 3424 RegistryEntry instances rather than just ExtrinsicObject instances. The result set will include id
- 3425 for all qualifying RegistryEntry instances whose name contains the word 'Acme' and that have a

3426 3427 3428 3429 3430 version greater than 1.3.

```
SELECT re.id from RegistryEntry re, Name nm where nm.value LIKE '%Acme%' AND
       re.id = nm.parent AND
       re.majorVersion >= 1 AND
       (re.majorVersion >= 2 OR re.minorVersion > 3);
```

#### 3433 8.3.7 Classification Queries

3434 This section describes the various classification related queries that must be supported.

#### 3435 8.3.7.1 Identifying ClassificationNodes

3436 Like all objects in [ebRIM], ClassificationNodes are identified by their ID. However, they may

- 3437 also be identified as a path attribute that specifies an XPATH expression [XPT] from a root
- 3438 classification node to the specified classification node in the XML document that would
- 3439 represent the ClassificationNode tree including the said ClassificationNode.

#### 3440 8.3.7.2 Getting ClassificationSchemes

3441 To get the collection of ClassificationSchemes the following query predicate must be supported: 3442 3443 3444

SELECT scheme.id FROM ClassificationScheme scheme;

3445 The above query returns all ClassificationSchemes. Note that the above query may also specify 3446 additional predicates (e.g. name, description etc.) if desired.

#### 3447 8.3.7.3 Getting Children of Specified ClassificationNode

3448 To get the children of a ClassificationNode given the ID of that node the following style of query 3449 must be supported:

3450 3451 3452 SELECT cn.id FROM ClassificationNode cn WHERE parent = <id>

3453 The above query returns all ClassificationNodes that have the node specified by  $\langle id \rangle$  as their 3454 parent attribute.

#### 3455 8.3.7.4 Getting Objects Classified By a ClassificationNode

3456 To get the collection of ExtrinsicObjects classified by specified ClassificationNodes the 3457 following style of query must be supported: 3458 3459 3460 3461 3462

```
SELECT id FROM ExtrinsicObject
WHERE
   id IN (SELECT classifiedObject FROM Classification
         WHERE
              classificationNode IN (SELECT id FROM ClassificationNode
                                   WHERE path = '/Geography/Asia/Japan'))
 AND
  id IN (SELECT classifiedObject FROM Classification
         WHERE
              classificationNode IN (SELECT id FROM ClassificationNode
                                    WHERE path = '/Industry/Automotive'))
```

- 3471 The above query gets the collection of ExtrinsicObjects that are classified by the Automotive
- 3472 Industry and the Japan Geography. Note that according to the semantics defined for
- 3473 GetClassifiedObjectsRequest, the query will also contain any objects that are classified by

3474 descendents of the specified ClassificationNodes.

3463

3464

3465

<u>3466</u>

3467

3468

3469

3470

#### 3475 8.3.7.5 Getting Classifications That Classify an Object

3476 To get the collection of Classifications that classify a specified Object the following style of 3477 query must be supported:

```
SELECT id FROM Classification c
       WHERE c.classifiedObject = <id>;
```

3482	8.3.8 Association Queries
3483	This section describes the various Association related queries that must be supported.
3484	8.3.8.1 Getting All Association With Specified Object As Its Source
3485 3486	To get the collection of Associations that have the specified Object as its source, the following query must be supported:
3486 3487 3488 3489	SELECT id FROM Association WHERE sourceObject = <id></id>
3490	8.3.8.2 Getting All Association With Specified Object As Its Target
3491 3492 3493	To get the collection of Associations that have the specified Object as its target, the following query must be supported:
3495 3494 3495	SELECT id FROM Association WHERE targetObject = <id></id>
3496	8.3.8.3 Getting Associated Objects Based On Association Attributes
3497 3498	To get the collection of Associations that have specified Association attributes, the following queries must be supported:
3499 3500	Select Associations that have the specified name.
3500 3501 3502	SELECT id FROM Association WHERE name = <name></name>
3503 3504 3505	Select Associations that have the specified association type, where association type is a string containing the corresponding field name described in [ebRIM].
3504 3505 3506 3507 3508	SELECT id FROM Association WHERE associationType = <associationtype></associationtype>
3509	8.3.8.4 Complex Association Queries
3510 3511 3512	The various forms of Association queries may be combined into complex predicates. The following query selects Associations that have a specific sourceObject, targetObject and associationType:
3513 3514 3515 3516 3517 3518	<pre>SELECT id FROM Association WHERE sourceObject = <idl> AND targetObject = <idl> AND associationType = <associationtype>;</associationtype></idl></idl></pre>
3519	8.3.9 Package Queries
2520	

To find all Packages that a specified RegistryObject belongs to, the following query is specified: select id FROM Package WHERE id IN (RegistryObject\_packages(<id>));

## 3524 8.3.9.1 Complex Package Queries

The following query gets all Packages that a specified object belongs to, that are not deprecated and where name contains "RosettaNet."

3543

3552

```
SELECT id FROM Package p, Name n WHERE
      p.id IN (RegistryObject packages(<id>)) AND
       nm.value LIKE '%RosettaNet%' AND nm.parent = p.id AND
       p.status <> 'Deprecated'
```

#### 3533 8.3.10 ExternalLink Queries

3534 To find all ExternalLinks that a specified ExtrinsicObject is linked to, the following query is 3535 specified: 3536 3537 3538

SELECT id From ExternalLink WHERE id IN (RegistryObject externalLinks(<id>))

3539 To find all ExtrinsicObjects that are linked by a specified ExternalLink, the following query is 3540 specified: 3541 3542

SELECT id From ExtrinsicObject WHERE id IN (RegistryObject\_linkedObjects(<id>))

#### 3544 8.3.10.1 Complex ExternalLink Queries

3545 The following query gets all ExternalLinks that a specified ExtrinsicObject belongs to, that 3546 contain the word 'legal' in their description and have a URL for their externalURI. 3547 3548 3549

```
SELECT id FROM ExternalLink WHERE
               id IN (RegistryObject externalLinks(<id>)) AND
3550
               description LIKE '%legal%' AND
               externalURI LIKE `%http://%'
```

#### 3553 8.3.11 Audit Trail Queries

3554 To get the complete collection of AuditableEvent objects for a specified ManagedObject, the 3555 following query is specified:

SELECT id FROM AuditableEvent WHERE registryObject = <id>

#### 8.4 Content Retrieval 3559

3560 A client retrieves content via the Registry by sending the GetContentRequest to the

3561 QueryManager. The GetContentRequest specifies a list of Object references for Objects that

need to be retrieved. The OueryManager returns the specified content by sending a 3562

3563 GetContentResponse message to the RegistryClient interface of the client. If there are no errors

3564 encountered, the GetContentResponse message includes the specified content as additional

- 3565 payloads within the message. In addition to the GetContentResponse payload, there is one
- 3566 additional payload for each content that was requested. If there are errors encountered, the
- 3567 RegistryResponse payload includes an error and there are no additional content specific 3568 payloads.
- 3569 8.4.1 Identification Of Content Payloads

3570 Since the GetContentResponse message may include several repository items as additional 3571 payloads, it is necessary to have a way to identify each payload in the message. To facilitate this

- 3572 identification, the Registry must do the following:
- Use the ID of the ExtrinsicObject, as the value of the Content-ID header field for the mimepart that contains the corresponding repository item for the ExtrinsicObject
- In case of [ebMS] transport, use the ID for each RegistryObject instance that describes the repository item in the Reference element for that object in the Manifest element of the ebXMLHeader.

