

ebXML Registry Services and Protocols Version 3.0

OASIS Standard, 2 May, 2005

Document identifier:
regrep-rs-3.0-os

Location:
<http://docs.oasis-open.org/regrep-rs/v3.0/>

Editors:

Name	Affiliation
Sally Fuger	Individual
Farrukh Najmi	Sun Microsystems
Nikola Stojanovic	RosettaNet

Contributors:

Name	Affiliation
Diego Ballve	Individual
Ivan Bedini	France Telecom
Kathryn Breininger	The Boeing Company
Joseph Chiusano	Booz Allen Hamilton
Peter Kacandes	Adobe Systems
Paul Macias	LMI Government Consulting
Carl Mattocks	CHECKMi
Matthew MacKenzie	Adobe Systems
Monica Martin	Sun Microsystems
Richard Martell	Galdos Systems Inc
Duane Nickull	Adobe Systems
Goran Zugic	ebXMLsoft Inc.

12

13 **Abstract:**

14 This document defines the services and protocols for an ebXML Registry

15 A separate document, ebXML Registry: Information Model [ebRIM], defines the types of
16 metadata and content that can be stored in an ebXML Registry.

17 **Status:**

18 This document is an OASIS ebXML Registry Technical Committee Approved Draft
19 Specification.

20 Committee members should send comments on this specification to the [regrep@lists.oasis-](mailto:regrep@lists.oasis-open.org)
21 [open.org](mailto:regrep@lists.oasis-open.org) list. Others should subscribe to and send comments to the [comment@lists.oasis-open.org](mailto:regrep-
22 <a href=) list. To subscribe, send an email message to [request@lists.oasis-open.org](mailto:regrep-comment-
23 <a href=) with the word "subscribe" as the body of the message.

24 For information on whether any patents have been disclosed that may be essential to
25 implementing this specification, and any offers of patent licensing terms, please refer to the
26 Intellectual Property Rights section of the OASIS ebXML Registry TC web page
27 (<http://www.oasis-open.org/committees/regrep/>).

28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72

Table of Contents

1 Introduction.....	13
1.1 Audience.....	13
1.2 Terminology.....	13
1.3 Notational Conventions.....	13
1.3.1 UML Diagrams.....	13
1.3.2 Identifier Placeholders.....	13
1.3.3 Constants.....	13
1.3.4 Bold Text.....	14
1.3.5 Example Values.....	14
1.4 XML Schema Conventions.....	14
1.4.1 Schemas Defined by ebXML Registry.....	14
1.4.2 Schemas Used By ebXML Registry.....	15
1.5 Registry Actors.....	16
1.6 Registry Use Cases.....	16
1.7 Registry Architecture.....	16
1.7.1 Registry Clients.....	17
1.7.1.1 Client API.....	17
1.7.2 Registry Service Interfaces.....	17
1.7.3 Service Interface: Protocol Bindings.....	17
1.7.4 Authentication and Authorization.....	18
1.7.5 Metadata Registry and Content Repository.....	18
2 Registry Protocols.....	19
2.1 Requests and Responses.....	19
2.1.1 RegistryRequestType.....	19
2.1.1.1 Syntax:.....	19
2.1.1.2 Parameters:.....	20
2.1.1.3 Returns:.....	20
2.1.1.4 Exceptions:.....	20
2.1.2 RegistryRequest.....	20
2.1.3 RegistryResponseType.....	20
2.1.3.1 Syntax:.....	21
2.1.3.2 Parameters:.....	21
2.1.4 RegistryResponse.....	21
2.1.5 RegistryErrorList.....	21
2.1.5.1 Syntax:.....	22
2.1.5.2 Parameters:.....	22
2.1.6 RegistryError.....	22
2.1.6.1 Syntax:.....	22
2.1.6.2 Parameters:.....	22
3 SOAP Binding.....	24
3.1 ebXML Registry Service Interfaces: Abstract Definition.....	24
3.2 ebXML Registry Service Interfaces SOAP Binding.....	25
3.3 ebXML Registry Service Interfaces SOAP Service Template.....	25
3.4 Mapping of Exception to SOAP Fault.....	26

73	4 HTTP Binding.....	27
74	4.1 HTTP Interface URL Pattern.....	27
75	4.2 RPC Encoding URL.....	27
76	4.2.1 Standard URL Parameters.....	27
77	4.2.2 QueryManager Binding.....	28
78	4.2.2.1 Sample getRegistryObject Request.....	28
79	4.2.2.2 Sample getRegistryObject Response.....	28
80	4.2.2.3 Sample getRepositoryItem Request.....	29
81	4.2.2.4 Sample getRepositoryItem Response.....	29
82	4.2.3 LifeCycleManager HTTP Interface.....	29
83	4.3 Submitter Defined URL.....	29
84	4.3.1 Submitter defined URL Syntax.....	30
85	4.3.2 Assigning URL to a RegistryObject	30
86	4.3.3 Assigning URL to a Repository Item	31
87	4.4 File Path Based URL.....	31
88	4.4.1 File Folder Metaphor.....	31
89	4.4.2 File Path of a RegistryObject.....	31
90	4.4.2.1 File Path Example.....	32
91	4.4.3 Matching URL To Objects.....	32
92	4.4.4 URL Matches a Single Object.....	32
93	4.4.5 URL Matches Multiple Object.....	33
94	4.4.6 Directory Listing.....	33
95	4.4.7 Access Control In RegistryPackage Hierarchy.....	34
96	4.5 URL Resolution Algorithm.....	34
97	4.6 Security Consideration.....	34
98	4.7 Exception Handling.....	35
99	5 Lifecycle Management Protocols.....	36
100	5.1 Submit Objects Protocol.....	36
101	5.1.1 SubmitObjectsRequest.....	36
102	5.1.1.1 Syntax:.....	36
103	5.1.1.2 Parameters:.....	37
104	5.1.1.3 Returns:.....	37
105	5.1.1.4 Exceptions:.....	37
106	5.1.2 Unique ID Generation.....	37
107	5.1.3 ID Attribute And Object References.....	37
108	5.1.4 Audit Trail.....	38
109	5.1.5 Sample SubmitObjectsRequest.....	38
110	5.2 The Update Objects Protocol.....	38
111	5.2.1 UpdateObjectsRequest.....	39
112	5.2.1.1 Syntax:.....	39
113	5.2.1.2 Parameters:.....	39
114	5.2.1.3 Returns:.....	39
115	5.2.1.4 Exceptions:.....	39
116	5.2.2 Audit Trail.....	40
117	5.3 The Approve Objects Protocol.....	40
118	5.3.1 ApproveObjectsRequest.....	40

119	5.3.1.1 Syntax:.....	40
120	5.3.1.2 Parameters:.....	40
121	5.3.1.3 Returns:.....	41
122	5.3.1.4 Exceptions:.....	41
123	5.3.2 Audit Trail.....	41
124	5.4 The Deprecate Objects Protocol.....	41
125	5.4.1 DeprecateObjectsRequest.....	41
126	5.4.1.1 Syntax:.....	42
127	5.4.1.2 Parameters:.....	42
128	5.4.1.3 Returns:.....	42
129	5.4.1.4 Exceptions:.....	42
130	5.4.2 Audit Trail.....	42
131	5.5 The Undeprecate Objects Protocol.....	42
132	5.5.1 UndeprecateObjectsRequest.....	43
133	5.5.1.1 Syntax:.....	43
134	5.5.1.2 Parameters:.....	43
135	5.5.1.3 Returns:.....	44
136	5.5.1.4 Exceptions:.....	44
137	5.5.2 Audit Trail.....	44
138	5.6 The Remove Objects Protocol.....	44
139	5.6.1 RemoveObjectsRequest.....	44
140	5.6.1.1 Syntax:.....	44
141	5.6.1.2 Parameters:.....	45
142	5.6.1.3 Returns:.....	45
143	5.6.1.4 Exceptions:.....	45
144	5.7 Registry Managed Version Control.....	46
145	5.7.1 Version Controlled Resources.....	46
146	5.7.2 Versioning and Object Identification.....	46
147	5.7.3 Logical ID.....	46
148	5.7.4 Version Identification.....	46
149	5.7.4.1 Version Identification for a RegistryObject.....	47
150	5.7.4.2 Version Identification for a RepositoryItem.....	47
151	5.7.5 Versioning of ExtrinsicObject and Repository Items.....	47
152	5.7.5.1 ExtrinsicObject and Shared RepositoryItem.....	48
153	5.7.6 Versioning and Composed Objects.....	48
154	5.7.7 Versioning and References.....	48
155	5.7.8 Versioning and Audit Trail.....	48
156	5.7.9 Inter-versions Association.....	49
157	5.7.10 Client Initiated Version Removal.....	49
158	5.7.11 Registry Initiated Version Removal.....	49
159	5.7.12 Locking and Concurrent Modifications.....	49
160	5.7.13 Version Creation.....	49
161	5.7.14 Versioning Override.....	50
162	6 Query Management Protocols.....	51
163	6.1 Ad Hoc Query Protocol.....	51
164	6.1.1 AdhocQueryRequest.....	51
165	6.1.1.1 Syntax:.....	51

166	6.1.1.2 Parameters:.....	52
167	6.1.1.3 Returns:.....	52
168	6.1.1.4 Exceptions:.....	52
169	6.1.2 AdhocQueryResponse.....	52
170	6.1.2.1 Syntax:.....	52
171	6.1.2.2 Parameters:.....	53
172	6.1.3 AdhocQuery.....	53
173	6.1.3.1 Syntax:.....	53
174	6.1.3.2 Parameters:.....	53
175	6.1.4 ReponseOption.....	53
176	6.1.4.1 Syntax:.....	54
177	6.1.4.2 Parameters:.....	54
178	6.2 Iterative Query Support.....	54
179	6.2.1 Query Iteration Example.....	55
180	6.3 Stored Query Support.....	55
181	6.3.1 Submitting a Stored Query.....	55
182	6.3.1.1 Declaring Query Parameters.....	55
183	6.3.1.2 Canonical Context Parameters.....	56
184	6.3.2 Invoking a Stored Query.....	56
185	6.3.2.1 Specifying Query Invocation Parameters.....	57
186	6.3.3 Response to Stored Query Invocation.....	57
187	6.3.4 Access Control on a Stored Query.....	57
188	6.3.5 Canonical Query: Get Client's User Object.....	58
189	6.4 SQL Query Syntax.....	58
190	6.4.1 Relational Schema for SQL Queries.....	58
191	6.4.2 SQL Query Results.....	58
192	6.5 Filter Query Syntax.....	59
193	6.5.1 Filter Query Structure.....	59
194	6.5.2 Query Elements.....	59
195	6.5.3 Filter Elements.....	60
196	6.5.3.1 FilterType.....	61
197	6.5.3.2 SimpleFilterType.....	61
198	6.5.3.3 BooleanFilter.....	62
199	6.5.3.4 FloatFilter.....	62
200	6.5.3.5 IntegerFilter.....	63
201	6.5.3.6 DateTimeFilter.....	63
202	6.5.3.7 StringFilter.....	63
203	6.5.3.8 CompoundFilter.....	63
204	6.5.4 Nested Query Elements.....	64
205	6.5.5 Branch Elements.....	64
206	6.6 Query Examples.....	65
207	6.6.1 Name and Description Queries.....	65
208	6.6.2 Classification Queries.....	66
209	6.6.2.1 Retrieving ClassificationSchemes.....	66
210	6.6.2.2 Retrieving Children of Specified ClassificationNode.....	66
211	6.6.2.3 Retrieving Objects Classified By a ClassificationNode.....	66
212	6.6.2.4 Retrieving Classifications that Classify an Object.....	67

213	6.6.3 Association Queries.....	67
214	6.6.3.1 Retrieving All Associations With Specified Object As Source.....	67
215	6.6.3.2 Retrieving All Associations With Specified Object As Target.....	67
216	6.6.3.3 Retrieving Associated Objects Based On Association Type.....	68
217	6.6.3.4 Complex Association Query.....	68
218	6.6.4 Package Queries.....	68
219	6.6.5 ExternalLink Queries.....	69
220	6.6.6 Audit Trail Queries.....	70
221	7 Event Notification Protocols.....	71
222	7.1 Use Cases.....	71
223	7.1.1 CPP Has Changed.....	71
224	7.1.2 New Service is Offered.....	71
225	7.1.3 Monitor Download of Content.....	71
226	7.1.4 Monitor Price Changes.....	71
227	7.1.5 Keep Replicas Consistent With Source Object.....	71
228	7.2 Registry Events.....	71
229	7.3 Subscribing to Events.....	72
230	7.3.1 Event Selection.....	72
231	7.3.2 Notification Action.....	72
232	7.3.3 Subscription Authorization.....	73
233	7.3.4 Subscription Quotas.....	73
234	7.3.5 Subscription Expiration.....	73
235	7.3.6 Subscription Rejection.....	73
236	7.4 Unsubscribing from Events.....	73
237	7.5 Notification of Events.....	73
238	7.6 Retrieval of Events.....	74
239	7.7 Pruning of Events.....	74
240	8 Content Management Services.....	75
241	8.1 Content Validation.....	75
242	8.1.1 Content Validation: Use Cases.....	75
243	8.1.1.1 Validation of HL7 Conformance Profiles.....	75
244	8.1.1.2 Validation of Business Processes.....	75
245	8.1.1.3 Validation of UBL Business Documents.....	75
246	8.2 Content Cataloging.....	76
247	8.2.1 Content-based Discovery: Use Cases.....	76
248	8.2.1.1 Find All CPPs Where Role is "Buyer".....	76
249	8.2.1.2 Find All XML Schema's That Use Specified Namespace.....	76
250	8.2.1.3 Find All WSDL Descriptions with a SOAP Binding.....	76
251	8.3 Abstract Content Management Service.....	76
252	8.3.1 Inline Invocation Model.....	77
253	8.3.2 Decoupled Invocation Model.....	78
254	8.4 Content Management Service Protocol.....	79
255	8.4.1 ContentManagementServiceRequestType.....	79
256	8.4.1.1 Syntax:.....	79
257	8.4.1.2 Parameters:.....	80
258	8.4.1.3 Returns:.....	80

259	8.4.1.4 Exceptions:.....	80
260	8.4.2 ContentManagementServiceResponseType.....	80
261	8.4.2.1 Syntax:.....	80
262	8.4.2.2 Parameters:.....	81
263	8.5 Publishing / Configuration of a Content Management Service.....	81
264	8.5.1 Multiple Content Management Services and Invocation Control Files.....	82
265	8.6 Invocation of a Content Management Service.....	83
266	8.6.1 Resolution Algorithm For Service and Invocation Control File.....	83
267	8.6.2 Audit Trail and Cataloged Content.....	83
268	8.6.3 Referential Integrity.....	83
269	8.6.4 Error Handling.....	83
270	8.7 Validate Content Protocol.....	84
271	8.7.1 ValidateContentRequest.....	84
272	8.7.1.1 Syntax:.....	84
273	8.7.1.2 Parameters:.....	85
274	8.7.1.3 Returns:.....	85
275	8.7.1.4 Exceptions:.....	85
276	8.7.2 ValidateContentResponse.....	85
277	8.7.2.1 Syntax:.....	85
278	8.7.2.2 Parameters:.....	85
279	8.8 Catalog Content Protocol.....	86
280	8.8.1 CatalogContentRequest.....	86
281	8.8.1.1 Syntax:.....	86
282	8.8.1.2 Parameters:.....	87
283	8.8.1.3 Returns:.....	87
284	8.8.1.4 Exceptions:.....	87
285	8.8.2 CatalogContentResponse.....	87
286	8.8.2.1 Syntax:.....	87
287	8.8.2.2 Parameters:.....	88
288	8.9 Illustrative Example: Canonical XML Cataloging Service.....	88
289	8.10 Canonical XML Content Cataloging Service.....	89
290	8.10.1 Publishing of Canonical XML Content Cataloging Service.....	90
291	9 Cooperating Registries Support.....	91
292	9.1 Cooperating Registries Use Cases.....	91
293	9.1.1 Inter-registry Object References.....	91
294	9.1.2 Federated Queries.....	91
295	9.1.3 Local Caching of Data from Another Registry.....	91
296	9.1.4 Object Relocation.....	91
297	9.2 Registry Federations.....	92
298	9.2.1 Federation Metadata.....	92
299	9.2.2 Local Vs. Federated Queries.....	93
300	9.2.2.1 Local Queries.....	93
301	9.2.2.2 Federated Queries.....	93
302	9.2.2.3 Membership in Multiple Federations.....	94
303	9.2.3 Federated Lifecycle Management Operations.....	94
304	9.2.4 Federations and Local Caching of Remote Data.....	94
305	9.2.5 Caching of Federation Metadata.....	94