### 3578 8.4.2 GetContentResponse Message Structure

The following message fragment illustrates the structure of the GetContentResponse Message that is returning a Collection of CPPs as a result of a GetContentRequest that specified the IDs for the requested objects.

```
Content-type: multipart/related; boundary="Boundary"; type="text/xml";
--BoundarY
Content-ID: <GetContentRequest@example.com>
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'</pre>
  xmlns:eb= 'http://www.oasis-open.org/committees/ebxml-msg/schema/draft-msg-header-03.xsd'>
<SOAP-ENV:Header>
... ebMS header goes here if using ebMS
</SOAP-ENV:Header>
<SOAP-ENV:Body>
... ebMS manifest gooes here if using ebMS
<?xml version="1.0" encoding="UTF-8"?>
<GetContentRequest>
  <ObjectRefList>
     <ObjectRef id="d8163dfb-f45a-4798-81d9-88aca29c24ff" .../>
      <ObjectRef id="212c3a78-1368-45d7-acc9-a935197e1e4f" .../>
   </ObjectRefList>
</GetContentRequest>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
--Boundary
Content-ID: d8163dfb-f45a-4798-81d9-88aca29c24ff
Content-Type: text/xml
<?xml version="1.0" encoding="UTF-8"?>
<CPP>
</CPP>
--Boundarv--
Content-ID: 212c3a78-1368-45d7-acc9-a935197e1e4f
Content-Type: text/xml
<CPP>
</CPP>
--Boundary-
```

# 3633 9 Registry Security

This chapter describes the security features of the ebXML Registry. It is assumed that the reader is familiar with the security related classes in the Registry information model as described in [ebRIM]. Security glossary terms can be referenced from RFC 2828.

## 3637 9.1 Security Concerns

In the current version of this specification, we address data integrity and source integrity (item 1 in Appendix F.1). We have used a minimalist approach to address the access control concern as in item 2 of Appendix F.1. Essentially, "any known entity (Submitting Organization) can publish content and anyone can view published content." The Registry information model has been designed to allow more sophisticated security policies in future versions of this specification.

## **3643 9.2 Integrity of Registry Content**

3644 It is assumed that most business registries do not have the resources to validate the veracity of

the content submitted to them. "The mechanisms described in this section can be used to ensure

that any tampering with the content submitted by a Submitting Organization can be detected.

3647 Furthermore, these mechanisms support unambiguous identification of the Responsible

3648 Organization for any registry content. The Registry Client has to sign the contents before

submission – otherwise the content will be rejected. Note that in the discussions in this section
 we assume a Submitting Organization to be also the Responsible Organization. Future version of

3650 we assume a Submitting Organization to be also the Responsible Organization. Future version of 3651 this specification may provide more examples and scenarios where a Submitting Organization

3651 this specification may provide more examples and scenarios where a Submitting Organization

3652 and Responsible Organization are different.

## 3653 9.2.1 Message Payload Signature

The integrity of the Registry content requires that all submitted content be signed by the Registry client. The signature on the submitted content ensures that:

- Any tampering of the content can be detected.
- The content's veracity can be ascertained by its association with a specific Submitting
   Organization.

3659 This section specifies the requirements for generation, packaging and validation of payload

3660 signatures. A payload signature is packaged with the payload. Therefore the requirements apply

3661 regardless of whether the Registry Client and the Registration Authority communicate over

vanilla SOAP with Attachments or ebXML Messaging Service [ebMS]. Currently, ebXML

3663 Messaging Service does not specify the generation, validation and packaging of payload

3664 signatures. The specification of payload signatures is left upto the application (such as Registry).

3665 So the requirements on the payload signatures augment the [ebMS] specification.

## 3666 Use Case

This Use Case illustrates the use of header and payload signatures (we discuss header signatureslater).

- RC1 (Registry Client 1) signs the content (generating a payload signature) and publishes the content along with the payload signature to the Registry.
- RC2 (Registry Client 2) retrieves RC1's content from the Registry.

3672	• RC2 wants to verify that RC1 published the content. In order to do this, when RC2 retrieves
3673	the content, the response from the Registration Authority to RC2 contains the following:
3674	<ul> <li>Payload containing the content that has been published by RC1.</li> </ul>
3675 3676	<ul> <li>RC1's payload signature (represented by a ds:Signature element) over RC1's published content.</li> </ul>
3677 3678 3679	<ul> <li>The public key for validating RC1's payload signature in ds:Signature element ( using the KeyInfo element as specified in [XMLDSIG] ) so RC2 can obtain the public key for signature (e.g. retrieve a certificate containing the public key for RC1).</li> </ul>
3680 3681	<ul> <li>A ds:Signature element containing the header signature. Note that the Registration Authority (not RC1) generates this signature.</li> </ul>
3682	9.2.2 Payload Signature Requirements
3683	9.2.2.1 Payload Signature Packaging Requirements
3684 3685 3686	A payload signature is represented by a ds:Signature element. The payload signature must be packaged with the payload as specified here. This packaging assumes that the payload is always signed.
3687 3688	• The payload and its signature must be enclosed in a MIME multipart message with a Content-Type of multipart/Related.
3689 3690	• The first body part must contain the XML signature as specified in Section 9.2.2.2, "Payload Signature Generation Requirements".
3691	• The second through n <sup>th</sup> body part must be the content.
3692 3693	The packaging of the payload signature with one payload is as follows:
3694	MIME-Version: 1.0
3695	Content-Type: multipart/Related; boundary=MIME_boundary; type=text/xml;
3696	Content-Description: ebXML Message
3697	
3698	MIME boundary
3699	Content-Type: text/xml; charset=UTF-8
3700	Content-Transfer-Encoding: 8bit
3701	Content-ID: http://claiming-it.com/claim061400a.xml
3702	
3703	xml version='1.0' encoding="utf-8"?
3704	<soap-env: envelope=""></soap-env:>
3705	
3706	SOAP-ENV: Envelope>
3707	
3708	MIME_boundary
3709	Content-Type: multipart/Related; boundary=PAYLOAD_boundary
3710	

- 3711 --PAYLOAD\_boundary
- 3712 Content-Type: text/xml; charset=UTF-8
- 3713 Content-Transfer-Encoding: 8bit
- 3714 Content-ID: payload1