306	9.2.6 Time Synchronization Between Registry Peers.....	94
307	9.2.7 Federations and Security.....	94
308	9.2.8 Federation Lifecycle Management Protocols	95
309	9.2.8.1 Joining a Federation.....	95
310	9.2.8.2 Creating a Federation.....	95
311	9.2.8.3 Leaving a Federation.....	95
312	9.2.8.4 Dissolving a Federation.....	95
313	9.3 Object Replication.....	96
314	9.3.1 Use Cases for Object Replication.....	96
315	9.3.2 Queries And Replicas.....	96
316	9.3.3 Lifecycle Operations And Replicas.....	97
317	9.3.4 Object Replication and Federated Registries.....	97
318	9.3.5 Creating a Local Replica.....	97
319	9.3.6 Transactional Replication.....	97
320	9.3.7 Keeping Replicas Current.....	97
321	9.3.8 Lifecycle Management of Local Replicas.....	98
322	9.3.9 Tracking Location of a Replica.....	98
323	9.3.10 Remote Object References to a Replica.....	98
324	9.3.11 Removing a Local Replica.....	98
325	9.4 Object Relocation Protocol.....	98
326	9.4.1 RelocateObjectsRequest.....	101
327	9.4.1.1 Parameters:.....	101
328	9.4.1.2 Returns:.....	101
329	9.4.1.3 Exceptions:.....	101
330	9.4.2 AcceptObjectsRequest.....	101
331	9.4.2.1 Parameters:.....	102
332	9.4.2.2 Returns:.....	102
333	9.4.2.3 Exceptions:.....	102
334	9.4.3 Object Relocation and Remote ObjectRefs.....	102
335	9.4.4 Notification of Object Relocation To ownerAtDestination.....	103
336	9.4.5 Notification of Object Commit To sourceRegistry.....	103
337	9.4.6 Object Ownership and Owner Reassignment.....	103
338	9.4.7 Object Relocation and Timeouts.....	103
339	10 Registry Security.....	104
340	10.1 Security Use Cases.....	104
341	10.1.1 Identity Management.....	104
342	10.1.2 Message Security.....	104
343	10.1.3 Repository Item Security.....	104
344	10.1.4 Authentication.....	104
345	10.1.5 Authorization and Access Control.....	104
346	10.1.6 Audit Trail.....	104
347	10.2 Identity Management.....	105
348	10.3 Message Security.....	105
349	10.3.1 Transport Layer Security.....	105
350	10.3.2 SOAP Message Security.....	105
351	10.3.2.1 Request Message Signature.....	105

352	10.3.2.2 Response Message Signature.....	105
353	10.3.2.3 KeyInfo Requirements.....	106
354	10.3.2.4 Message Signature Validation.....	106
355	10.3.2.5 Message Signature Example.....	106
356	10.3.2.6 Message With RepositoryItem: Signature Example.....	107
357	10.3.2.7 SOAP Message Security and HTTP/S.....	109
358	10.3.3 Message Confidentiality.....	109
359	10.3.4 Key Distribution Requirements.....	109
360	10.4 Authentication.....	109
361	10.4.1 Registry as Authentication Authority.....	110
362	10.4.2 External Authentication Authority.....	110
363	10.4.3 Authenticated Session Support.....	110
364	10.5 Authorization and Access Control.....	110
365	10.6 Audit Trail.....	110
366	11 Registry SAML Profile.....	112
367	11.1 Terminology.....	112
368	11.2 Use Cases for SAML Profile.....	112
369	11.2.1 Registry as SSO Participant:	112
370	11.3 SAML Roles Played By Registry.....	113
371	11.3.1 Service Provider Role.....	113
372	11.3.1.1 Service Provider Requirements.....	113
373	11.4 Registry SAML Interface.....	114
374	11.5 Requirements for Registry SAML Profile	114
375	11.6 SSO Operation.....	114
376	11.6.1 Scenario Actors.....	114
377	11.6.2 SSO Operation – Unauthenticated HTTP Requestor.....	115
378	11.6.2.1 Scenario Sequence.....	115
379	11.6.3 SSO Operation – Authenticated HTTP Requestor.....	116
380	11.6.4 SSO Operation – Unauthenticated SOAP Requestor.....	116
381	11.6.4.1 Scenario Sequence.....	117
382	11.6.5 SSO Operation – Authenticated SOAP Requestor.....	118
383	11.6.5.1 Scenario Sequence.....	119
384	11.6.6 <samlp:AuthnRequest> Generation Rules.....	120
385	11.6.7 <samlp:Response> Processing Rules.....	120
386	11.6.8 Mapping Subject to User.....	120
387	11.7 External Users.....	121
388	12 Native Language Support (NLS).....	122
389	12.1 Terminology.....	122
390	12.2 NLS and Registry Protocol Messages.....	122
391	12.3 NLS Support in RegistryObjects	122
392	12.3.1 Character Set of LocalizedString.....	124
393	12.3.2 Language of LocalizedString.....	124
394	12.4 NLS and Repository Items	124
395	12.4.1 Character Set of Repository Items.....	124
396	12.4.2 Language of Repository Items.....	124
397	13 Conformance.....	125

398	13.1 Conformance Profiles.....	125
399	13.2 Feature Matrix.....	125
400	14 References.....	129
401	14.1 Normative References.....	129
402	14.2 Informative.....	130
403		

Illustration Index

Figure 1: Simplified View of ebXML Registry Architecture.....	17
Figure 2: Registry Protocol Request-Response Pattern.....	19
Figure 3: Example Registry Package Hierarchy.....	32
Figure 4: Example of a Directory Listing.....	34
Figure 5: Submit Objects Protocol.....	36
Figure 6: Update Objects Protocol.....	38
Figure 7: Approve Objects Protocol.....	40
Figure 8: Deprecate Objects Protocol.....	41
Figure 9: Undeprecate Objects Protocol.....	43
Figure 10: Remove Objects Protocol.....	44
Figure 11: Ad Hoc Query Protocol.....	51
Figure 12: Filter Type Hierarchy.....	61
Figure 13: Content Validation Service.....	75
Figure 14: Content Cataloging Service.....	76
Figure 15: Content Management Service: Inline Invocation Model.....	78
Figure 16: Content Management Service: Decoupled Invocation Model.....	79
Figure 17: Cataloging Service Configuration.....	82
Figure 18: Validate Content Protocol.....	84
Figure 19: Catalog Content Protocol.....	86
Figure 20: Example of CPP cataloging using Canonical XML Cataloging Service.....	89
Figure 21: Inter-registry Object References.....	91
Figure 22: Registry Federations.....	92
Figure 23: Federation Metadata Example.....	93
Figure 24: Object Replication.....	96
Figure 25: Object Relocation.....	99
Figure 26: Relocate Objects Protocol.....	100
Figure 27: SAML SSO Typical Scenario.....	113
Figure 28: SSO Operation – Unauthenticated HTTP Requestor.....	115
Figure 29: SSO Operation - Unauthenticated SOAP Requestor.....	117
Figure 30: SSO Operation - Authenticated SOAP Requestor.....	119

1 Introduction

406 An ebXML Registry is an information system that securely manages any content type and the
407 standardized metadata that describes it.

408 The ebXML Registry provides a set of services that enable sharing of content and metadata between
409 organizational entities in a federated environment. An ebXML Registry may be deployed within an
410 application server, a web server or some other service container. The registry MAY be available to
411 clients as a public, semi-public or private web site.

412 This document defines the services provided by an ebXML Registry and the protocols used by clients
413 of the registry to interact with these services.

414 A separate document, ebXML Registry: Information Model [ebRIM], defines the types of metadata and
415 content that can be stored in an ebXML Registry.

1.1 Audience

417 The target audience for this specification is the community of software developers who are:

- 418 • Implementers of ebXML Registry Services
- 419 • Implementers of ebXML Registry Clients

1.2 Terminology

421 The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
422 RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF
423 RFC 2119 [RFC2119].

424 The term “*repository item*” is used to refer to content (e.g., an XML document or a DTD) that resides in
425 a repository for storage and safekeeping. Each repository item is described by a RegistryObject
426 instance. The RegistryObject catalogs the RepositoryItem with metadata.

1.3 Notational Conventions

428 Throughout the document the following conventions are employed to define the data structures used.
429 The following text formatting conventions are used to aide readability:

1.3.1 UML Diagrams

431 Unified Modeling Language [UML] diagrams are used as a way to concisely describe concepts. They
432 are not intended to convey any specific Implementation or methodology requirements.

1.3.2 Identifier Placeholders

434 Listings may contain values that reference ebXML Registry objects by their id attribute. These id values
435 uniquely identify the objects within the ebXML Registry. For convenience and better readability, these
436 key values are replaced by meaningful textual variables to represent such id values.

437 For example, the placeholder in the listing below refers to the unique id defined for an example Service
438 object:

439

```
440 <rim:Service id="{EXAMPLE_SERVICE_ID}">
```

1.3.3 Constants

442 Constant values are printed in the Courier New font always, regardless of whether they are defined
443 by this document or a referenced document.

444 1.3.4 Bold Text

445 Bold text is used in listings to highlight those aspects that are most relevant to the issue being
446 discussed. In the listing below, an example value for the contentLocator slot is shown in italics
447 if that is what the reader should focus on in the listing:

448

```
449 <rim:Slot name="urn:oasis:names:tc:ebxml-  
450 regrep:rim:RegistryObject:contentLocator">  
451 ...  
452 </rim:Slot>
```

453

454 1.3.5 Example Values

455 These values are represented in *italic* font. In the listing below, an example value for the
456 contentLocator slot is shown in italics:

457

```
458 <rim:Slot name="urn:oasis:names:tc:ebxml-  
459 regrep:rim:RegistryObject:contentLocator">  
460 <rim:ValueList>  
461 <rim:Value>http://example.com/myschema.xsd</rim:Value>  
462 </rim:ValueList>  
463 </rim:Slot>
```

464

465 1.4 XML Schema Conventions

466 This specification uses schema documents conforming to W3C XML Schema [Schema1] and normative
467 text to describe the syntax and semantics of XML-encoded objects and protocol messages. In cases of
468 disagreement between the ebXML Registry schema documents and schema listings in this
469 specification, the schema documents take precedence. Note that in some cases the normative text of
470 this specification imposes constraints beyond those indicated by the schema documents.

471 Conventional XML namespace prefixes are used throughout this specification to stand for their
472 respective namespaces as follows, whether or not a namespace declaration is present in the example.
473 The use of these namespace prefixes in instance documents is non-normative. However, for
474 consistency and understandability instance documents SHOULD use these namespace prefixes.

475 1.4.1 Schemas Defined by ebXML Registry

476

Prefix	XML Namespace	Comments
rim:	urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0	This is the Registry Information Model namespace [ebRIM]. The prefix is generally elided in mentions of Registry Information Model elements in text.
rs:	urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0	This is the ebXML Registry namespace that defines base types for registry service requests and responses [ebRS]. The prefix is generally elided in mentions of ebXML Registry protocol-related elements in text.
query:	urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0	This is the ebXML Registry query namespace that is used in the query protocols used between clients and the QueryManager service [ebRS].

Prefix	XML Namespace	Comments
lcm:	urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0	This is the ebXML Registry Life Cycle Management namespace that is used in the life cycle management protocols used between clients and the LifeCycleManager service [ebRS].
cms:	urn:oasis:names:tc:ebxml-regrep:xsd:cms:3.0	This is the ebXML Registry Content Management Services namespace that is used in the content management protocols used between registry and pluggable content management services [ebRS].

477

478 **1.4.2 Schemas Used By ebXML Registry**

479

Prefix	XML Namespace	Comments
saml:	urn:oasis:names:tc:SAML:2.0:assertion	This is the SAML V2.0 assertion namespace [SAMLCore]. The prefix is generally elided in mentions of SAML assertion-related elements in text.
samlp:	urn:oasis:names:tc:SAML:2.0:protocol	This is the SAML V2.0 protocol namespace [SAMLCore]. The prefix is generally elided in mentions of XML protocol-related elements in text.
ecp:	urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp	This is the SAML V2.0 Enhanced Client Proxy profile namespace, specified in this document and in a schema [SAMLECP-xsd].
ds:	http://www.w3.org/2000/09/xmldsig#	This is the XML Signature namespace [XMLSig].
xenc:	http://www.w3.org/2001/04/xmlenc#	This is the XML Encryption namespace [XMLEnc].
SOAP-ENV:	http://schemas.xmlsoap.org/soap/envelope	This is the SOAP V1.1 namespace [SOAP1.1].
paos:	urn:liberty:paos:2003-08	This is the Liberty Alliance PAOS (reverse SOAP) namespace.
xsi:	http://www.w3.org/2001/XMLSchema-instance	This namespace is defined in the W3C XML Schema specification [Schema1] for schema-related markup that appears in XML instances.
wsse:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd	This namespace is defined by the Web Services Security: SOAP Message Security 1.0 specification [WSS-SMS]. It is used by registry to secure soap message communication.

Prefix	XML Namespace	Comments
wsu:	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd	This namespace is defined by the Web Services Security: SOAP Message Security 1.0 specification [WSS-SMS]. It is used by registry to secure soap message communication.

480

481 1.5 Registry Actors

482 This section describes the various actors who interact with the registry.

Actor	Description
Registry Operator	An organization that operates an ebXML Registry and makes its services available.
Registry Administrator	A privileged user of the registry that is responsible for performing administrative tasks necessary for the ongoing operation of the registry. Such a user is analogous to a “super user” that is authorized to perform <i>any</i> action.
Registry Guest	A user of the registry whose identity is not known to the registry. Such a user has limited privileges within the registry.
Registered User	A user of the registry whose identity is known to the registry as an authorized user of the registry.
Submitter	A user that submits content and or metadata to the registry. A Submitter MUST be a Registered User.
Registry Client	A software program that interacts with the registry using registry protocols.

483

484 1.6 Registry Use Cases

485 Once deployed, the ebXML Registry provides generic content and metadata management services and
486 as such supports an open-ended and broad set of use cases. The following are some common use
487 cases that are being addressed by ebXML Registry.

- 488 • Web Services Registry: publish, management, discovery and reuse of web service descriptions in
489 WSDL, ebXML CPPA and other forms.
- 490 • Controlled Vocabulary Registry: Enables publish, management, discovery and reuse of controlled
491 vocabularies including taxonomies, code lists, ebXML Core Components, XML Schema and UBL
492 schema.
- 493 • Business Process Registry: Enables publish, management, discovery and reuse of Business
494 Process specifications such as ebXML BPSS, BPEL and other forms.
- 495 • Electronic Medical Records Repository
- 496 • Geological Information System (GIS) Repository that stores GIS data from sensors

497

498 1.7 Registry Architecture

499 The following figure provides a simplified view of the architecture of the ebXML Registry.

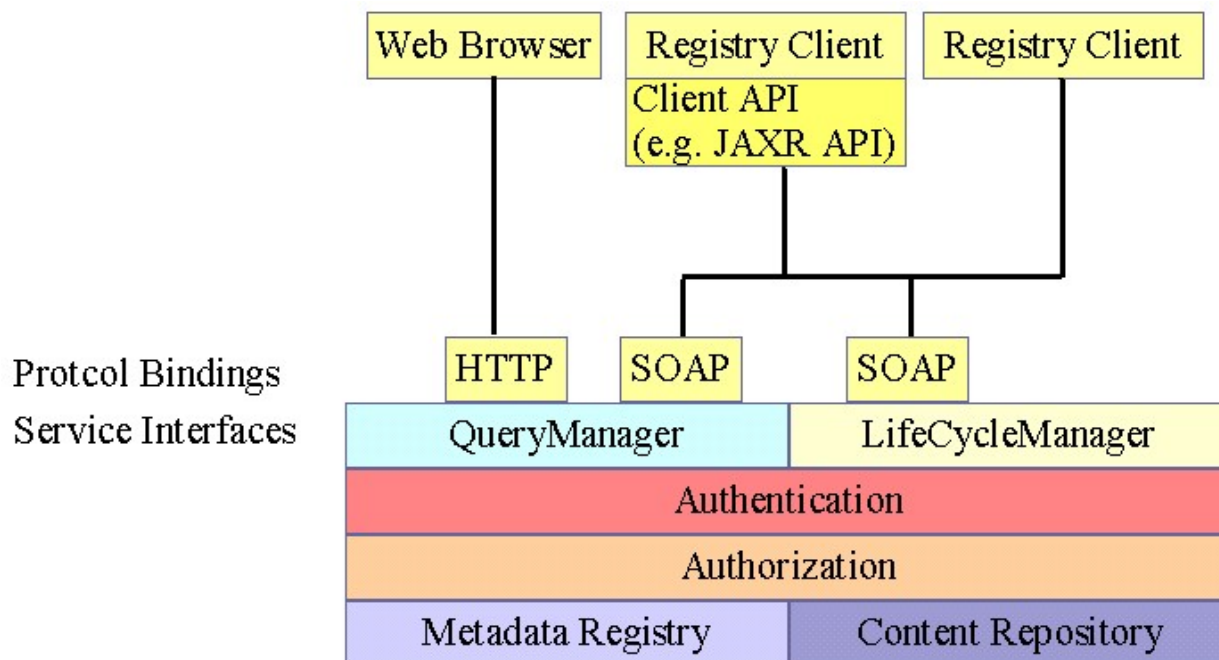


Figure 1: Simplified View of ebXML Registry Architecture

502 1.7.1 Registry Clients

503 A Registry Client is a software program that interacts with the registry using registry protocols. The
 504 Registry Client MAY be a Graphical User Interface (GUI), software service or agent. The Registry
 505 Client typically accesses the registry using SOAP 1.1 with Attachments [SwA] protocol.

506 A Registry Client may run on a client machine or may be a web tier service running on a server and
 507 may accessed by a web browser. In either case the Registry Client interacts with the registry using
 508 registry protocols.

509 1.7.1.1 Client API

510 A Registry client MAY access a registry interface directly. Alternatively, it MAY use a registry client API
 511 such as the Java API for XML Registries [JAXR] to access the registry. Client APIs such as [JAXR]
 512 provide programming convenience and are typically specific to a programming language.

513 1.7.2 Registry Service Interfaces

514 The ebXML Registry consists of the following service interfaces:

- 515 • A LifecycleManager interface that provides a collection of operations for end-to-end lifecycle
 516 management of metadata and content within the registry. This includes publishing, update, approval
 517 and deletion of metadata and content.
- 518 • A QueryManager interface that provides a collection of operations for the discovery and retrieval of
 519 metadata and content within the registry.

520 [RS-Interface-WSDL] provides an abstract (protocol neutral) definition of these Registry Service
 521 interfaces in WSDL format.

522 1.7.3 Service Interface: Protocol Bindings

523 This specification defines the following concrete protocol binding for the abstract service interfaces of
 524 the ebXML Registry:

- 525 • SOAP Binding that allows a Registry Client to access the registry using SOAP 1.1 with
526 Attachments [SwA]. [RS-Bindings-WSDL] defines the binding of the abstract Registry
527 Service interfaces to the SOAP protocol in WSDL format.
- 528 • HTTP Binding that allows a Web Browser client to access the registry using HTTP 1.1
529 protocol.
- 530 Additional bindings may be defined in the future as needed by the community.

531 **1.7.4 Authentication and Authorization**

532 A Registry Client SHOULD be authenticated by the registry to determine the identity associated with
533 them. Typically, this is the identity of the user associated with the Registry Client. Once the registry
534 determines the identity it MUST perform authorization and access control checks before permitting the
535 Registry Client's request to be processed.

536 **1.7.5 Metadata Registry and Content Repository**

537 An ebXML Registry is both a registry of metadata and a repository of content. A typical ebXML Registry
538 implementation uses some form of persistent store such as a database to store its metadata and
539 content. Architecturally, registry is distinct from the repository. However, all access to the registry as
540 well as repository is through the operations defined by the Registry Service interfaces.

2 Registry Protocols

541

542 This chapter introduces the registry protocols supported by the registry service interfaces. Specifically it
543 introduces the generic message exchange patterns that are common to all registry protocols.

2.1 Requests and Responses

544

545 Specific registry request and response messages derive from common types defined in XML Schema in
546 [RR-RS-XSD]. The Registry Client sends an element derived from **RegistryRequestType** to a registry,
547 and the registry generates an element adhering to or deriving from **RegistryResponseType**, as shown
548 next.

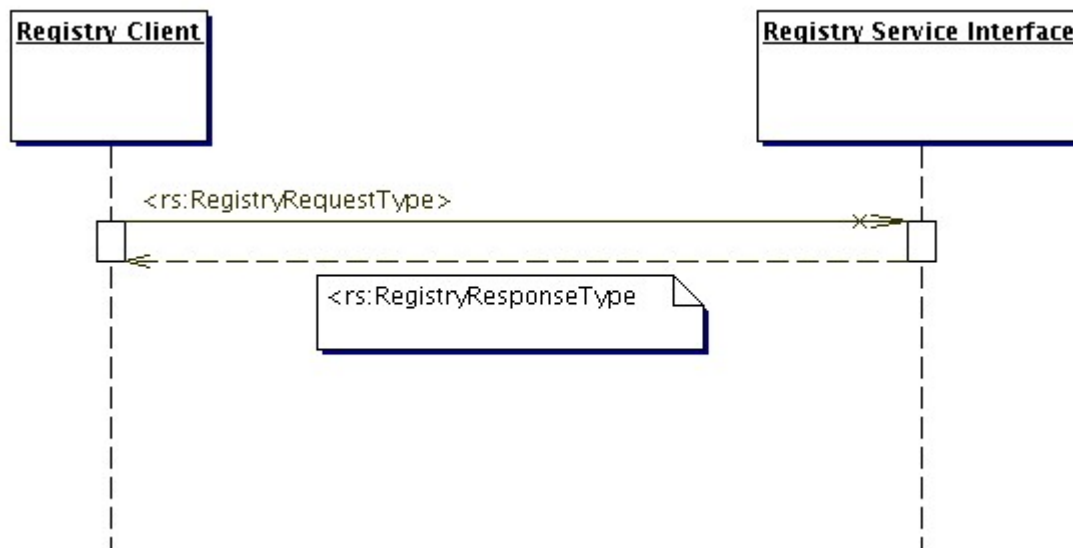


Figure 2: Registry Protocol Request-Response Pattern

550

551 Throughout this section, text mentions of elements and types are indicated with a namespace prefix.
552 The namespace prefix conventions are defined in the "Introduction" chapter.

553 Each registry request is atomic and either succeeds or fails in entirety. In the event of success, the
554 registry sends a RegistryResponse with a status of "Success" back to the client. In the event of failure,
555 the registry sends a RegistryResponse with a status of "Failure" back to the client. In the event of an
556 immediate response for an asynchronous request, the registry sends a RegistryResponse with a status
557 of "Unavailable" back to the client. Failure occurs when one or more Error conditions are raised in the
558 processing of the submitted objects. Warning messages do not result in failure of the request.

2.1.1 RegistryRequestType

559

560 The RegistryRequestType type is used as a common base type for all registry request messages.

2.1.1.1 Syntax:

561

```
562 <complexType name="RegistryRequestType">
563   <sequence>
564     <!-- every request may be extended using Slots. -->
565     <element maxOccurs="1" minOccurs="0" name="RequestSlotList"
566 type="rim:SlotListType"/>
567   </sequence>
568   <attribute name="id" type="anyURI" use="required"/>
```

```
569      <!--Comment may be used by requestor to describe the request. Used
570      in VersionInfo.comment-->
571      <attribute name="comment" type="string" use="optional"/>
572    </complexType>
573    <element name="RegistryRequest" type="tns:RegistryRequestType"/>
```

574 **2.1.1.2 Parameters:**

575 **comment:** This parameter allows the requestor to specify a string value that describes
576 the action being performed by the request. This parameter is used by the "Registry
577 Managed Version Control" feature of the registry.

578 **id:** This parameter specifies a request identifier that is used by the corresponding
579 response to correlate the response with its request. It MAY also be used to correlate a
580 request with another related request. The value of the id parameter MUST abide by the
581 same constraints as the value of the id attribute for the <rim:IdentifiableType> type.

582 **RequestSlotList:** This parameter specifies a collection of Slot instances. A
583 RegistryRequestType MAY include Slots as an extensibility mechanism that provides a
584 means of adding additional attributes to the request in form of Slots. The use of registry
585 implementation specific slots MUST be ignored silently by a registry that does not
586 support such Slots and MAY not be interoperable across registry implementations.
587

588 **2.1.1.3 Returns:**

589 All RegistryRequests return a response derived from the common RegistryResponseType base type.

590 **2.1.1.4 Exceptions:**

591 The following exceptions are common to all registry protocol requests:

592 **AuthorizationException:** Indicates that the requestor attempted to perform an
593 operation for which he or she was not authorized.

594 **InvalidRequestException:** Indicates that the requestor attempted to perform an
595 operation that was semantically invalid.

596 **SignatureValidationException:** Indicates that a Signature specified for the request
597 failed to validate.

598 **TimeoutException:** Indicates that the processing time for the request exceeded a
599 registry specific limit.

600 **UnsupportedCapabilityException:** Indicates that this registry did not support the
601 capability required to service the request.

602 In addition to above exceptions there are additional exceptions defined by [WSS-SMS] that a registry
603 protocol request MUST return when certain errors occur during the processing of the <wsse:Security>
604 SOAP Header element.

605 **2.1.2 RegistryRequest**

606 RegistryRequest is an element whose base type is RegistryRequestType. It adds no additional
607 elements or attributes beyond those described in RegistryRequestType. The RegistryRequest element
608 MAY be used by a registry to support implementation specific registry requests.

609 **2.1.3 RegistryResponseType**

610 The RegistryResponseType type is used as a common base type for all registry responses.

611 2.1.3.1 Syntax:

```
612 <complexType name="RegistryResponseType">
613   <sequence>
614     <!-- every response may be extended using Slots. -->
615     <element maxOccurs="1" minOccurs="0" name="ResponseSlotList"
616 type="rim:SlotListType"/>
617     <element minOccurs="0" ref="tns:RegistryErrorList"/>
618   </sequence>
619   <attribute name="status" type="rim:referenceURI" use="required"/>
620   <!-- id is the request id for the request for which this is a
621 response -->
622   <attribute name="requestId" type="anyURI" use="optional"/>
623 </complexType>
624 <element name="RegistryResponse" type="tns:RegistryResponseType"/>
```

625 2.1.3.2 Parameters:

626 **status:** The status attribute is used to indicate the status of the request. The value of
627 the status attribute MUST be a reference to a ClassificationNode within the canonical
628 ResponseStatusType ClassificationScheme as described in [ebRIM]. A Registry MUST
629 support the status types as defined by the canonical ResponseStatusType
630 ClassificationScheme. The canonical ResponseStatusType ClassificationScheme may
631 be extended by adding additional ClassificationNodes to it.

632 The following canonical values are defined for the ResponseStatusType
633 ClassificationScheme:

- 634 • **Success** - This status specifies that the request was successful.
- 635 • **Failure** - This status specifies that the request encountered a failure. One or more
636 errors MUST be included in the RegistryErrorList in this case or returned as a
637 SOAP Fault.
- 638 • **Unavailable** – This status specifies that the response is not yet available. This may
639 be the case if this RegistryResponseType represents an immediate response to an
640 asynchronous request where the actual response is not yet available.

641 **requestId:** This parameter specifies the id of the request for which this is a response.
642 It matches value of the id attribute of the corresponding RegistryRequestType.

643 **ResponseSlotList:** This parameter specifies a collection of Slot instances. A
644 RegistryResponseType MAY include Slots as an extensibility mechanism that provides
645 a means of adding dynamic attributes in form of Slots. The use of registry
646 implementation specific slots MUST be ignored silently by a Registry Client that does
647 not support such Slots and MAY not be interoperable across registry implementations.

648 **RegistryErrorList:** This parameter specifies an optional collection of RegistryError
649 elements in the event that there are one or more errors that were encountered while
650 the registry processed the request for this response. This is described in more detail in
651 6.9.4.

652 2.1.4 RegistryResponse

653 RegistryResponse is an element whose base type is RegistryResponseType. It adds no additional
654 elements or attributes beyond those described in RegistryResponseType. RegistryResponse is used by
655 many registry protocols as their response.

656 2.1.5 RegistryErrorList

657 A RegistryErrorList specifies an optional collection of RegistryError elements in the event that there are

658 one or more errors that were encountered while the registry processed a request.

659 2.1.5.1 Syntax:

```
660 <element name="RegistryErrorList">
661   <complexType>
662     <complexContent>
663       <restriction base="{http://www.w3.org/2001/XMLSchema}anyType">
664         <sequence>
665           <element ref="rs:RegistryError" maxOccurs="unbounded"/>
666         </sequence>
667         <attribute name="highestSeverity" type="rim:referenceURI" />
668       </restriction>
669     </complexContent>
670   </complexType>
671 </element>
```

672 2.1.5.2 Parameters:

673 *highestSeverity*: This parameter specifies the ErrorType for the highest severity
674 RegistryError in the RegistryErrorList. Values for highestSeverity are defined by
675 ErrorType in .

676 *RegistryError*: A RegistryErrorList has one or more RegistryErrors. A RegistryError
677 specifies an error or warning message that is encountered while the registry processes
678 a request. RegistryError is defined in 2.1.6.

679

680 2.1.6 RegistryError

681 A RegistryError specifies an error or warning message that is encountered while the registry processes
682 a request.