3715	
	<pre><ds:signature></ds:signature></pre>
3716	Payload signature
3717	
3718	
3719	PAYLOAD_boundary
3720	Content-Type: text/xml; charset=UTF-8
3721	Content-Transfer-Encoding: 8bit
3722	Content-ID: payload2
3723	<submitobjectsrequest></submitobjectsrequest>
3724	MIME_boundary
3725	
3726	9.2.2.2 Payload Signature Generation Requirements
3727 3728	The ds:Signature element [XMLDSIG] for a payload signature must be generated as specified in this section. Note: the "ds" name space reference is to http://www.w3.org/2000/09/xmldsig#
3729 3730 3731 3732 3733 3733	<ul> <li>ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. [XMLDSIG] allows more than one Algorithm attribute, and a client may use any of these attributes. However, signing using the following Algorithm attribute: <u>http://www.w3.org/2000/09/xmldsig/#dsa-sha1</u> will allow interoperability with all XMLDSIG compliant implementations, since XMLDSIG requires the implementation of this algorithm.</li> </ul>
3735 3736 3737	The ds:SignatureMethod element must contain a ds:CanonicalizationMethod element. The following Canonicalization algorithm (specified in [XMLDSIG]) must be supported http://www.w3.org/TR/2001/REC-xml-c14n-2001315
3738 3739	• One ds:Reference element to reference each of the payloads that needs to be signed must be created. The ds:Reference element:
3740 3741	<ul> <li>Must identify the payload to be signed using the URI attribute of the ds:Reference element.</li> </ul>
3742 3743 3744	<ul> <li>Must contain the <ds:digestmethod> as specified in [XMLDSIG]. A client must be support the following digest algorithm:         <u>http://www.w3.org/2000/09/xmldsig/#sha1</u></ds:digestmethod></li> </ul>
3745	<ul> <li>Must contain a <ds:digestvalue> which is computed as specified in [XMLDSIG].</ds:digestvalue></li> </ul>
	The ds:SignedValue must be generated as specified in [XMLDSIG].
3747 3748	The ds:KeyInfo element may be present. However, when present, the ds:KeyInfo field is subject to the requirements stated in Section 9.4, "KeyDistrbution and KeyInfo element".
	9.2.2.3 Message Payload Signature Validation
3750	The ds:Signature element must be validated by the Registry as specified in the [XMLDSIG].
3751	9.2.2.4 Payload Signature Example
3752 3753	The following example shows the format of the payload signature:
3754	<ds:signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"></ds:signature>
3755	<ds:signedinfo></ds:signedinfo>
3756	<signaturemethod algorithm="&lt;u&gt;http://www.w3.org/TR/2000/09/xmldsig#dsa-shal&lt;/u&gt;"></signaturemethod>
	······································

~	
3757	<ds:canonicalizationmethod></ds:canonicalizationmethod>
3758	Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
3759	
3760	<ds:reference uri="#Payload1"></ds:reference>
3761	<ds:digestmethod digestalgorithm="http://www.w3.org/TR/2000/09/xmldsig#shal"></ds:digestmethod>
3762	<ds:digestvalue> </ds:digestvalue>
3763	
3764	
3765	<ds:signaturevalue> </ds:signaturevalue>
3766	
3767	

## 3768 9.3 Authentication

3769 The Registry must be able to authenticate the identity of the Principal associated with client

- 3770 requests. The identity of the Principal can be identified by verifying the message header
- 3771 signature with the certificate of the Principal. The certificate may be in the message itself or
- 3772 provided to the registry through means unspecified in this specification. If not provided in the
- 3773 message, this specification does not specify how the Registry correlates a specific message with
- a certificate. Authentication of each payload must also be possible by using the signature
- 3775 associated with each payload. Authentication is also required to identify the "privileges" a
- 3776 Principal is authorized ("authorization") to have with respect to specific objects in the Registry.
- 3777 The Registry must perform authentication on a per message basis. From a security point of view,
- all messages are independent and there is no concept of a session encompassing multiple
- 3779 messages or conversations. Session support may be added as an optimization feature in future
- 3780 versions of this specification.
- 3781 It is important to note that the message header signature can only guarantee data integrity and it
- 3782 may be used for Authentication knowing that it is vulnerable to replay types of attacks. True
- 3783 support for authentication requires timestamps or nonce (nonrecurring series of numbers to
- 3784 identify each message) that are signed.

## 3785 9.3.1 Message Header Signature

- Message headers are signed to provide data integrity while the message is in transit. Note that the signature within the message header also signs the digests of the payloads.
- 3788 Header Signature Requirements
- 3789 Message headers can be signed and are referred to as a header signature. This section specifies
- the requirements for generation, packaging and validation of a header signature. These
- 3791 requirements apply when the Registry Client and Registration Authority communicate using
- 3792 vanilla SOAP with Attachments. When ebXML MS is used for communication, then the
- 3793 message handler (i.e. [ebMS]) specifies the generation, packaging and validation of XML
- 3794 signatures in the SOAP header. Therefore the header signature requirements do not apply when
- the ebXML MS is used for communication. However, payload signature generation requirements
- 3796 (specified elsewhere in this document) do apply whether vanilla SOAP with Attachments or
- 3797 ebXML MS is used for communication.

### 3798 **9.3.1.1 Packaging Requirements**

A header signature is represented by a ds:Signature element. The ds:Signature element generated
must be packaged in a <SOAP-ENV:Header> element. The packaging of the ds:Signature
element in the SOAP header field is shown below.

3802 3803 MIME-Version: 1.0 3804 Content-Type: Multipart/Related; boundary=MIME boundary; type=text/xml; 3805 Content-Description: ebXML Message 3806 3807 -- MIME boundary 3808 Content-Type: text/xml; charset=UTF-8 3809 Content-Transfer-Encoding: 8bit 3810 Content-ID: http://claiming-it.com/claim061400a.xml 3811 3812 <?xml version='1.0' encoding="utf-8"?> 3813 <SOAP-ENV:Envelope 3814 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"> 3815 <SOAP-ENV:Header> 3816 <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#"> 3817 ...signature over soap envelope 3818 </ds:Signature> 3819 </SOAP-ENV: Header> 3820 <SOAP-ENV: Body> 3821 3822 </SOAP-ENV: Body> 3823 </SOAP-ENV: Envelope> 3824

- 3825 9.3.1.2 Header Signature Generation Requirements
- The ds:Signature element [XMLDSIG] for a header signature must be generated as specified in this section. A ds:Signature element contains:
- 3828 ds:SignedInfo
- 3829 ds:SignatureValue
- 3830 ds:KeyInfo
- 3831 The ds:SignedInfo element must be generated as follows:
- ds:SignatureMethod must be present. [XMLDSIG] requires that the algorithm be identified using the Algorithm attribute. While [XMLDSIG] allows more than one Algorithm Attribute, a client must be capable of signing using only the following Algorithm attribute: <u>http://www.w3.org/2000/09/xmldsig/#dsa-sha1</u> This algorithm is being chosen because all XMLDSIG implementations conforming to the [XMLDSIG] specification support it.
- 3837 2. The ds:SignatureMethod elment must contain a ds:CanonicalizationMethod element. The
   3838 following Canonicalization algorithm (specified in [XMLDSIG]) must be supported:
- 3839 http://www.w3.org/TR/2001/REC-xml-c14n-20010315

- 3840 3. A ds:Reference element to include the <SOAP-ENV:Envelope> in the signature calculation.
- 3841 This signs the entire ds:Reference element and:
- 3842 Must include the following ds:Transform:
- 3843 <u>http://www.w3.org/2000/09/xmldsig#enveloped-signature</u>
- 3844 This ensures that the signature (which is embedded in the <SOAP-ENV:Header> 3845 element) is not included in the signature calculation.
- 3846 Must identify the <SOAP-ENV:Envelope> element using the URI attribute of the
   3847 ds:Reference element (The URI attribute is optional in the [XMLDSIG] specification.).
   3848 The URI attribute must be "".
- 3849 Must contain the <ds:DigestMethod> as specified in [XMLDSIG]. A client must support the following digest algorithm: <u>http://www.w3.org/2000/09/xmldsig/#sha1</u>
- Must contain a <ds:DigestValue>, which is computed as specified in [XMLDSIG].
- 3852 The ds:SignedValue must be generated as specified in [XMLDSIG].
- The ds:KeyInfo element may be present. When present, it is subject to the requirements stated in Section 9.4, "KeyDistrbution and KeyInfo element".
- 3855 9.3.1.3 Header Signature Validation Requirements
- The ds:Signature element for the ebXML message header must be validated by the recipient as specified by [XMLDSIG].
- 3858 9.3.1.4 Header Signature Example

```
3859
         The following example shows the format of a header signature:
3860
3861
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
3862
            <ds:SignedInfo>
3863
                <SignatureMethod Algorithm=<u>http://www.w3.org/TR/2000/09/xmldsig#dsa-sha1/></u>
3864
                <ds:CanonicalizationMethod>
3865
                    Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-2001026">
3866
                </ds:CanonicalizationMethod>
3867
                <ds:Reference URI= "">
3868
                    <ds.Transform>
3869
                        http://www.w3.org/2000/09/xmldsig#enveloped-signature
3870
                    </ds:Transform>
3871
                    <ds:DigestMethod DigestAlgorithm="./xmldsig#sha1">
3872
                    <ds:DigestValue> ... </ds:DigestValue>
3873
                </ds:Reference>
3874
             </ds:SignedInfo>
3875
             <ds:SignatureValue> ... </ds:SignatureValue>
3876
         </ds:Signature>
3877
```

# 3878 9.4 Key Distribution and KeyInfo Element

3879 To validate a signature, the recipient of the signature needs the public key corresponding to the 3880 signer's public key. The participants may use the KeyInfo field of ds:Signature, or distribute the public keys out-of-band. In this section we consider the case when the public key is sent in theKeyInfo field. The following use cases need to be handled:

- Registration Authority needs the public key of the Registry Client to validate the signature
- Registry Client needs the public key of the Registration Authority to validate the Registry's signature.
- Registry Client RC1 needs the public key of Registry Client (RC2) to validate the content signed by RC1.
- [XMLDSIG] provides a ds:*KeyInfo* element that can be used to pass the recipient
   information for retrieving the public key. ds:*KeyInfo* is an optional element as specified in
   [XMLDSIG]. This field together with the procedures outlined in this section is used to
   securely pass the public key to a recipient. ds:Keyinfo can be used to pass information such
   as keys, certificates, names etc. The intended usage of KeyInfo field is to send the X509
   Certificate, and subsequently extract the public key from the certificate. Therefore, the
   KeyInfo field must contain a X509 Certificate as specified in [XMLDSIG], if the KeyInfo
- 3895 field is present.
- 3896 The following assumptions are also made:
- 3897 1. A Certificate is associated both with the Registration Authority and a Registry Client.
- 38982. A Registry Client registers its certificate with the Registration Authority. The mechanism3899 used for this is not specified here.
- 3900 3. A Registry Client obtains the Registration Authority's certificate and stores it in its own local
   3901 key store. The mechanism is not specified here.
- 3902 Couple of scenarios on the use of KeyInfo field is in Appendix F.8.

# 3903 9.5 Confidentiality

## 3904 9.5.1 On-the-wire Message Confidentiality

It is suggested but not required that message payloads exchanged between clients and the
 Registry be encrypted during transmission. This specification does not specify how payload
 encryption is to be done.

## 3908 9.5.2 Confidentiality of Registry Content

- 3909 In the current version of this specification, there are no provisions for confidentiality of Registry
- 3910 content. All content submitted to the Registry may be discovered and read by any client. This
- implies that the Registry and the client need to have an a priori agreement regarding encryption
- algorithm, key exchange agreements, etc. This service is not addressed in this specification.

## 3913 9.6 Authorization

- 3914 The Registry must provide an authorization mechanism based on the information model defined
- in [ebRIM]. In this version of the specification the authorization mechanism is based on a default
- 3916 Access Control Policy defined for a pre-defined set of roles for Registry users. Future versions of
- 3917 this specification will allow for custom Access Control Policies to be defined by the Submitting
- 3918 Organization. The authorization is going to be applied on a specific set of privileges. A

3919 privelege is the ability to carry a specific action.

## 3920 9.6.1 Actions

- 3921Life Cycle Actions
- 3922 submitObjects
- 3923 updateObjects
- 3924 addSlots
- 3925 removeSlots
- 3926 approveObjects
- 3927 deprecateObjects
- 3928 removeObjects
- 3929 Read Actions
- 3930 The various getXXX() methods in QueryManagement Service.

## **3931 9.7 Access Control**

- 3932 The Registry must create a default AccessControlPolicy object that grants the default
- 3933 permissions to Registry users based upon their assigned role. The following table defines the
- 3934 Permissions granted by the Registry to the various pre-defined roles for Registry users based
- 3935 upon the default AccessControlPolicy. Note that we have "ContentOwner" as a role. This role
- 3936 maps to the Submitting Organization in the current version of the specification.
- 3937

### **Table 11: Default Access Control Policies**

Role	Permissions
ContentOwner	Access to <i>all</i> methods on Registry Objects that are owned by the ContentOwner.
RegistryAdministrator	Access to all methods on all Registry Objects
RegistryGuest	Access to <i>all</i> read-only (getXXX) methods on <i>all</i> Registry Objects (read-only access to all content).

3938 The Registry must implement the default AccessControlPolicy and associate it with all Objects 3939 in the Registry. The following list summarizes the default role-based AccessControlPolicy:

- Anyone can publish content, but needs to be a Registered User
- Anyone can access the content without requiring authentication
- The ContentOwner has access to all methods for Registry Objects created by it.
- The RegistryAdministrator has access to all methods on all Registry Objects
- Unauthenticated clients can access all read-only (getXXX) methods

- At the time of content submission, the Registry must assign the default ContentOwner role to
- 3946 the Submitting Organization (SO) as authenticated by the credentials in the submission
- message. In the current version of this specification, the Submitting Organization will be theDN as identified by the certificate
- Clients that browse the Registry need not use certificates. The Registry must assign the default RegistryGuest role to such clients.

# 3951 Appendix A Web Service Architecture

## 3952 A.1 Registry Service Abstract Specification

- 3953 The normative definition of the Abstract Registry Service in WSDL is defined at the following
- 3954 location on the web:
- 3955 <u>http://www.oasis-open.org/committees/regrep/documents/2.0/services/Registry.wsdl</u>

## 3956 A.2 Registry Service SOAP Binding

- 3957 The normative definition of the concrete Registry Service binding to SOAP in WSDL is defined 3958 at the following location on the web:
- 3959 <u>http://www.oasis-open.org/committees/regrep/documents/2.0/services/SOAPBinding.wsdl</u>

3960

#### ebXML Registry Schema Definitions **Appendix B** 3961

#### **B.1 RIM Schema** 3962

- 3963 The normative XML Schema definition that maps [ebRIM] classes to XML can be found at the
- 3964 following location on the web:
- 3965 http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd

#### **B.2 Query Schema** 3966

- The normative XML Schema definition for the XML query syntax for the registry service 3967
- 3968 interface can be found at the following location on the web:
- 3969 http://www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd

#### **B.3 Registry Services Interface Schema** 3970

- 3971 The normative XML Schema definition that defines the XML requests and responses supported
- 3972 by the registry service interfaces in this document can be found at the following location on the web:
- 3973
- 3974 http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd

#### **B.4 Examples of Instance Documents** 3975

- 3976 A growing number of non-normative XML instance documents that conform to the normative
- 3977 Schema definitions described earlier may be found at the following location on the web:
- 3978 http://cvs.sourceforge.net/cgi-bin/viewcvs.cgi/ebxmlrr/ebxmlrr-spec/misc/samples/
- 3979

## 3980 Appendix C Interpretation of UML Diagrams

This section describes in *abstract terms* the conventions used to define ebXML business processdescription in UML.