683 2.1.6.1 Syntax:

```
684 <element name="RegistryError">
685   <complexType>
686     <simpleContent>
687       <extension base="string">
688         <attribute name="codeContext" type="string" use="required"/>
689         <attribute name="errorCode" type="string" use="required"/>
690         <attribute default="urn:oasis:names:tc:ebxml-
691 regrep:ErrorSeverityType:Error" name="severity" type="rim:referenceURI"
692 />
693         <attribute name="location" type="string" use="optional"/>
694       </extension>
695     </simpleContent>
696   </complexType>
697 </element>
```

698 2.1.6.2 Parameters:

699 *codeContext*: This attribute specifies a string that indicates contextual text that
700 provides additional detail to the errorCode. For example, if the errorCode is
701 InvalidRequestException the codeContext MAY provide the reason why the request
702 was invalid.

703 *errorCode*: This attribute specifies a string that indicates the error that was
704 encountered. Implementations MUST set this attribute to the Exception or Error as
705 defined by this specification (e.g. InvalidRequestException).

706 *severity*: This attribute indicates the severity of error that was encountered. The value
707 of the severity attribute MUST be a reference to a ClassificationNode within the
708 canonical ErrorSeverityType ClassificationScheme as described in [ebRIM]. A Registry
709 MUST support the error severity types as defined by the canonical ErrorSeverityType
710 ClassificationScheme. The canonical ErrorSeverityType ClassificationScheme may be
711 extended by adding additional ClassificationNodes to it.

712 The following canonical values are defined for the ErrorSeverityType
713 ClassificationScheme:

- 714 • **Error** – An Error is a fatal error encountered by the registry while processing a
715 request. A registry MUST return a status of Failure in the RegistryResponse for a
716 request that encountered Errors during its processing.
- 717 • **Warning** – A Warning is a non-fatal error encountered by the registry while
718 processing a request. A registry MUST return a status of Success in the
719 RegistryResponse for a request that only encountered Warnings during its
720 processing and encountered no Errors.

721 *location*: This attribute specifies a string that indicated where in the code the error
722 occurred. Implementations SHOULD show the stack trace and/or, code module and line
723 number information where the error was encountered in code.

3 SOAP Binding

This chapter defines the SOAP protocol binding for the ebXML Registry service interfaces. The SOAP binding enables access to the registry over the SOAP 1.1 with Attachments [SwA] protocol. The complete SOAP Binding is described by the following WSDL description files:

- ebXML Registry Service Interfaces: Abstract Definition [RR-INT-WSDL]
- ebXML Registry Service Interfaces: SOAP Binding [RR-SOAPB-WSDL]
- ebXML Registry Service Interfaces: SOAP Service [RR-SOAPS-WSDL]

3.1 ebXML Registry Service Interfaces: Abstract Definition

In [RR-INT-WSDL], each registry Service Interface is mapped to an abstract WSDL portType as follows:

- A portType is defined for each Service Interface:

```
<portType name="QueryManagerPortType">
...
</portType>
<portType name="LifecycleManagerPortType">
...
</portType>
```

- Within each portType an operation is defined for each protocol supported by the service interface:

```
<portType name="QueryManagerPortType">
  <operation name="submitAdhocQuery">
    ...
  </operation>
</portType>
```

- Within each operation the request and response message for the corresponding protocol are defined as input and output for the operation:

```
<portType name="QueryManagerPortType">
  <operation name="submitAdhocQuery">
    <input message="tns:msgAdhocQueryRequest"/>
    <output message="tns:msgAdhocQueryResponse"/>
  </operation>
</portType>
```

- For each message used in an operation a message element is defined that references the element corresponding to the registry protocol request or response message from the XML Schema for the registry service interface [RR-LCM-XSD], [RR-QM-XSD]:

```
<message name="msgAdhocQueryRequest">
  <part element="query:AdhocQueryRequest"
    name="partAdhocQueryRequest"/>
</message>
<message name="msgAdhocQueryResponse">
  <part element="query:AdhocQueryResponse"
    name="partAdhocQueryResponse"/>
</message>
```


772 3.2 ebXML Registry Service Interfaces SOAP Binding

773 In [RR-SOAPB-WSDL], a SOAP Binding is defined for the registry service interfaces as follows:

- 774 • For each portType corresponding to a registry service interface and defined in [RR-INT-WSDL] a
775 <binding> element is defined which has name <ServiceInterfaceName>Binding
- 776 • The <binding> element references the portType defined in [RR-INT-WSDL] via its type attribute
- 777 • The <soap:binding> extension element uses the “document” style
- 778 • An operation element is defined for each protocol defined for the service interface. The operation
779 name relates to the protocol request message.
- 780 • The <soap:operation> extension element has <input> and <output> elements that have
781 <soap:body> elements with use="literal".

782

```
783     <binding name="QueryManagerBinding"  
784     type="interfaces:QueryManagerPortType">  
785       <soap:binding style="document"  
786       transport="http://schemas.xmlsoap.org/soap/http"/>  
787       <operation name="submitAdhocQuery">  
788         <soap:operation soapAction="urn:oasis:names:tc:ebxml-  
789 regrep:wSDL:registry:bindings:3.0:QueryManagerPortType#submitAdhocQuery  
790 "/>  
791         <input>  
792           <soap:body use="literal"/>  
793         </input>  
794         <output>  
795           <soap:body use="literal"/>  
796         </output>  
797       </operation>  
798     </binding>
```

799

800 3.3 ebXML Registry Service Interfaces SOAP Service Template

801 In [RR-SOAPS-WSDL], a non-normative template is provided for a WSDL Service that uses the SOAP
802 Binding from the registry service interfaces as follows:

- 803 • A single service element defines the concrete ebXML Registry SOAP Service. The template uses
804 the name “ebXMLRegistrySOAPSService”.
- 805 • The service element includes a port definitions, where each port corresponds with one of the service
806 interfaces defined for the registry. Each port includes an HTTP URL for accessing that port specified
807 by the location attribute of the <soap:address> element. The HTTP URL to the SOAP Service MUST
808 conform to the pattern <base URL>/soap where <base URL> MUST be the same as the value of the
809 *home* attribute of the instance of the Registry class defined by [ebRIM] that represents this registry.
- 810 • Each port definition also references a SOAP binding element described in the previous section.

811

```
812     <service name="ebXMLRegistrySOAPSService">  
813       <port binding="bindings:QueryManagerBinding"  
814       name="QueryManagerPort">  
815         <soap:address location="http://your.server.com/soap"/>  
816       </port>  
817       <port binding="bindings:LifeCycleManagerBinding"  
818       name="LifeCycleManagerPort">  
819         <soap:address location="http://your.server.com/soap"/>  
820       </port>  
821     </service>
```

822

823

3.4 Mapping of Exception to SOAP Fault

824 The registry protocols defined in this specification include the specification of Exceptions that a registry
825 MUST return when certain exceptional conditions are encountered during the processing of the
826 protocol request message. A registry MUST return Exceptions specified in registry protocol messages
827 as SOAP Faults as described in this section. In addition a registry MUST conform to [WSI-BP] when
828 generating the SOAP Fault. A registry MUST NOT sign a SOAP Fault message it returns.

829 The following table provides details on how a registry MUST map exceptions to SOAP Faults.

830

SOAP Fault Element	Description	Example
faultcode	The faultCode MUST be present and MUST be the name of the Exception qualified by the URN prefix: urn:oasis:names:tc:ebxml-regrep:rs:exception:	<i>urn:oasis:names:tc:ebxml-regrep:rs:exception:ObjectNotFoundException</i>
faultstring	The faultstring MUST be present and SHOULD provide some information explaining the nature of the exception.	<i>Object with id urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription not found in registry.</i>
detail	At least one detail element MUST be present. The detail element SHOULD include the stack trace and/or, code module and line number information where the Exception was encountered in code. If the Exception has nested Exceptions within it then the registry SHOULD include the nested exceptions as nested detail elements within the top level detail element.	
faultactor	At least one faultactor MUST be present. The first faultactor MUST be the base URL of the registry.	<i>http://example.server.com:8080/oma/r/registry</i>

Table 1: Mapping a Registry Exception to SOAP Fault

4 HTTP Binding

831

832 This chapter defines the HTTP protocol binding for the ebXML Registry abstract service interfaces. The
833 HTTP binding enables access to the registry over the HTTP 1.1 protocol.

834 The HTTP interface provides multiple options for accessing RegistryObjects and RepositoryItems via
835 the HTTP protocol. These options are:

836

- 837 • RPC Encoding URL: Allows client access to objects via a URL that is based on encoding a
838 Remote Procedure Call (RPC) to a registry interface as an HTTP protocol request.
- 839 • Submitter Defined URL: Allows client access to objects via Submitter defined URLs.
- 840 • File Path Based URL: Allows clients access to objects via a URL based upon a file path derived
841 from membership of object in a RegistryPackage membership hierarchy.

842 Each of the above methods has its advantages and disadvantages and each method may be better
843 suited for different use cases as illustrated by table below:

844

HTTP Access Method	Advantages	Disadvantages
RPC Encoding URL	<ul style="list-style-type: none">• The URL is constant and deterministic• Submitter need not explicitly assign URL	<ul style="list-style-type: none">• The URL is long and not human-friendly to remember
Submitter Defined URL	<ul style="list-style-type: none">• Very human-friendly URL• Submitter may assign any URL• The URL is constant and deterministic	<ul style="list-style-type: none">• Submitter must explicitly assign URL• Requires additional resources in the registry
File Path Based URL	<ul style="list-style-type: none">• Submitter need not explicitly assign URL• Intuitive URL that is based upon a familiar file / folder metaphor	<ul style="list-style-type: none">• The URL is NOT constant and deterministic• Requires placing objects as members in RegistryPackages

Table 2: Comparison of HTTP Access Methods

4.1 HTTP Interface URL Pattern

845

846 The HTTP URLs used by the HTTP Binding MUST conform to the pattern *<base URL>/http/<url suffix>*
847 where *<base URL>* MUST be the same as the value of the *home* attribute of the instance of the
848 Registry class defined by [ebRIM] that represents this registry. The *<url suffix>* depends upon the
849 HTTP Access Method and various request specific parameters that will be described later in this
850 chapter.

4.2 RPC Encoding URL

851

852 The RPC Encoding URL method of the HTTP interface maps the operations defined by the abstract
853 registry interfaces to the HTTP protocol using an RPC style. It defines how URL parameters are used to
854 specify the interface, method and invocation parameters needed to invoke an operation on a registry
855 interface such as the QueryManager interface.

856 The RPC Encoding URL method also defines how an HTTP response is used to carry the response
857 generated by the operation specified in the request.

4.2.1 Standard URL Parameters

858

859 The following table specifies the URL parameters supported by RPC Encoding URLs. A Registry MAY

860 implement additional URL parameters in addition to these parameters. Note that the URL Parameter
 861 names MUST be processed by the registry in a case-insensitive manner while the parameter values
 862 MUST be processed in a case-sensitive manner.

URL Parameter	Required	Description	Example
interface	YES	Defines the service interface that is the target of the request.	QueryManager
method	YES	Defines the method (operation) within the interface that is the target of the request.	getRegistryObject
param-<key>	NO	Defines named parameters to be passed into a method call. Note that some methods require specific parameters.	param-id= <i>urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription</i>

Table 3: Standard URL Parameters

863 4.2.2 QueryManager Binding

864 A registry MUST support a RPC Encoded URL HTTP binding to QueryManager service interface. To
 865 specify the QueryManager interface as its target, the *interface* parameter of the URL MUST be
 866 "QueryManager." In addition the following URL parameters are defined by the QueryManager HTTP
 867 Interface.

868

Method	Parameter	Return Value	HTTP Request Type
getRegistryObject	id	The RegistryObject that matches the specified id.	GET
getRepositoryItem	id	The RepositoryItem that matches the specified id. Note that a RepositoryItem may be arbitrary content (e.g. a GIF image).	GET

Table 4: RPC Encoded URL: Query Manager Methods

869

870 Note that in the examples that follow, name space declarations are omitted to conserve space. Also
 871 note that some lines may be wrapped due to lack of space.

872 4.2.2.1 Sample getRegistryObject Request

873 The following example shows a getRegistryObject request.

874

```
875 GET /http?interface=QueryManager&method=getRegistryObject&param-  
876 id= urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription  
877 HTTP/1.1
```

878

879 4.2.2.2 Sample getRegistryObject Response

880 The following example shows an ExtrinsicObject, which is a concrete sub-class of RegistryObject being
 881 returned as a response to the getRegistryObject method invocation.

882

```
883 HTTP/1.1 200 OK
884 Content-Type: text/xml
885 Content-Length: 555
886
887 <?xml version="1.0"?>
888 <ExtrinsicObject
889   id =
890   "urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription"
891   objectType="{OBJECT_TYPE}">
892   ...
893 </ExtrinsicObject>
```

894

895 **4.2.2.3 Sample getRepositoryItem Request**

896 The following example shows a getRepositoryItem request.

897

```
898 GET /http?interface=QueryManager&method=getRepositoryItem&param-
899 id= urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription
900 HTTP/1.1
```

901

902 **4.2.2.4 Sample getRepositoryItem Response**

903 The following example assumes that the repository item was a Collaboration Protocol Profile as defined
904 by [ebCPP]. It could return any type of content (e.g. a GIF image).

905

```
906 HTTP/1.1 200 OK
907 Content-Type: text/xml
908 Content-Length: 555
909
910 <?xml version="1.0"?>
911 <CollaborationProtocolProfile>
912   ...
913 </CollaborationProtocolProfile>
```

914

915 **4.2.3 LifeCycleManager HTTP Interface**

916 The RPC Encoded URL mechanism of the HTTP Binding does not support the LifeCycleManager
917 interface. The reason is that the LifeCycleManager operations require HTTP POST which is already
918 supported by the SOAP binding.

919 **4.3 Submitter Defined URL**

920 A Submitter MAY specify zero or more Submitter defined URLs for a RegistryObject or RepositoryItem.
921 These URLs MAY then be used by clients to access the object using the GET request of the HTTP
922 protocol. Submitter defined URLs serve as an alternative to the RPC Encoding URL defined by the
923 HTTP binding for the QueryManager interface. The benefit of Submitter defined URLs is that objects
924 are made accessible via a URL that is meaningful and memorable to the user. The cost of Submitter
925 defined URLs is that the Submitter needs to specify the Submitter defined URL and that the Submitter
926 defined URL takes additional storage resources within the registry.

927 Consider the examples below to see how Submitter defined URLs compare with the URL defined by the
928 HTTP binding for the QueryManager interface.

929 Following is a sample URL defined by the HTTP binding for the QueryManager interface to access a
930 RegistryObject that is an ExtrinsicObject describing a GIF image:

```
931  
932  
933 http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&met  
934 hod=getRegistryObject&param-  
935 id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription  
936
```

937
938 The same RegistryObject (an ExtrinsicObject) may be accessed via the following Submitter defined
939 URL:

```
940  
941  
942 http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.xml  
943  
944
```

945 Following is a sample URL defined by the HTTP binding for the QueryManager interface to access a
946 repository item that is a GIF image:

```
947  
948  
949 http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&met  
950 hod=getRepositoryItem&param-  
951 id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription  
952
```

953
954 The same repository item may be accessed via the following Submitter defined URL:

```
955  
956  
957 http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.jpg  
958  
959
```

960 4.3.1 Submitter defined URL Syntax

961 A Submitter MUST specify a Submitter defined URL as a URL suffix that is relative to the base URL of
962 the registry. The URL suffix for a Submitter defined URL MUST be unique across all Submitter defined
963 URLs defined for all objects within a registry.