### 3983 C.1 UML Class Diagram

A UML class diagram is used to describe the Service Interfaces required to implement an
 ebXML Registry Services and clients. The UML class diagram contains:

- 3986
- A collection of UML interfaces where each interface represents a Service Interface for a Registry service.
- 3989
   3990
   3990
   3990
   Action (as defined by [ebCPP]) within the Service Interface representing the UML interface.
- 3992
  3. Each method within a UML interface specifies one or more parameters, where the type of each method argument represents the ebXML message type that is exchanged as part of the Action corresponding to the method. Multiple arguments imply multiple payload documents within the body of the corresponding ebXML message.

### 3996 C.2 UML Sequence Diagram

A UML sequence diagram is used to specify the business protocol representing the interactions
 between the UML interfaces for a Registry specific ebXML business process. A UML sequence
 diagram provides the necessary information to determine the sequencing of messages, request to
 response association as well as request to error response association.

4000 response association as well as request to error response association.

4001 Each sequence diagram shows the sequence for a specific conversation protocol as method calls 4002 from the requestor to the responder. Method invocation may be synchronous or asynchronous

4003 based on the UML notation used on the arrow-head for the link. A half arrow-head represents

4004 asynchronous communication. A full arrow-head represents synchronous communication.

- 4005 Each method invocation may be followed by a response method invocation from the responder to
- 4006 the requestor to indicate the ResponseName for the previous Request. Possible error response is
- 4007 indicated by a conditional response method invocation from the responder to the requestor. See
- 4008 Figure 7 on page 27 for an example.

## 4009 Appendix D SQL Query

## 4010 D.1 SQL Query Syntax Specification

- 4011 This section specifies the rules that define the SQL Query syntax as a subset of SQL-92. The
- 4012 terms enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax 4013 conforms to the <query specification>, modulo the restrictions identified below:
- 4014 1. A <select list> may contain at most one <select sublist>.
- 4015
  4016
  2. In a <select list> must be is a single column whose data type is UUID, from the table in the <from clause>.
- 4017 3. A **<derived column>** may not have an **<as clause>**.
- 4018
   4. does not contain the optional <group by clause> and <having clause> (clauses)
- 4020 5. A can only consist of and **<correlation name>**.
- 40216. A does not have the optional AS between and4022<correlation name>.
- 4023 7. There can only be one in the **<from clause>**.
- 4024 8. Restricted use of sub-queries is allowed by the syntax as follows. The <in predicate> allows
  4025 for the right hand side of the <in predicate> to be limited to a restricted <query</li>
  4026 specification> as defined above.
- 4027 9. A **<search condition>** within the **<where clause>** may not include a **<query expression>**.
- 4028 10. Simple joins are allowed only if they are based on indexed columns within the relational4029 schema.
- 4030 11. The SQL query syntax allows for the use of <sql invoked routines> invocation from
  4031 [SQL/PSM] as the RHS of the <in predicate>.

## 4032 D.2 Non-Normative BNF for Query Syntax Grammar

 $\begin{array}{r} 4040 \\ 4041 \\ 4042 \\ 4043 \\ 4044 \\ 4045 \end{array}$ 

4ŏ4<u>6</u>

SQLAndExpr = SQLNotExpr ("AND" SQLNotExpr)\*

055

058 059

-060

406.

4063 4064

4065

40<u>66</u>

4067

4068 40<u>69</u>

4075

 $4076 \\ 4077 \\ 4078 \\ 4078 \\ 4079$ 

 $\begin{array}{r} 4079\\ 4080\\ 4080\\ 4081\\ 4082\\ 4083\\ 4083\\ 4084\\ 4085\\ 4086\end{array}$ 

 $4087 \\ 4088 \\ 4088 \\ 4089$ 

4090

4091

4092

4093 4094

4Ŏ9Ŏ 4100

4101 4102

 $\begin{array}{c} 4110\\ 4111\\ 4112\\ 4113\\ 4114\\ 4115\\ 4116\\ 4117\\ 4118\\ 4119\\ 4120\\ 4121\\ 4122\\ 4123\\ \end{array}$ 

 $41\overline{2}4$ 

```
SQLNotExpr = [ "NOT" ] SQLCompareExpr
SQLCompareExpr =
    (SQLColRef "IS") SQLIsClause
  | SQLSumExpr [ SQLCompareExprRight ]
SQLCompareExprRight =
   SQLLikeClause
    SQLInClause
  | SQLCompareOp SQLSumExpr
SQLCompareOp =
    " = "
    "<>"
   ">"
   ">="
    " < "
   "<="
SQLInClause = [ "NOT" ] "IN" "(" SQLLValueList ")"
SQLLValueList = SQLLValueElement ( "," SQLLValueElement )*
SQLLValueElement = "NULL" | SQLSelect
SQLIsClause = SQLColRef "IS" [ "NOT" ] "NULL"
SQLLikeClause = [ "NOT" ] "LIKE" SQLPattern
SQLPattern = STRING LITERAL
SQLLiteral =
   STRING LITERAL
    INTEGER LITERAL
  | FLOATING POINT LITERAL
SQLColRef = SQLLvalue
SQLLvalue = SQLLvalueTerm
SQLLvalueTerm = ID ( "." ID )*
SQLSumExpr = SQLProductExpr (( "+" | "-" ) SQLProductExpr )*
SQLProductExpr = SQLUnaryExpr (( "*" | "/" ) SQLUnaryExpr )*
SQLUnaryExpr = [ ( "+" | "-") ] SQLTerm
SQLTerm = "(" SQLOrExpr ")"
   SOLColRef
  SQLLiteral
INTEGER LITERAL = (["0"-"9"])+
FLOATING POINT LITERAL =
          (["0"-"9"]) + "." (["0"-"9"]) + (EXPONENT)?
          "." (["0"-"9"])+ (EXPONENT)?
          (["0"-"9"])+ EXPONENT
         (["0"-"9"])+ (EXPONENT)?
EXPONENT = ["e", "E"] (["+", "-"])? (["0"-"9"])+
STRING LITERAL: "'" (~["'"])* ( "''" (~["'"])* )* "'"
ID = ( <LETTER> ) + ( "_" | "$" | "#" | <DIGIT> | <LETTER> ) *
LETTER = ["A"-"Z", "a"-"Z"]
DIGIT = ["0"-"9"]
```

## 4125 **D.3 Relational Schema For SQL Queries**

- 4126 The normative Relational Schema definition for SQL queries can be found at the following
- 4127 location on the web:
- 4128 http://www.oasis-open.org/committees/regrep/documents/2.0/sql/database.sql
- 4129
- 4130 The stored procedures that must be supported by the SQL query feature are defined at the following
- 4131 location on the web:
- 4132 <u>http://www.oasis-open.org/committees/regrep/documents/2.0/sql/storedProcedures.sql</u>
- 4133

#### Non-normative Content Based Ad Hoc Queries **Appendix E** 4134

- 4135 The Registry SQL query capability supports the ability to search for content based not only on
- 4136 metadata that catalogs the content but also the data contained within the content itself. For
- example it is possible for a client to submit a query that searches for all Collaboration Party 4137
- Profiles that define a role named "seller" within a RoleName element in the CPP document itself. 4138
- 4139 Currently content-based query capability is restricted to XML content.

#### E.1 Automatic Classification of XML Content 4140

- 4141 Content-based queries are indirectly supported through the existing classification mechanism 4142 supported by the Registry.
- 4143 A submitting organization may define logical indexes on any XML schema or DTD when it is
- 4144 submitted. An instance of such a logical index defines a link between a specific attribute or
- 4145 element node in an XML document tree and a ClassificationNode in a classification scheme
- 4146 within the registry.
- 4147 The registry utilizes this index to automatically classify documents that are instances of the
- schema at the time the document instance is submitted. Such documents are classified according 4148
- 4149 to the data contained within the document itself.
- 4150 Such automatically classified content may subsequently be discovered by clients using the
- existing classification-based discovery mechanism of the Registry and the query facilities of the 4151
- 4152 QueryManager.

#### 4153 [Note] This approach is conceptually similar to the way databases support 4154 indexed retrieval. DBAs define indexes on tables in the schema. When 4155 data is added to the table, the data gets automatically indexed.

#### E.2 Index Definition 4156

- This section describes how the logical indexes are defined in the SubmittedObject element 4157
- 4158 defined in the Registry Schema. The complete Registry Schema is available via hyperlinks in
- 4159 Appendix B.
- 4160 A SubmittedObject element for a schema or DTD may define a collection of
- 4161 ClassificationIndexes in a ClassificationIndexList optional element. The ClassificationIndexList
- is ignored if the content being submitted is not of the SCHEMA objectType. 4162
- 4163 The ClassificationIndex element inherits the attributes of the base class RegistryObject in
- [ebRIM]. It then defines specialized attributes as follows: 4164
- 4165 1. classificationNode: This attribute references a specific ClassificationNode by its ID.
- 2. contentIdentifier: This attribute identifies a specific data element within the document 4166 4167 instances of the schema using an XPATH expression as defined by [XPT].

#### E.3 Example Of Index Definition 4168

4169 To define an index that automatically classifies a CPP based upon the roles defined within its RoleName elements, the following index must be defined on the CPP schema or DTD:

4172 4173 4174 4175 4176	<classificationindex classificationNode='id-for-role-classification-scheme' contentIdentifier='/Role//RoleName' /&gt;</classificationindex 

### 4177 E.4 Proposed XML Definition

4178

<b>H</b> 1/0	
4179	</th
4180	A ClassificationIndexList is specified on ExtrinsicObjects of objectType
4181	'Schema' to define an automatic Classification of instance objects of the
4182	schema using the specified classificationNode as parent and a
4183	ClassificationNode created or selected by the object content as selected
4184	by the contentIdentifier
4185	>
4186	ELEMENT ClassificationIndex EMPTY
4187	ATTLIST ClassificationIndex</th
4188	<pre>%ObjectAttributes;</pre>
4189	classificationNode IDREF #REQUIRED
4190	contentIdentifier CDATA #REQUIRED
4191	>
4192	
4193	ClassificationIndexList contains new ClassificationIndexes
4194	ELEMENT ClassificationIndexList (ClassificationIndex) *
4195	

### 4196 E.5 Example of Automatic Classification

Assume that a CPP is submitted that defines two roles as "seller" and "buyer." When the CPP is
submitted it will automatically be classified by two ClassificationNodes named "buyer" and
"seller" that are both children of the ClassificationNode (e.g. a node named Role) specified in the
classificationNode attribute of the ClassificationIndex. If either of the two ClassificationNodes
named "buyer" and "seller" did not previously exist, the LifeCycleManager would automatically
create these ClassificationNodes.

## 4203 Appendix F Security Implementation Guideline

4204 This section provides a suggested blueprint for how security processing may be implemented in

4205 the Registry. It is meant to be illustrative not prescriptive. Registries may choose to have

different implementations as long as they support the default security roles and authorizationrules described in this document.

### 4208 **F.1 Security Concerns**

4209 The security risks broadly stem from the following concerns. After a description of these

- 4210 concerns and potential solutions, we identify the concerns that we address in the current 4211 specificiation
- 4212 1. Is the content of the registry (data) trustworthy?
- a) How to make sure "what is in the registry" is "what is put there" by a submitting
  organization? This concern can be addressed by ensuring that the publisher is
  authenticated using digital signature (Source Integrity), message is not corrupted during
  transfer using digital signature (Data Integrity), and the data is not altered by
  unauthorized subjects based on access control policy (Authorization)
- b) How to protect data while in transmission?
- 4219Communication integrity has two ingredients Data Integrity (addressed in 1a) and Data4220Confidentiality that can be addressed by encrypting the data in transmission. How to4221protect against a replay attack?
- 4222 c) Is the content up to date? The versioning as well as any time stamp processing, when4223 done securely will ensure the "latest content" is guaranteed to be the latest content.
- d) How to ensure only bona fide responsible organizations add contents to registry?
   Ensuring Source Integrity (as in 1a).
- 4226 e) How to ensure that bona fide publishers add contents to registry only at authorized4227 locations? (System Integrity)
- f) What if the publishers deny modifying certain content after-the-fact? To prevent this
   (Nonrepudiation) audit trails may be kept which contain signed message digests.
- 4230 g) What if the reader denies getting information from the registry?
- 4231 2. How to provide selective access to registry content? The broad answer is, by using an access
  4232 control policy applies to (a), (b), and (c) directly.
- 4233 a) How does a submitting organization restrict access to the content to only specific registry readers?
- b) How can a submitting organization allow some "partners" (fellow publishers) to modify content?
- 4237 c) How to provide selective access to partners the registry usage data?
- d) How to prevent accidental access to data by unauthorized users? Especially with hw/sw
  failure of the registry security components? The solution to this problem is by having
  System Integrity.
- 4241 e) Data confidentiality of RegistryObject

- 4242 3. How do we make "who can see what" policy itself visible to limited parties, even excluding
  4243 the administrator (self & confidential maintenance of access control policy). By making sure
  4244 there is an access control policy for accessing the policies themselves.
- 4245
  4246
  4246
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247
  4247</l
- 4248 a) How to transfer credentials (authorization/authentication) to federated registries?
- b) How do aggregators get credentials (authorization/authentication) transferred to them?
- 4250 c) How to store credentials through a session?

### 4251 F.2 Authentication

- 4252 1. As soon as a message is received, the first work is the authentication. A principal object is4253 created.
- 4254
  4255
  4256
  2. If the message is signed, it is verified (including the validity of the certificate) and the DN of the certificate becomes the identity of the principal. Then the Registry is searched for the principal and if found, the roles and groups are filled in.
- 4257 3. If the message is not signed, an empty principal is created with the role RegistryGuest. This4258 step is for symmetry and to decouple the rest of the processing.
- 4259 4. Then the message is processed for the command and the objects it will act on.

### 4260 **F.3 Authorization**

- 4261 For every object, the access controller will iterate through all the AccessControlPolicy objects
- 4262 with the object and see if there is a chain through the permission objects to verify that the
- 4263 requested method is permitted for the Principal. If any of the permission objects which the object
- 4264 is associated with has a common role, or identity, or group with the principal, the action is
- 4265 permitted.