964 The use of relative URLs is illustrated as follows:

- 965 • **Base URL for Registry:** <http://localhost:8080/ebxml/registry>
- 966 • **Implied Prefix URL for HTTP interface:** <http://localhost:8080/ebxml/registry/http>
- 967 • **Submitter Defined URL suffix:** /pictures/nikola/zeus
- 968 • **Complete URL:** <http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus>

969 4.3.2 Assigning URL to a RegistryObject

970 A Submitter MAY assign one or more Submitter defined URLs to a RegistryObject.

971 The Submitter defined URL(s) MAY be assigned by the Submitter using a canonical slot on the
972 RegistryObject. The Slot is identified by the name:

973

974

975

976

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:locator
```

977 Each value in the collection of values for this Slot specifies a Submitter defined URL suffix for that
978 RegistryObject. The registry MUST return the RegistryObject when the HTTP client sends an HTTP
979 GET request whose URL matches any of the URLs specified within the locator Slot (if any) for that
980 RegistryObject.

981 **4.3.3 Assigning URL to a Repository Item**

982 A Submitter MAY assign one or more Submitter defined URLs to a Repository Item.

983 The Submitter defined URL(s) may be assigned by the Submitter using a canonical slot on the
984 ExtrinsicObject for the repository item. The Slot is identified by the name:

985

986

987

988

```
urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:contentLocator
```

989 Each value in the collection of values for this Slot specifies a Submitter defined URL suffix for the
990 RepositoryItem associated with the ExtrinsicObject. The registry MUST return the RepositoryItem when
991 the HTTP client sends an HTTP GET request whose URL matches any of the URLs specified within the
992 contentLocator slot (if any) for the ExtrinsicObject for that RepositoryItem.

993 **4.4 File Path Based URL**

994 The File Path Based URL mechanism enables HTTP clients to access RegistryObjects and
995 RepositoryItems using a URL that is derived from the RegistryPackage membership hierarchy for the
996 RegistryObject or RepositoryItem.

997 **4.4.1 File Folder Metaphor**

998 The RegistryPackage class as defined by [ebRIM] enables objects to be structurally organized by a
999 RegistryPackage membership hierarchy. As such, a RegistryPackage serves a role similar to that of a
1000 Folder within the File and Folder metaphor that is common within filesystems in most operating
1001 systems. Similarly, the members of a RegistryPackage serve a role similar to the files within a folder in
1002 the File and Folder metaphor.

1003 In this file-folder metaphor, a Submitter creates a RegistryPackage to create the functional equivalent
1004 of a folder and creates a RegistryObject to create the functional equivalent of a file. The Submitter adds
1005 a RegistryObjects as a member of a RegistryPackage to create the functional equivalent of adding a
1006 file to a folder.

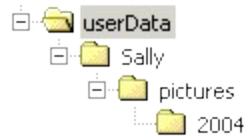
1007 **4.4.2 File Path of a RegistryObject**

1008 Each RegistryObject has an implicit *file path*. The file path of a RegistryObject is a path structure
1009 similar to the Unix file path structure. The file path is composed of file path segments. Analogous to the
1010 Unix file path, the last segment within the file path represents the RegistryObject, while preceding
1011 segments represent the RegistryPackage(s) within the membership hierarchy of the RegistryObject.
1012 Each segment consists of the *name* of the RegistryPackage or the RegistryObject. Because the name
1013 attribute is of type InternationalString the path segment matches the name of an object within a specific
1014 locale.

1015 4.4.2.1 File Path Example

1016 Consider the example where a registry has a RegistryPackage hierarchy as illustrated below using the
1017 name of the objects in locale "en_US":

1018



1019

Figure 3: Example Registry Package Hierarchy

1020 Now let us assume that the RegistryPackage named "2004" has an ExtrinsicObject named "baby.gif"
1021 for a repository item that is a photograph in the GIF format. In this example the file paths for various
1022 objects in locale "en_US" are shown in table below:

1023

Object Name	File Path
userData	/userData
Sally	/userData/Sally
pictures	/userData/Sally/pictures
2004	/userData/Sally/pictures/2004
baby.gif	/userData/Sally/pictures/2004/baby.gif

Table 5: File Path Examples

1024 Note that above example assumes that the RegistryPackage named userData is a root level package
1025 (not contained within another RegistryPackage).

1026 4.4.3 Matching URL To Objects

1027 A registry client MAY access RegistryObjects and RepositoryItems over the HTTP GET request using
1028 URL patterns that are based upon the File Path for the target objects. This section describes how a
1029 registry resolves File Path URLs specified by an HTTP client.

1030 The registry MUST process each path segment from the beginning of the path to the end and for each
1031 path segment match the segment to the value attribute of a LocalizedString in the name attribute of a
1032 RegistryObject. For all but the last path segment, the matched RegistryObject MUST be a
1033 RegistryPackage. The last path segment MAY match any RegistryObject including a RegistryPackage.
1034 If any path segment fails to be matched then the URL is not resolvable by the File Path based URL
1035 method. When matching any segment other than the first segment the registry MUST also ensure that
1036 the matched RegistryObject is a member of the RegistryPackage that matches the previous segment.

1037 4.4.4 URL Matches a Single Object

1038 When a File Path based URL matches a single object there are two possible responses.

1039

- 1040 • If the URL pattern does not end in a '/' character or the last segment does not match a
1041 RegistryPackage then the Registry MUST send as response an XML document that is the
1042 XML representation of the RegistryObject that matches the last segment. If the last
1043 segment matches an ExtrinsicObject then if the URL specifies the HTTP GET parameter
1044 with name 'getRepositoryItem' and value of 'true' then the registry MUST return as
1045 response the repository item associated with the ExtrinsicObject.

1046 • If the URL pattern ends in a '/' character and the last segment matches a RegistryPackage
1047 then the Registry MUST send as response an HTML document that is the directory
1048 listing (section 4.4.6) of all RegistryObjects that are members of the RegistryPackage
1049 that matches the last segment.
1050

1051 **4.4.5 URL Matches Multiple Object**

1052 A registry MUST show a partial Directory Listing of a Registry Package when a File Path
1053 based URL matches multiple objects.

1054 A File Path based URL may match multiple objects if:

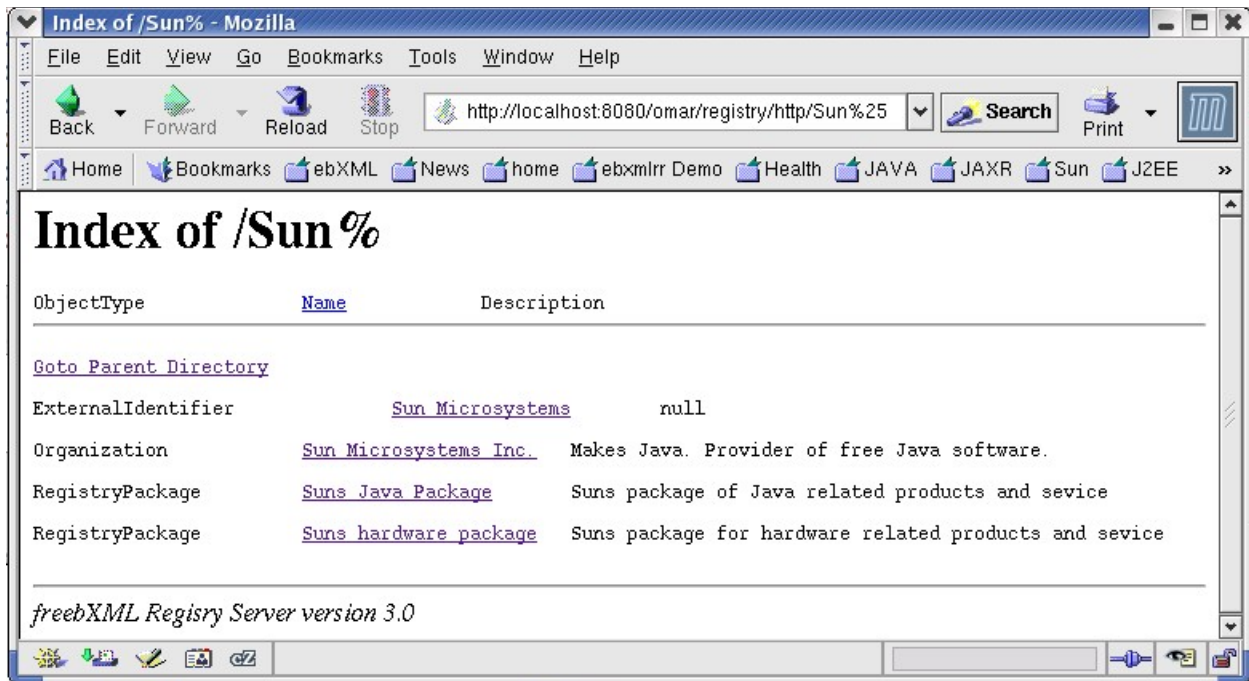
- 1056 • Multiple objects with the same name exist in the same RegistryPackage
- 1057 • The segment contains wildcard characters such as '%' or '?' to match the names of multiple
1058 objects within the same RegistryPackage. Note that wildcard characters must be URL encoded
1059 as defined by the HTTP protocol. For example the '%' character is encoded as '%25'.
1060

1061 **4.4.6 Directory Listing**

1062 A registry MUST return a directory listing as a response under certain circumstances as describes
1063 earlier. The directory listing MUST show a list of objects within a specific RegistryPackage.

1064 A registry SHOULD structure a directory listing such that each item in the listing provides information
1065 about a RegistryObject within the RegistryPackage. A registry MAY format its directory listing page in a
1066 registry specific manner. However, it is suggested that a registry SHOULD format it as an HTML page
1067 that minimally includes the objectType, name and description attributes for each RegistryObject in the
1068 directory listing.

1069 Figure 4 shows a non-normative example of a directory listing that matches all root level objects that
1070 have a name that begins with 'Sun' (path /Sun%25).
1071



1072

Figure 4: Example of a Directory Listing

1073 4.4.7 Access Control In RegistryPackage Hierarchy

1074 The ability to control who can add files and sub-folders to a folder is important in a file system. The
 1075 same is true for the File Path Based URL mechanism.

1076 A Submitter MAY assign a custom Access Control Policy to a Registry Package to create the functional
 1077 equivalent of assigning access control to a folder in the file-folder metaphor. The custom Access
 1078 Control Policy SHOULD use the "reference" action to control who can add RegistryObjects as members
 1079 of the folder as described in [eBRIM].

1080 4.5 URL Resolution Algorithm

1081 Since the HTTP Binding supports multiple mechanisms to resolve an HTTP URL a registry SHOULD
 1082 implement an algorithm to determine the correct HTTP Binding mechanism to resolve a URL.

1083 This section gives a non-normative URL resolution algorithm that a registry SHOULD use to determine
 1084 which of the various HTTP Binding mechanisms to use to resolve an HTTP URL.

1085 Upon receiving an HTTP GET request a registry SHOULD first check if the URL is an RPC Encoded
 1086 URL. This MAY be done by checking if the *interface* URL parameter is specified in the URL. If specified
 1087 the registry SHOULD resolve the URL using the RPC Encoded URL method as defined by section 4.2.
 1088 If the *interface* URL parameter is not specified then the registry SHOULD use the Submitter specified
 1089 URL method to check if the URL is resolvable. If the URL is still unresolvable then the registry SHOULD
 1090 check if the URL is resolvable using the File Path based URL method. If the URL is still unresolvable
 1091 then the registry should return an HTTP 404 (NotFound) error as defined by the HTTP protocol.

1092 4.6 Security Consideration

1093 A registry MUST enforce all Access Control Policies including restriction on the READ action when
 1094 processing a request to the HTTP binding of a service interface. This implies that a Registry MUST not
 1095 resolve a URL to a RegistryObject or RepositoryItem if the client is not authorized to read that object.

1096 **4.7 Exception Handling**

1097 If a service interface method generates an Exception it MUST be reported in a `RegistryErrorList`,
1098 and sent back to the client within the HTTP response for the HTTP request.

1099 When errors occur, the HTTP status code and message SHOULD correspond to the error(s) being
1100 reported in the `RegistryErrorList`. For example, if the `RegistryErrorList` reports that an object
1101 wasn't found, therefore cannot be returned, an appropriate error code SHOULD be 404, with a
1102 message of "ObjectNotFoundException". A detailed list of HTTP status codes can be found in
1103 [RFC2616]. The mapping between registry exceptions and HTTP status codes is currently unspecified.

5 Lifecycle Management Protocols

1104

1105 This section defines the protocols supported by Lifecycle Management service interface of the Registry.
1106 The Lifecycle Management protocols provide the functionality required by RegistryClients to manage
1107 the lifecycle of RegistryObjects and RepositoryItems within the registry.

1108 The XML schema for the Lifecycle Management protocols is described in [RR-LCM-XSD].

5.1 Submit Objects Protocol

1109

1110 This SubmitObjects allows a RegistryClient to submit one or more RegistryObjects and/or repository
1111 items.

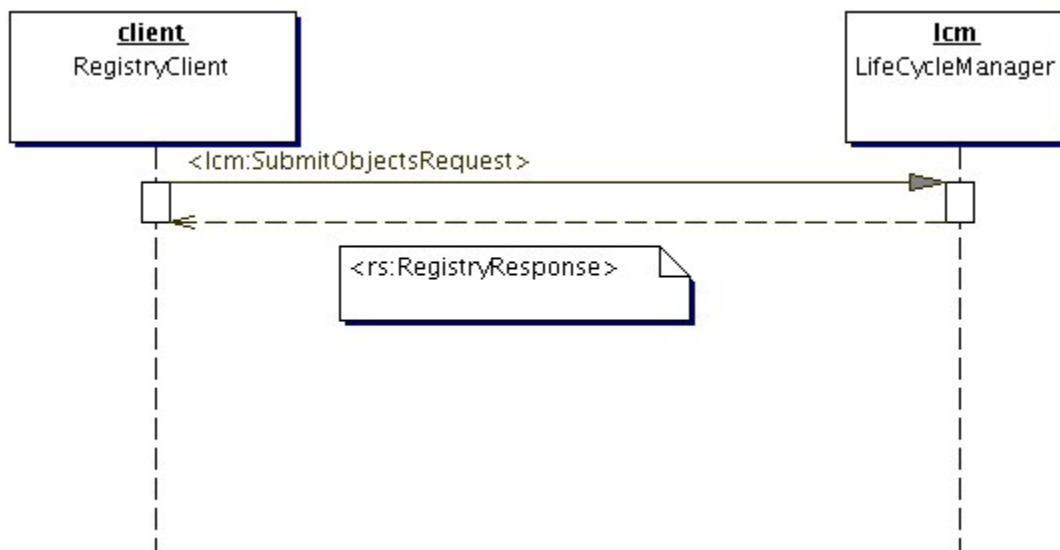


Figure 5: Submit Objects Protocol

1113

5.1.1 SubmitObjectsRequest

1114

1115 The SubmitObjectsRequest is used by a client to submit RegistryObjects and/or repository items to the
1116 registry.

5.1.1.1 Syntax:

1117