## 4266 **F.4 Registry Bootstrap**

- When a Registry is newly created, a default Principal object should be created with the identity
  of the Registry Admin's certificate DN with a role RegistryAdmin. This way, any message
  signed by the Registry Admin will get all the privileges.
- 4270 When a Registry is newly created, a singleton instance of AccessControlPolicy is created as the
- 4271 default AccessControlPolicy. This includes the creation of the necessary Permission instances as
- 4272 well as the Privilges and Privilege attributes.

## 4273 **F.5 Content Submission – Client Responsibility**

4274 The Registry client must sign the contents before submission – otherwise the content will be 4275 rejected.

### 4276 **F.6 Content Submission – Registry Responsibility**

- 4277 1. As with any other request, the client will first be authenticated. In this case, the Principal object will get the DN from the certificate.
- 4279 2. As per the request in the message, the RegistryEntry will be created.
- 4280 3. The RegistryEntry is assigned the singleton default AccessControlPolicy.
- 42814. If a principal with the identity of the SO is not available, an identity object with the SO's DN4282 is created.
- 4283 5. A principal with this identity is created.

## 4284 **F.7 Content Delete/Deprecate – Client Responsibility**

4285 The Registry client must sign the header before submission, for authentication purposes; 4286 otherwise, the request will be rejected

# 4287 **F.8 Content Delete/Deprecate – Registry Responsibility**

- 4288
  4289
  4289
  4289
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280
  4280</l
- 4291 2. As per the request in the message (delete or deprecate), the appropriate method in the4292 RegistryObject class will be accessed.
- 4293 3. The access controller performs the authorization by iterating through the Permission objects
  4294 associated with this object via the singleton default AccessControlPolicy.
- 42954. If authorization succeeds then the action will be permitted. Otherwise an error response is4296 sent back with a suitable AuthorizationException error message.

### 4297 F.9 Using ds:KeyInfo Field

- 4298 Two typical usage scenarios for ds:KeyInfo are described below.
- 4299 Scenario 1
- 4300 1. Registry Client (RC) signs the payload and the SOAP envelope using its private key.
- 4301 2. The certificate of RC is passed to the Registry in KeyInfo field of the header signature.
- 4302 3. The certificate of RC is passed to the Registry in KeyInfo field of the payload signature.
- 4303 4. Registration Authority retrieves the certificate from the KeyInfo field in the header signature
- 4304 5. Registration Authority validates the header signature using the public key from the certificate.
- 4306
  6. Registration Authority validates the payload signature by repeating steps 4 and 5 using the
  4307
  4308
  4308
  4308
  4309
  4309
  4309
- 4310 **Scenario 2**

4311 4312	1.	RC1 signs the payload and SOAP envelope using its private key and publishes to the Registry.
4313	2.	The certificate of RC1 is passed to the Registry in the KeyInfo field of the header signature.
4314 4315 4316	3.	The certificate of RC1 is passed to the Registry in the KeyInfo field of the payload signature. This step is required in addition to step 2 because when RC2 retrieves content, it should see RC1's signature with the payload.
4317	4.	RC2 retrieves content from the Registry.
4318 4319	5.	Registration Authority signs the SOAP envelope using its private key. Registration Authority sends RC1's content and the RC1's signature (signed by RC1).
4320 4321 4322	6.	Registration Authority need not send its certificate in the KeyInfo field sinceRC2 is assumed to have obtained the Registration Authority's certificate out of band and installed it in its local key store.

- 4323
   7. RC2 obtains Registration Authority's certificate out of its local key store and verifies the
   4324
   Registration Authority's signature.
- 4325 8. RC2 obtains RC1's certificate from the KeyInfo field of the payload signature and validates
  4326 the signature on the payload.

#### Appendix G Native Language Support (NLS) 4327

#### G.1 Definitions 4328

4329 Although this section discusses only character set and language, the following terms have to be 4330 defined clearly.

#### 4331 G.1.1 Coded Character Set (CCS):

- 4332 CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of
- CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on. 4333
- 4334 G.1.2 Character Encoding Scheme (CES):
- CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are 4335 4336 ISO-2022, UTF-8.

#### 4337 G.1.3 Character Set (charset):

- 4338 • charset is a set of rules for mapping from a sequence of octets to a sequence of characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR. 4339
- A list of registered character sets can be found at [IANA]. 4340

#### G.2 NLS And Request / Response Messages 4341

4342 For the accurate processing of data in both registry client and registry services, it is essential to know which character set is used. Although the body part of the transaction may contain the 4343 4344 charset in xml encoding declaration, registry client and registry services shall specify charset 4345 parameter in MIME header when they use text/xml. Because as defined in [RFC 3023], if a text/xml entity is received with the charset parameter omitted, MIME processors and XML 4346

processors MUST use the default charset value of "us-ascii". For example: 4347 4348 4349 4350

Content-Type: text/xml; charset=ISO-2022-JP

- 4351 Also, when an application/xml entity is used, the charset parameter is optional, and registry
- 4352 client and registry services must follow the requirements in Section 4.3.3 of [REC-XML] which 4353 directly address this contingency.
- 4354 If another Content-Type is chosen to be used, usage of charset must follow [RFC 3023].

#### G.3 NLS And Storing of RegistryObject 4355

- 4356 This section provides NLS guidelines on how a registry should store RegistryObject instances.
- 4357 A single instance of a concrete sub-class of RegistryObject is capable of supporting multiple
- 4358 locales. Thus there is no language or character set associated with a specific RegistryObject 4359 instance.
- 4360 A single instance of a concrete sub-class of RegistryObject supports multiple locales as follows. Each attribute of the RegistryObject that is I18N capable (e.g. name and description attributes in 4361

- 4362 RegistryObject class) as defined by [ebRIM], may have multiple locale specific values expressed
- 4363 as LocalizedString sub-elements within the XML element representing the I18N capable
- 4364 attribute. Each LocalizedString sub-element defines the value of the I18N capable attribute in a
- 4365 specific locale. Each LocalizedString element has a charset and lang attribute as well as a value
- 4366 attribute of type string.

### 4367 G.3.1 Character Set of LocalizedString

4368 The character set used by a locale specific String (LocalizedString) is defined by the charset 4369 attribute. It is highly recommended to use UTF-8 or UTF-16 for maximuminter-operability.

### 4370 **G.3.2 Language Information of** *LocalizedString*

4371 The language may be specified in xml:lang attribute (Section 2.12 [REC-XML]).

### 4372 G.4 NLS And Storing of Repository Items

- 4373 This section provides NLS guidelines on how a registry should store repository items.
- 4374 While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is
- 4375 always associated with a single repository item. The repository item may be in a single locale or
- 4376 may be in multiple locales. This specification does not specify the repository item.

### 4377 G.4.1 Character Set of Repository Items

- 4378 The MIME Content-Type mime header for the mime multi-part containing the repository
- 4379 item MAY contain a "charset" attribute that specifies the character set used by the repository4380 item. For example:
- 4381
- 4382 Content-Type: text/xml; charset="UTF-8"
- 4383

4384 It is highly recommended to use UTF-16 or UTF-8 for maximum inter-operability. The charset4385 of a repository item must be preserved as it is originally specified in the transaction.

### 4386 **G.4.2 Language information of repository item**

- The Content-language mime header for the mime bodypart containing the repository item may
  specify the language for a locale specific repository item. The value of the Content-language
  mime header property must conform to [RFC 1766].
- 4390 This document currently specifies only the method of sending the information of character set
- 4391 and language, and how it is stored in a registry. However, the language information may be used
- 4392 as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a
- 4393 language negotiation procedure, like registry client is asking a favorite language for messages
- 4394 from registry services, could be another functionality for the future revision of this document.

# 4395 Appendix H Registry Profile

4396 Every registry must support exactly one Registry Profile. The Registry Profile is an XML

- 4397 document that describes the capabilities of the registry. The profile document must conform to
- 4398 the RegistryProfile element as described in the Registry Services Interface schema defined in
- 4399 Appendix B. The registry must make the RegistryProfile accessible over HTTP protocol via a
- 4400 URL. The URL must conform to the pattern:
- 4401 <u>http://<base url>/registryProfile</u>
- 4402

# 4403 **10 References**

4404	[Bra97] Keywords for use in RFCs to Indicate Requirement Levels.
4405	[ebRIM] ebXML Registry Information Model version 2.0
4406	http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebRIM.pdf
4407	[ebRIM Schema] ebXML Registry Information Model Schema
4408	http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd
4409	[ebBPSS] ebXML Business Process Specification Schema
4410	http://www.ebxml.org/specs
4411	[ebCPP] ebXML Collaboration-Protocol Profile and Agreement Specification
4412	http://www.ebxml.org/specs/
4413	[ebMS] ebXML Messaging Service Specification, Version 1.0
4414	http://www.ebxml.org/specs/
4415	[XPT] XML Path Language (XPath) Version 1.0
4416	http://www.w3.org/TR/xpath
4417	[SQL] Structured Query Language (FIPS PUB 127-2)
4418	http://www.itl.nist.gov/fipspubs/fip127-2.htm
4419	[SQL/PSM] Database Language SQL — Part 4: Persistent Stored Modules
4420	(SQL/PSM) [ISO/IEC 9075-4:1996]
4421	[IANA] IANA (Internet Assigned Numbers Authority).
4422	Official Names for Character Sets, ed. Keld Simonsen et al.
4423	<u>ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets</u>
4424	[RFC 1766] IETF (Internet Engineering Task Force). RFC 1766:
4425	Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
4426	http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html
4427	[RFC 2119] IETF (Internet Engineering Task Force). RFC 2119
4428	[RFC 2130] IETF (Internet Engineering Task Force). RFC 2130
4429	[RFC 2277] IETF (Internet Engineering Task Force). RFC 2277:
4430 4431	IETF policy on character sets and languages, ed. H. Alvestrand. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html
4431	[RFC 2278] IETF (Internet Engineering Task Force). RFC 2278:
4432	
4433	IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998. http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html
4435	[RFC 2828] IETF (Internet Engineering Task Force). RFC 2828:
4436	Internet Security Glossary, ed. R. Shirey. May 2000.
4437	http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html
4438	[RFC 3023] ETF (Internet Engineering Task Force). RFC 3023:
4439	XML Media Types, ed. M. Murata. 2001.
4440	ftp://ftp.isi.edu/in-notes/rfc3023.txt
4441	[REC-XML] W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
4442	http://www.w3.org/TR/REC-xml
4443	[UUID] DCE 128 bit Universal Unique Identifier
4443	http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
4445	http://www.opengroup.org/publications/catalog/c706.htmttp://www.w3.org/TR/REC-xml

4446 [WSDL] W3C Note. Web Services Description Language (WSDL) 1.1

- 4447 <u>http://www.w3.org/TR/wsdl</u>
- 4448 [SOAP11] W3C Note. Simple Object Access Protocol, May 2000, http://www.w3.org/TR/SOAP
- 4450 [SOAPAt] W3C Note: SOAP with Attachments, Dec 2000, 4451 <u>http://www.w3.org/TR/SOAP-attachments</u>
- 4452 [XMLDSIG] XML-Signature Syntax and Processing,
- 4453 <u>http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/</u>

## 4454 **11 Disclaimer**

4455 The views and specification expressed in this document are those of the authors and are not

necessarily those of their employers. The authors and their employers specifically disclaim
 responsibility for any problems arising from correct or incorrect implementation or use of this

4458 design.

# 4459**12 Contact Information**

4460	Team Leader	
4461	Name:	Lisa J. Carnahan
4462	Company:	National Institute of Standards and Technology
4463	Street:	100 Bureau Drive, Stop 8970
4464	City, State, Postal Code:	Gaithersburg, Md. 20899
4465	Country:	USA
4466	Phone:	301-975-3362
4467	Email:	lisa.carnahan@nist.gov
4468		
4469	Editor	
4470	Name:	Anne A. Fischer
4471	Company:	Drummond Group, Inc.
4472	Street:	4700 Bryant Irvin Ct., Suite 303
4473	City, State, Postal Code:	Fort Worth, Texas 76107-7645
4474	Country:	USA
4475	Phone:	817-371-2367
4476	Email:	anne@drummondgroup.com
4477		
4478	Technical Editor	
4479	Name:	Farrukh S. Najmi
4480	Company:	Sun Microsystems
4481	Street:	1 Network Dr., MS BUR02-302
4482	City, State, Postal Code:	Burlington, MA 01803-0902
4483	Country:	USA
4484	Phone:	781-442-0703
4485	Email:	najmi@east.sun.com
4486		

# 4487 **13 Copyright Statement**

4488	Copyright (C) The Organization for the Advancement of Structured Information
4489	Standards [OASIS], 2001. All Rights Reserved.
4490	This document and translations of it may be copied and furnished to others, and derivative works
4491	that comment on or otherwise explain it or assist in its implementation may be prepared, copied,
4492	published and distributed, in whole or in part, without restriction of any kind, provided that the
4493	above copyright notice and this paragraph are included on all such copies and derivative works.
4494	However, this document itself may not be modified in any way, such as by removing the
4495	copyright notice or references to OASIS, except as needed for the purpose of developing OASIS
4496	specifications, in which case the procedures for copyrights defined in the OASIS Intellectual
4497	Property Rights document must be followed, or as required to translate it into languages other
4498	than English.
4499	The limited permissions granted above are perpetual and will not be revoked by OASIS or its
4500	successors or assigns.
4501	This document and the information contained herein is provided on an "AS IS" basis and OASIS
4502	DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT
4503	LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN
4504	WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
4505	MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
4506	OASIS takes no position regarding the validity or scope of any intellectual property or other
4507	rights that might be claimed to pertain to the implementation or use of the technology described
4508	in this document or the extent to which any license under such rights might or might not be
4509	available; neither does it represent that it has made any effort to identify any such rights.
4510	Information on OASIS's procedures with respect to rights in OASIS specifications can be found
4511	at the OASIS website. Copies of claims of rights made available for publication and any
4512	assurances of licenses to be made available, or the result of an attempt made to obtain a general
4513	license or permission for the use of such proprietary rights by implementors or users of this
4514	specification, can be obtained from the OASIS Executive Director.
4515	OASIS invites any interested party to bring to its attention any copyrights, patents or patent

- 4515 OASIS invites any interested party to bring to its attention any copyrights, patents or patent 4516 applications, or other proprietary rights which may cover technology that may be required to
- 4517 implement this specification. Please address the information to the OASIS Executive Director.