```
1118 <element name="SubmitObjectsRequest">  
1119 <complexType>  
1120 <complexContent>  
1121 <extension base="rs:RegistryRequestType">  
1122 <sequence>  
1123 <element ref="rim:RegistryObjectList"/>  
1124 </sequence>  
1125 </extension>  
1126 </complexContent>  
1127 </complexType>  
1128 </element>
```

1129 **5.1.1.2 Parameters:**

1130 *RegistryObjectList*: This parameter specifies a collection of RegistryObject instances
1131 that are being submitted to the registry. The RegistryObjects in the list may be brand
1132 new objects being submitted to the registry or they may be current objects already
1133 existing in the registry. In case of existing objects the registry MUST treat them in the
1134 same manner as UpdateObjectsRequest and simply update the existing objects.

1135 **5.1.1.3 Returns:**

1136 This request returns a RegistryResponse. See section 2.1.4 for details.

1137 **5.1.1.4 Exceptions:**

1138 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1139 be returned:

1140 *UnresolvedReferenceException*: Indicates that the requestor referenced an object
1141 within the request that was not resolved during the processing of the request.

1142 *UnsignedRepositoryItemException*: Indicates that the requestor attempted to submit a
1143 RepositoryItem that was not signed.

1144 *QuotaExceededException*: Indicates that the requestor attempted to submit more
1145 content than the quota allowed for them by the registry.

1146 **5.1.2 Unique ID Generation**

1147 As specified by [ebRIM], all RegistryObjects MUST have a unique id contained within the value of the
1148 id attribute. The id MUST be a valid URN and MUST be unique across all other RegistryObjects in the
1149 home registry for the RegistryObject.

1150 A Submitter MAY optionally supply the id attribute for submitted objects. If the Submitter supplies the id
1151 and it is a valid URN and does not conflict with the id of an existing RegistryObject within the home
1152 registry then the registry MUST honor the Submitter-supplied id value and use it as the value of the id
1153 attribute of the object in the registry. If the id is not a valid URN then the registry MUST return an
1154 InvalidRequestException. If the id conflicts with the id of an existing RegistryObject within the home
1155 registry then the registry MUST return InvalidRequestException for an UpdateObjectsRequest and treat
1156 it as an Update action for a SubmitObjectsRequest.

1157 If the client does not supply an id for a submitted object then the registry MUST generate a universally
1158 unique id. A registry generated id value MUST conform to the format of a URN that specifies a DCE 128
1159 bit UUID as specified in [UUID]:

1160 (e.g. *urn:uuid:a2345678-1234-1234-123456789012*).

1161 **5.1.3 ID Attribute And Object References**

1162 The id attribute of an object MAY be used by other objects to reference that object. Within a
1163 SubmitObjectsRequest, the id attribute MAY be used to refer to an object within the same
1164 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the
1165 SubmitObjectsRequest that needs to be referred to within the request document MAY be assigned an
1166 id by the submitter so that it can be referenced within the request. The submitter MAY give the object a
1167 valid URN, in which case the id is permanently assigned to the object within the registry. Alternatively,
1168 the submitter MAY assign an arbitrary id that is not a valid URN as long as the id is a unique anyURI
1169 value within the request document. In this case the id serves as a linkage mechanism within the
1170 request document but MUST be replaced with a registry generated id upon submission.

1171 When an object in a SubmitObjectsRequest needs to reference an object that is already in the registry,
1172 the request MAY contain an ObjectRef whose id attribute is the id of the object in the registry. This id is
1173 by definition a valid URN. An ObjectRef MAY be viewed as a proxy within the request for an object that
1174 is in the registry.

1175 **5.1.4 Audit Trail**

1176 The registry MUST create a single AuditableEvent object with eventType *Created* for all the
1177 RegistryObjects created by a SubmitObjectsRequest.

1178 **5.1.5 Sample SubmitObjectsRequest**

1179 The following example shows a simple SubmitObjectsRequest that submits a single Organization
1180 object to the registry. It does not show the complete SOAP Message with the message header and
1181 additional payloads in the message for the repository items.

1182

```
1183 <lcm:SubmitObjectsRequest>  
1184   <rim:RegistryObjectList>  
1185     <rim:Organization lid="{LOGICAL_ID}"  
1186       id="{ID}"  
1187       primaryContact="{CONTACT_USER_ID}">  
1188       <rim:Name>  
1189         <rim:LocalizedString value="Sun Microsystems Inc."  
1190 xml:lang="en-US"/>  
1191       </rim:Name>  
1192       <rim:Address city="Burlington" country="USA" postalCode="01867"  
1193 stateOrProvince="MA" street="Network Dr." streetNumber="1"/>  
1194       <rim:TelephoneNumber areaCode="781" countryCode="1" number="123-  
1195 456" phoneType="office"/>  
1196     </rim:Organization>  
1197   </rim:RegistryObjectList>  
1198 </SubmitObjectsRequest>
```

1199 **5.2 The Update Objects Protocol**

1200 The UpdateObjectsRequest protocol allows a Registry Client to update one or more existing
1201 RegistryObjects and/or repository items in the registry.

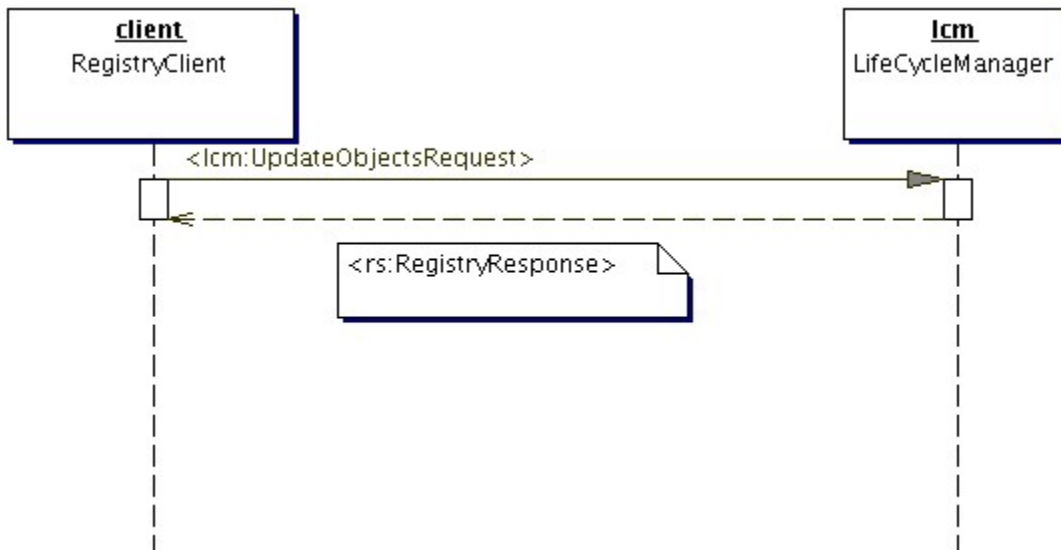


Figure 6: Update Objects Protocol

1203

1204 5.2.1 UpdateObjectsRequest

1205 The UpdateObjectsRequest is used by a client to update RegistryObjects and/or repository items that
1206 already exist within the registry.

1207 5.2.1.1 Syntax:

```
1208 <element name="UpdateObjectsRequest">  
1209   <complexType>  
1210     <complexContent>  
1211       <extension base="rs:RegistryRequestType">  
1212         <sequence>  
1213           <element ref="rim:RegistryObjectList"/>  
1214         </sequence>  
1215       </extension>  
1216     </complexContent>  
1217   </complexType>  
1218 </element>
```

1219 5.2.1.2 Parameters:

1220 *RegistryObjectList*: This parameter specifies a collection of RegistryObject instances
1221 that are being updated within the registry. All immediate RegistryObject children of the
1222 RegistryObjectList MUST be current RegistryObjects already in the registry.
1223 RegistryObjects MUST include all required attributes, even those the user does not
1224 intend to change. A missing attribute MUST be interpreted as a request to set that
1225 attribute to NULL or in case it has a default value, the default value will be assumed. If
1226 this collection contains an immediate child RegistryObject that does not already exist
1227 in the registry, then the registry MUST return an InvalidRequestException. If the user
1228 wishes to submit a mix of new and updated objects then he or she SHOULD use a
1229 SubmitObjectsRequest.
1230 If an ExtrinsicObject is being updated and no RepositoryItem is provided in the
1231 UpdateObjectsRequest then the registry MUST maintain any previously existing
1232 RepositoryItem associated with the original ExtrinsicObject with the updated
1233 ExtrinsicObject. If the client wishes to remove the RepositoryItem from an existing
1234 ExtrinsicObject they MUST use a RemoveObjectsRequest with
1235 deletionScope=DeleteRepositoryItemOnly.
1236

1237 5.2.1.3 Returns:

1238 This request returns a RegistryResponse. See section 2.1.4 for details.

1239 5.2.1.4 Exceptions:

1240 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1241 be returned:

1242 *UnresolvedReferenceException*: Indicates that the requestor referenced an object
1243 within the request that was not resolved during the processing of the request.

1244 *UnsignedRepositoryItemException*: Indicates that the requestor attempted to submit a
1245 RepositoryItem that was not signed.

1246 *QuotaExceededException*: Indicates that the requestor attempted to submit more
1247 content than the quota allowed for them by the registry.

1248 **5.2.2 Audit Trail**

1249 The registry MUST create a single AuditableEvent object with eventType *Updated* for all
1250 RegistryObjects updated via an UpdateObjectsRequest.

1251 **5.3 The Approve Objects Protocol**

1252 The Approve Objects protocol allows a client to approve one or more previously submitted
1253 RegistryObject objects using the LifeCycleManager service interface.

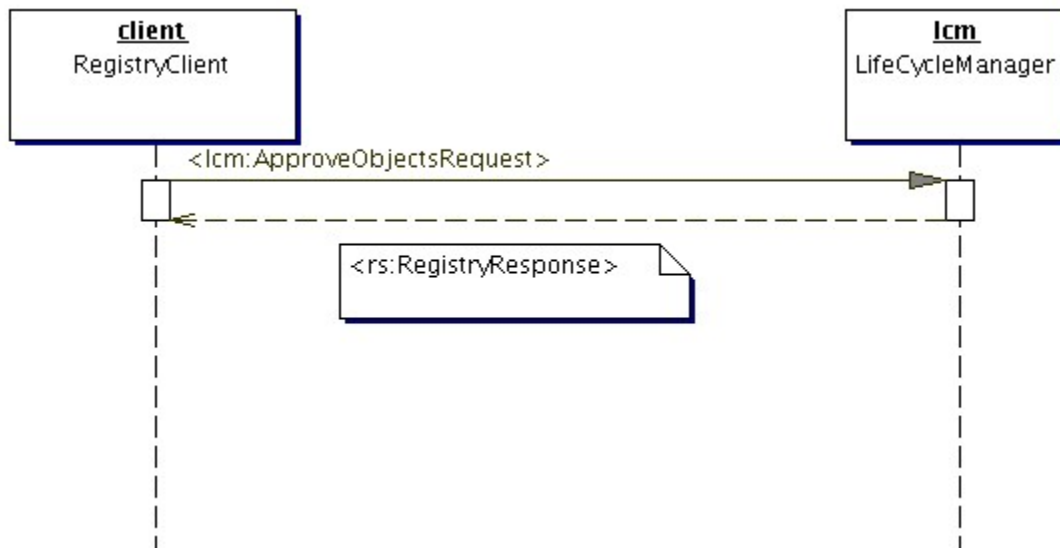


Figure 7: Approve Objects Protocol

1255 **5.3.1 ApproveObjectsRequest**

1256 The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject
1257 instances in the registry.

1258 **5.3.1.1 Syntax:**

```
1259 <element name="ApproveObjectsRequest">  
1260 <complexType>  
1261 <complexContent>  
1262 <extension base="rs:RegistryRequestType">  
1263 <sequence>  
1264 <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />  
1265 />  
1266 <element ref="rim:ObjectRefList" minOccurs="0" />  
1267 maxOccurs="1" />  
1268 </sequence>  
1269 </extension>  
1270 </complexContent>  
1271 </complexType>  
1272 </element>
```

1273 **5.3.1.2 Parameters:**

1274 **AdhocQuery:** This parameter specifies a query. A registry MUST approve all objects
1275 that match the specified query in addition to any other objects identified by other

1276 parameters.
1277 **ObjectRefList:** This parameter specifies a collection of references to existing
1278 RegistryObject instances in the registry. A registry MUST approve all objects that are
1279 referenced by this parameter in addition to any other objects identified by other
1280 parameters.

1281 5.3.1.3 Returns:

1282 This request returns a RegistryResponse. See section 2.1.4 for details.

1283 5.3.1.4 Exceptions:

1284 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1285 be returned:

1286 *ObjectNotFoundException:* Indicates that the requestor requested an object within the
1287 request that was not found.

1288

1289 5.3.2 Audit Trail

1290 The registry MUST create a single AuditableEvent object with eventType *Approved* for all
1291 RegistryObject instance approved via an ApproveObjectsRequest.

1292 5.4 The Deprecate Objects Protocol

1293 The Deprecate Object protocol allows a client to deprecate one or more previously submitted
1294 RegistryObject instances using the LifeCycleManager service interface. Once a RegistryObject is
1295 deprecated, no new references (e.g. new Associations, Classifications and ExternalLinks) to that object
1296 can be submitted. However, existing references to a deprecated object continue to function normally.

1297

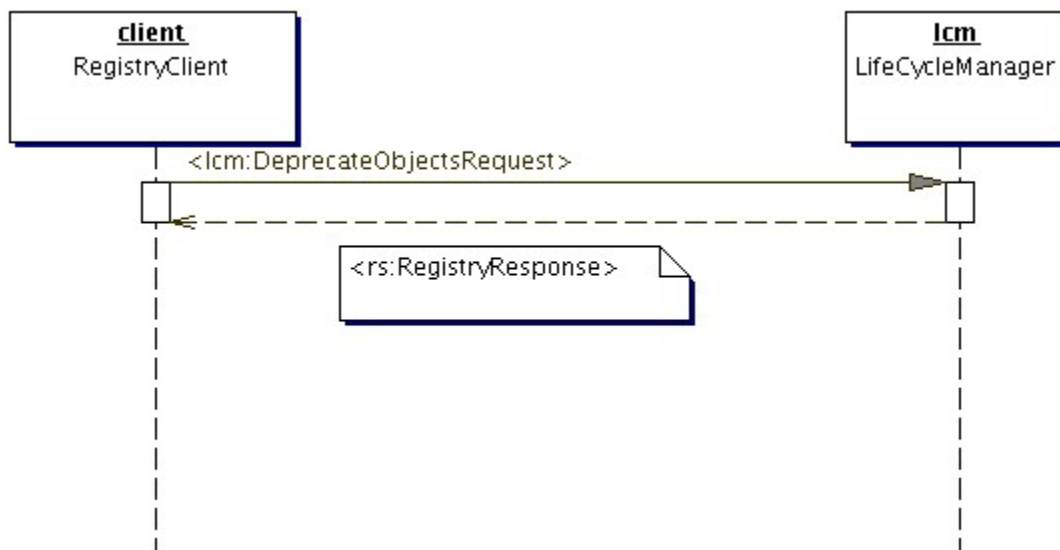


Figure 8: Deprecate Objects Protocol

1298 5.4.1 DeprecateObjectsRequest

1299 The DeprecateObjectsRequest is used by a client to deprecate one or more existing RegistryObject
1300 instances in the registry.

1301 5.4.1.1 Syntax:

```
1302 <element name="DeprecateObjectsRequest">
1303   <complexType>
1304     <complexContent>
1305       <extension base="rs:RegistryRequestType">
1306         <sequence>
1307           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
1308         />
1309           <element ref="rim:ObjectRefList" minOccurs="0"
1310 maxOccurs="1" />
1311         </sequence>
1312       </extension>
1313     </complexContent>
1314   </complexType>
1315 </element>
```

1316 5.4.1.2 Parameters:

1317 **AdhocQuery:** This parameter specifies a query. A registry MUST deprecate all objects
1318 that match the specified query in addition to any other objects identified by other
1319 parameters.

1320 **ObjectRefList:** This parameter specifies a collection of references to existing
1321 RegistryObject instances in the registry. A registry MUST deprecate all objects that are
1322 referenced by this parameter in addition to any other objects identified by other
1323 parameters.

1324 5.4.1.3 Returns:

1325 This request returns a RegistryResponse. See section 2.1.4 for details.

1326 5.4.1.4 Exceptions:

1327 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1328 be returned:

1329 **UnresolvedReferenceException:** Indicates that the requestor referenced an object
1330 within the request that was not resolved during the processing of the request.

1331 5.4.2 Audit Trail

1332 The registry MUST create a single AuditableEvent object with eventType *Deprecated* for all
1333 RegistryObject deprecated via a DeprecateObjectsRequest.

1334 5.5 The Undeprecate Objects Protocol

1335 The Undeprecate Objects protocol of the LifecycleManager service interface allows a client to undo the
1336 deprecation of one or more previously deprecated RegistryObject instances. When a RegistryObject is
1337 undeprecated, it goes back to the Submitted status and new references (e.g. new Associations,
1338 Classifications and ExternalLinks) to that object can now again be submitted.

1339

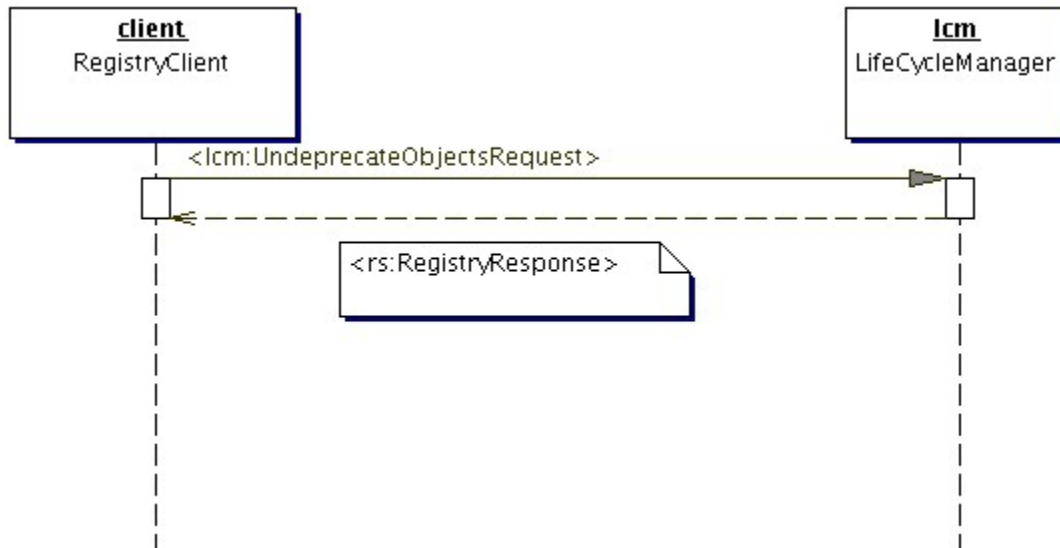


Figure 9: Undeprecate Objects Protocol

1340 5.5.1 UndeprecateObjectsRequest

1341 The UndeprecateObjectsRequest is used by a client to undeprecate one or more existing
 1342 RegistryObject instances in the registry. The registry MUST silently ignore any attempts to undeprecate
 1343 a RegistryObject that is not deprecated.

1344 5.5.1.1 Syntax:

```

1345 <element name="UndeprecateObjectsRequest">
1346   <complexType>
1347     <complexContent>
1348       <extension base="rs:RegistryRequestType">
1349         <sequence>
1350           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1"
1351 />
1352           <element ref="rim:ObjectRefList" minOccurs="0"
1353 maxOccurs="1" />
1354         </sequence>
1355       </extension>
1356     </complexContent>
1357   </complexType>
1358 </element>
1359 </element>
  
```

1360 5.5.1.2 Parameters:

1361 **AdhocQuery:** This parameter specifies a query. A registry MUST undeprecate all
 1362 objects that match the specified query in addition to any other objects identified by
 1363 other parameters.

1364 **ObjectRefList:** This parameter specifies a collection of references to existing
 1365 RegistryObject instances in the registry. A registry MUST undeprecate all objects that
 1366 are referenced by this parameter in addition to any other objects identified by other
 1367 parameters.

1368 **5.5.1.3 Returns:**

1369 This request returns a RegistryResponse. See section 2.1.4 for details.

1370 **5.5.1.4 Exceptions:**

1371 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1372 be returned:

1373 *UnresolvedReferenceException*: Indicates that the requestor referenced an object
1374 within the request that was not resolved during the processing of the request.

1375 **5.5.2 Audit Trail**

1376 The Registry Service MUST create a single AuditableEvent object with eventType *Undeprecated* for all
1377 RegistryObjects undeprecated via an UndeprecateObjectsRequest.

1378 **5.6 The Remove Objects Protocol**

1379 The Remove Objects protocol allows a client to remove one or more RegistryObject instances and/or
1380 repository items using the LifeCycleManager service interface.

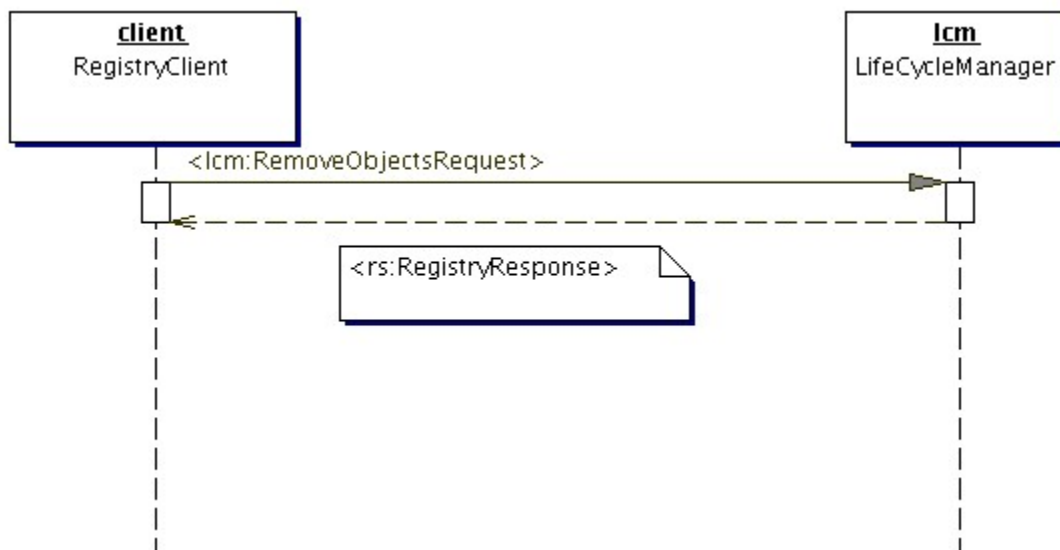


Figure 10: Remove Objects Protocol

1382 For details on the schema for the business documents shown in this process refer to .

1383 **5.6.1 RemoveObjectsRequest**

1384 The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject and/or
1385 repository items from the registry.

1386 **5.6.1.1 Syntax:**

```
1387 <element name="RemoveObjectsRequest">  
1388   <complexType>  
1389     <complexContent>  
1390       <extension base="rs:RegistryRequestType">  
1391         <sequence>  
1392           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />  
1393         />  
1394       />  
1395     />  
1396   />
```

```

1394         <element ref="rim:ObjectRefList" minOccurs="0"
1395 maxOccurs="1" />
1396     </sequence>
1397     <attribute name="deletionScope"
1398 default="urn:oasis:names:tc:ebxml-regrep:DeletionScopeType:DeleteAll"
1399 type="rim:referenceURI" use="optional"/>
1400 </extension>
1401 </complexContent>
1402 </complexType>
1403 </element>

```

1404 5.6.1.2 Parameters:

1405 **deletionScope:** This parameter indicates the scope of impact of the
1406 RemoveObjectsRequest. The value of the deletionScope attribute MUST be a
1407 reference to a ClassificationNode within the canonical DeletionScopeType
1408 ClassificationScheme as described in appendix A of [ebRIM]. A Registry MUST support
1409 the deletionScope types as defined by the canonical DeletionScopeType
1410 ClassificationScheme. The canonical DeletionScopeType ClassificationScheme may
1411 easily be extended by adding additional ClassificationNodes to it.

1412 The following canonical ClassificationNodes are defined for the DeletionScopeType
1413 ClassificationScheme:

1414 **DeleteRepositoryItemOnly:** This deletionScope specifies that the registry
1415 MUST delete the RepositoryItem for the specified ExtrinsicObjects but MUST
1416 NOT delete the specified ExtrinsicObjects. This is useful in keeping references
1417 to the ExtrinsicObjects valid. A registry MUST set the status of the
1418 ExtrinsicObject instance to *Withdrawn* in this case.

1419 **DeleteAll:** This deletionScope specifies that the request MUST delete both the
1420 RegistryObject and the RepositoryItem (if any) for the specified objects. A
1421 RegistryObject can be removed using a RemoveObjectsRequest with
1422 deletionScope DeleteAll only if all references (e.g. Associations,
1423 Classifications, ExternalLinks) to that RegistryObject have been removed.

1424 **AdhocQuery:** This parameter specifies a query. A registry MUST remove all objects
1425 that match the specified query in addition to any other objects identified by other
1426 parameters.

1427 **ObjectRefList:** *This parameter specifies a collection of references to existing*
1428 *RegistryObject instances in the registry.* A registry MUST remove all objects that are
1429 referenced by this parameter in addition to any other objects identified by other
1430 parameters.

1431 5.6.1.3 Returns:

1432 This request returns a RegistryResponse. See section 2.1.4 for details.

1433 5.6.1.4 Exceptions:

1434 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY
1435 be returned:

1436 **UnresolvedReferenceException:** Indicates that the requestor referenced an object
1437 within the request that was not resolved during the processing of the request.

1438 **ReferencesExistException:** Indicates that the requestor attempted to remove a
1439 RegistryObject while references to it still exist. Note that it is valid to remove a
1440 RegistryObject and all RegistryObjects that refer to it within the same request. In such
1441 cases the ReferencesExistException MUST not be thrown.

1442 **5.7 Registry Managed Version Control**

1443 This section describes the version control features of the ebXML Registry. This feature is based upon
1444 [DeltaV]. The ebXML Registry provides a simplified façade that provides a small subset of [DeltaV]
1445 functionality.

1446 **5.7.1 Version Controlled Resources**

1447 All repository items in an ebXML Registry are implicitly version-controlled resources as defined by
1448 section 2.2.1 of [DeltaV]. No explicit action is required to make them a version-controlled resource.

1449 In addition RegistryObject instances are also implicitly version-controlled resources. However, a
1450 registry may limit version-controlled resources to a sub-set of RegistryObject classes based upon
1451 registry specific policies.

1452 Minimally, a registry implementing the version control feature SHOULD make the following types as
1453 version-controlled resources:

1454 ClassificationNode

1455 ClassificationScheme

1456 Organization

1457 ExtrinsicObject

1458 RegistryPackage

1459 Service

1460 The above list is chosen to exclude all composed types and include most of remaining RegistryObject
1461 types for which there are known use cases requiring versioning.

1462 **5.7.2 Versioning and Object Identification**

1463 Each version of a RegistryObject is a unique object and as such has its own unique value for its id
1464 attribute as defined by [ebRIM].

1465 **5.7.3 Logical ID**

1466 All versions of a RegistryObject are logically the same object and are referred to as the `logical`
1467 RegistryObject. A logical RegistryObject is a tree structure where nodes are specific versions of the
1468 RegistryObject.

1469 A specific version of a logical RegistryObject is referred to as a `RegistryObject instance`.

1470 A RegistryObject instance MUST have a *Logical ID (LID)* to identify its membership in a particular
1471 logical RegistryObject. Note that this is in contrast with the `id` attribute that MUST be unique for each
1472 version of the same logical RegistryObject. A client may refer to the logical RegistryObject in a version
1473 independent manner using its LID.

1474 A RegistryObject is assigned a LID using the `lid` attribute of the RegistryObject class. If the submitter
1475 assigns the `lid` attribute, she must guarantee that it is a globally unique URN. A registry MUST honor a
1476 valid submitter-supplied LID. If the submitter does not specify a LID then the registry MUST assign a
1477 LID and the value of the LID attribute MUST be identical to the value of the `id` attribute of the first
1478 (originally created) version of the logical RegistryObject.

1479 **5.7.4 Version Identification**

1480 An ebXML Registry supports independent versioning of both RegistryObject metadata as well as
1481 repository item content. It is therefore necessary to keep distinct version information for a
1482 RegistryObject instance and its repository item if it happens to be an ExtrinsicObject instance.

1483 **5.7.4.1 Version Identification for a RegistryObject**

1484 A RegistryObject MUST have a versionInfo attribute whose type is the VersionInfo class defined by
 1485 ebRIM. The versionInfo attributes identifies the version information for that RegistryObject instance. A
 1486 registry MUST not allow two versions of the same RegistryObject to have the same
 1487 versionInfo.versionName attribute value.

1488 **5.7.4.2 Version Identification for a RepositoryItem**

1489 When a RegistryObject is an ExtrinsicObject with an associated repository item, the version
 1490 identification for the repository item is distinct from the version identification for the ExtrinsicObject.

1491 An ExtrinsicObject that has an associated repository item MUST have a contentVersionInfo attribute
 1492 whose type is the VersionInfo class defined by ebRIM. The contentVersionInfo attributes identifies the
 1493 version information for that repository item instance.

1494 An ExtrinsicObject that does not have an associated repository item MUST NOT have a
 1495 contentVersionInfo attribute defined.

1496 A registry MUST allow two versions of the same ExtrinsicObject to have the same
 1497 contentVersionInfo.versionName attribute value because multiple ExtrinsicObject versions MAY share
 1498 the same RepositoryItem version.

1499 **5.7.5 Versioning of ExtrinsicObject and Repository Items**

1500 An ExtrinsicObject and its associated repository item may be updated independently and therefore
 1501 versioned independently.

1502 A registry MUST maintain separate version trees for an ExtrinsicObject and its associated repository
 1503 item as described earlier.

1504 Table 6 shows all the combinations for versioning an ExtrinsicObject and its repository item. After
 1505 eliminating invalid or impossible combinations as well as those combinations where no action is
 1506 needed, the only combinations that require versioning are showed in gray background rows. Of these
 1507 there are only two unique cases (referred to as case A and B). Note that it is not possible to version a
 1508 repository item without versioning its ExtrinsicObject.

1509

ExtrinsicObject Exists	RepositoryItem Exists	ExtrinsicObject Updated	RepositoryItem Updated	Comment
No	No			Do nothing
No	Yes			Not possible
Yes	No	No	No	Do nothing
		No	Yes	Not possible
		Yes	No	Version ExtrinsicObject (case A)
		Yes	Yes	Not possible

Yes	Yes	No	No	Do nothing
		No	Yes	Not possible
		Yes	No	Version ExtrinsicObject (case A)
		Yes	Yes	Version ExtrinsicObject and RepositoryItem (case B)

Table 6: Versioning of ExtrinsicObject and Repository Item

1510

1511 5.7.5.1 ExtrinsicObject and Shared RepositoryItem

1512 Because an ExtrinsicObject and its repository item are versioned independently (case B) it is possible
 1513 for multiple versions of the ExtrinsicObject to share the same version of the repository item. In such
 1514 cases the contentVersionInfo attributes MUST be the same across multiple version of the
 1515 ExtrinsicObject.

1516 5.7.6 Versioning and Composed Objects

1517 When a registry creates a new version of a RegistryObject it MUST create copies of all composed¹
 1518 objects as new objects that are composed within the new version. This is because each version is a
 1519 unique object and composed objects by definition are not shareable across multiple objects.
 1520 Specifically, each new copy of a composed object MUST have a new id since it is a different object
 1521 than the original composed object in the previous version.

1522 A registry MUST not version composed objects.

1523 5.7.7 Versioning and References

1524 An object reference from a RegistryObject references a specific version of the referenced
 1525 RegistryObject. When a registry creates a new version of a referenced RegistryObject it MUST NOT
 1526 move references from other objects from the previous version to the new version of the referenced
 1527 object. Clients that wish to always reference the latest version of an object MAY use the Event
 1528 Notification feature to update references when new versions are created and thus always reference the
 1529 latest version.

1530 A special case is when a SubmitObjectsRequest or an UpdateObjectRequest contains an object that is
 1531 being versioned by the registry and the request contains other objects that reference the object being
 1532 versioned. In such case, the registry MUST update all references within the submitted objects to the
 1533 object being versioned such that those objects now reference the new version of the object being
 1534 created by the request.

1535 5.7.8 Versioning and Audit Trail

1536 The canonical EventType ClassificationScheme used by the Audit Trail feature defines an Updated
 1537 event type and then defines a Versioned event type as a child of the Updated event type
 1538 ClassificationNode. The semantic are that a Versioned event type is specialization of the Updated
 1539 event type.

1540 A registry MUST use the Updated event type in the AuditableEvent when it updates a RegistryObject

¹ Composed object types are identified in figure 1 in [eBRIM] figure 1 as classes with composition or “solid diamond” relationship with RegistryObject type.

1541 without creating a new version.

1542 A registry MUST use the Versioned event type in the AuditableEvent when it creates a new version of
1543 a logical RegistryObject.

1544 A registry MUST NOT use the Created event type in the AuditableEvent when it creates a new version
1545 of a logical RegistryObject.

1546 **5.7.9 Inter-versions Association**

1547 Within any single branch within the version tree for an object any given version implicitly supersedes
1548 the version immediately prior to it. Sometimes it may be necessary to explicitly indicate which version
1549 supersedes another version for the same object. This is especially true when two versions are siblings
1550 branch roots of the version tree for the same object.

1551 A client MAY specify an Association between any two versions of an object within the objects version
1552 tree using the canonical associationType "Supersedes" to indicate that the sourceObject supersedes
1553 the target targetObject within the Association.

1554 A client MUST NOT specify an Association between two version of an object using the canonical
1555 associationType "Supersedes" if the sourceObject is an earlier version within the same branch in the
1556 version tree than the targetObject as this violates the implicit "Supersedes" association between the
1557 two version.

1558 Note that this section is functionally equivalent to the predecessor-set successor-set elements of the
1559 Version Properties as defined by [DeltaV].

1560 **5.7.10 Client Initiated Version Removal**

1561 An ebXML Registry MAY allow clients to remove specified versions of a RegistryObject. A client MAY
1562 delete older version of an object using the RemoveObjectsRequest by specifying the version by its
1563 unique id. Removing an ExtrinsicObject instance MUST remove its repository item if no other version
1564 references that repository item.

1565 **5.7.11 Registry Initiated Version Removal**

1566 The registry MAY prune older versions based upon registry specific administrative policies in order to
1567 manage storage resources.

1568 **5.7.12 Locking and Concurrent Modifications**

1569 This specification does not define a workspace feature with explicit checkin and checkout capabilities
1570 as defined by [DeltaV]. An ebXML Registry MAY support such features in an implementation specific
1571 manner.

1572 This specification does not prescribe a locking or branching model. An implementation may choose to
1573 support an optimistic (non-locking) model. Alternatively or in addition, an implementation may support a
1574 locking model that supports explicit checkout and checkin capability. A future technical note or
1575 specification may address some of these capabilities.

1576 **5.7.13 Version Creation**

1577 The registry manages creation of new version of a RegistryObject or a repository item automatically. A
1578 registry that supports versioning MUST implicitly create a new version for a repository item if the
1579 repository item is updated via a SubmitObjectsRequest or UpdateObjectsRequest. In such cases it
1580 MUST also create a new version of its ExtrinsicObject.

1581 If the client only wishes to update and version the ExtrinsicObject it may do so using an
1582 UpdateObjectsRequest without providing a repository item. In such cases the registry MUST assign the
1583 repository item version associated with the previous version of the ExtrinsicObject.

1584 **5.7.14 Versioning Override**

1585 A client MAY specify a *dontVersion* hint on a per RegistryObject basis when doing a submit or update
1586 of a RegistryObject. A registry SHOULD not create a new version for that RegistryObject when the
1587 dontVersion hint has value of "true". The dontVersion hint MAY be specified as a canonical Slot with
1588 the following name:

1589
1590 `urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersion`

1592 The value of the dontVersion Slot, if specified, MUST be either "true" or "false".

1593 A client MAY specify a *dontVersionContent* hint on a per ExtrinsicObject basis when doing a submit or
1594 update of an ExtrinsicObject with a repository item. A registry SHOULD not create a new version for
1595 that repository item when the dontVersionContent hint has value of "true". The dontVersionContent hint
1596 MAY be specified as a canonical Slot with the following name:

1597
1598 `urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersionContent`

1600 The value of the dontVersionContent Slot, if specified, MUST be either "true" or "false".

1601 A client MAY also specify the dontVersion and dontVersionContent Slots on the RegistryRequest using
1602 the `<rs:ReqstSlotList>` element. A registry MUST treat these Slots when specified on the request as
1603 equivalent to being specified on every RegistryObject within the request. The value of these Slots as
1604 specified on the request take precedence over value of these Slots as specified on RegistryObjects
1605 within the request.

6 Query Management Protocols

1606

1607 This section defines the protocols supported by QueryManager service interface of the Registry. The
1608 Query Management protocols provide the functionality required by RegistryClients to query the registry
1609 and discover RegistryObjects and RepositoryItems.

1610 The XML schema for the Query Management protocols is described in [RR-QUERY-XSD].

6.1 Ad Hoc Query Protocol

1611

1612 The Ad hoc Query protocol of the QueryManager service interface allows a client to query the registry
1613 and retrieve RegistryObjects and/or RepositoryItems that match the specified query.

1614 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
1615 AdhocQueryRequest contains a sub-element that specifies a query in one of the query syntaxes
1616 supported by the registry.

1617 The QueryManager sends an AdhocQueryResponse back to the client as response. The
1618 AdhocQueryResponse returns a collection of objects that match the query. The collection is potentially
1619 heterogeneous depending upon the query expression and request options.

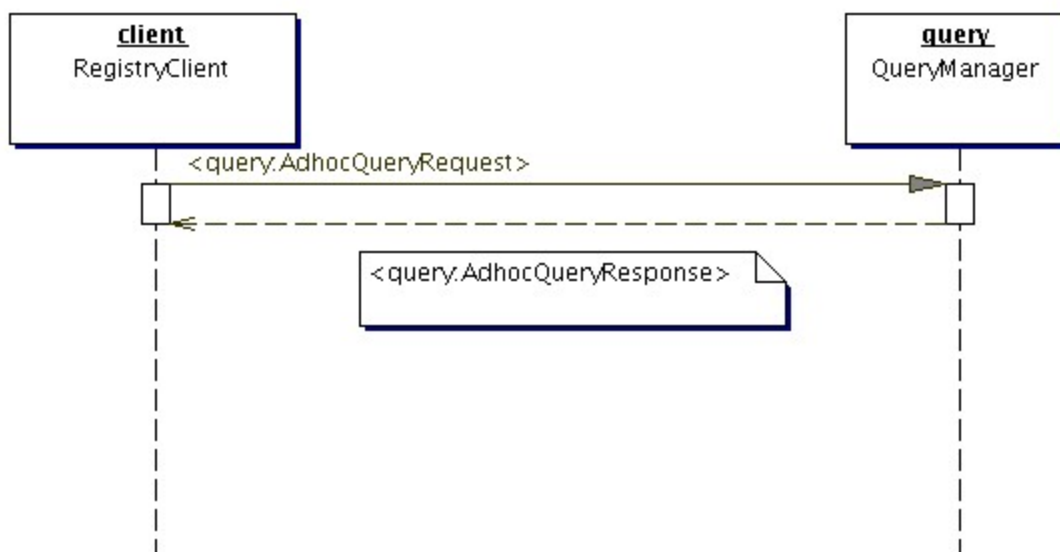


Figure 11: Ad Hoc Query Protocol

6.1.1 AdhocQueryRequest

1620

1621 The AdhocQueryRequest is used to submit a query to the registry.

6.1.1.1 Syntax:

1622

```
1623 <element name="AdhocQueryRequest">  
1624   <complexType>  
1625     <complexContent>  
1626       <extension base="rs:RegistryRequestType">  
1627         <sequence>  
1628           <element maxOccurs="1" minOccurs="1"  
1629             ref="tns:ResponseOption"/>  
1630           <element ref="rim:AdhocQuery" />  
1631         </sequence>  
1632         <attribute default="false" name="federated"  
1633           type="boolean" use="optional"/>  
1634       </extension>  
1635     </complexContent>  
1636   </complexType>  
1637 </element>
```

1634
1635
1636
1637
1638
1639
1640

```
<attribute name="federation" type="anyURI" use="optional"/>  
<attribute default="0" name="startIndex" type="integer"/>  
<attribute default="-1" name="maxResults" type="integer"/>  
</extension>  
</complexContent>  
</complexType>  
</element>
```

1641 6.1.1.2 Parameters:

1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665

AdhocQuery: This parameter specifies the actual query. It is described in detail in section 6.1.3.

federated: This optional parameter specifies that the registry must process this query as a federated query. By default its value is *false*. This value **MUST** be false when a registry routes a federated query to another registry in order to avoid an infinite loop in federated query processing.

federation: This optional parameter specifies the id of the target Federation for a federated query in case the registry is a member of multiple federations. In the absence of this parameter a registry must route the federated query to all federations of which it is a member. This value **MUST** be unspecified when a registry routes a federated query to another registry in order to avoid an infinite loop in federated query processing.

maxResults: This optional parameter specifies a limit on the maximum number of results the client wishes the query to return. If unspecified, the registry **SHOULD** return either all the results, or in case the result set size exceeds a registry specific limit, the registry **SHOULD** return a sub-set of results that are within the bounds of the registry specific limit. See section 6.2.1 for an illustrative example.

ResponseOption: This required parameter allows the client to control the format and content of the AdhocQueryResponse generated by the registry in response to this request. See section 6.1.4 for details.

startIndex: This optional integer value is used to indicate which result *must* be returned as the first result when iterating over a large result set. The default value is 0, which returns the result set starting with index 0 (first result). See section 6.2.1 for an illustrative example.

1666 6.1.1.3 Returns:

1667 This request returns an AdhocQueryResponse. See section 6.1.2 for details.

1668 6.1.1.4 Exceptions:

1669 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions **MAY**
1670 be returned:

1671 *InvalidQueryException:* signifies that the query syntax or semantics was invalid. Client
1672 must fix the query syntax or semantic error and re-submit the query.

1673 6.1.2 AdhocQueryResponse

1674 The AdhocQueryResponse is sent by the registry as a response to an AdhocQueryRequest.

1675 6.1.2.1 Syntax:

1676
1677

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

```

1678     <complexContent>
1679         <extension base="rs:RegistryResponseType">
1680             <sequence>
1681                 <element ref="rim:RegistryObjectList" />
1682             </sequence>
1683             <attribute default="0" name="startIndex" type="integer"/>
1684             <attribute name="totalResultCount" type="integer"
1685 use="optional"/>
1686         </extension>
1687     </complexContent>
1688 </complexType>
1689 </element>

```

1690 6.1.2.2 Parameters:

1691 **RegistryObjectList:** This is the element that contains the RegistryObject instances
1692 that matched the specified query.

1693 **startIndex:** This optional integer value is used to indicate the index for the first result
1694 in the result set returned by the query, within the complete result set matching the
1695 query. By default, this value is 0. See section 6.2.1 for an illustrative example.

1696 **totalResultCount:** This optional parameter specifies the size of the complete result
1697 set matching the query within the registry. When this value is unspecified, the client
1698 should assume it is the size of the result set contained within the result. See section
1699 6.2.1 for an illustrative example.

1700 6.1.3 AdhocQuery

1701 A client specifies a <rim:AdhocQuery> element within an AdhocQueryRequest to specify the actual
1702 query being submitted.

1703 6.1.3.1 Syntax:

```

1704 <complexType abstract="true" name="AdhocQueryType">
1705     <complexContent>
1706         <extension base="tns:RegistryObjectType">
1707             <sequence>
1708                 <element ref="tns:QueryExpression"
1709 minOccurs="0" maxOccurs="1" />
1710             </sequence>
1711         </extension>
1712     </complexContent>
1713 </complexType>
1714 <element name="AdhocQuery" type="tns:AdhocQueryType"
1715 substitutionGroup="tns:RegistryObject" />

```

1716

1717 6.1.3.2 Parameters:

1718 **queryExpression:** This element contains the actual query expression. The schema for
1719 queryExpression is extensible and can support any query syntax supported by the
1720 registry.

1721 6.1.4 ReponseOption

1722 A client specifies a ResponseOption structure within an AdhocQueryRequest to indicate the format of
1723 the results within the corresponding AdhocQueryResponse.

1724

1725 6.1.4.1 Syntax:

```
1726 <complexType name="ResponseOptionType">
1727   <attribute default="RegistryObject" name="returnType">
1728     <simpleType>
1729       <restriction base="NCName">
1730         <enumeration value="ObjectRef"/>
1731         <enumeration value="RegistryObject"/>
1732         <enumeration value="LeafClass"/>
1733         <enumeration value="LeafClassWithRepositoryItem"/>
1734       </restriction>
1735     </simpleType>
1736   </attribute>
1737   <attribute default="false" name="returnComposedObjects"
1738     type="boolean"/>
1739 </complexType>
1740 <element name="ResponseOption" type="tns:ResponseOptionType"/>
```

1741

1742 6.1.4.2 Parameters:

1743 **returnComposedObjects:** This optional parameter specifies whether the
1744 RegistryObjects returned should include composed objects as defined by Figure 1 in
1745 [ebRIM]. The default is to return all composed objects.

1746 **returnType:** This optional enumeration parameter specifies the type of RegistryObject
1747 to return within the response. Values for returnType are as follows:

- 1748 • **ObjectRef** - This option specifies that the AdhocQueryResponse MUST
1749 contain a collection of <rim:ObjectRef> elements. The purpose of this option
1750 is to return references to registry objects rather than the actual objects.
- 1751 • **RegistryObject** - This option specifies that the AdhocQueryResponse
1752 MUST contain a collection of <rim:RegistryObject> elements.
- 1753 • **LeafClass** - This option specifies that the AdhocQueryResponse MUST
1754 contain a collection of elements that correspond to leaf classes as defined in
1755 [RR-RIM-XSD].
- 1756 • **LeafClassWithRepositoryItem** - This option is same as LeafClass option
1757 with the additional requirement that the response include the
1758 RepositoryItems, if any, for every <rim:ExtrinsicObject> element in the
1759 response.

1760 If “returnType” specified does not match a result returned by the query, then the
1761 registry *must* use the closest matching semantically valid returnType that matches the
1762 result.

1763 To illustrate, consider a case where OrganizationQuery is asked to return
1764 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume
1765 LeafClass option instead.

1766

1767 6.2 Iterative Query Support

1768 The AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large result
1769 set matching a logical query by allowing multiple AdhocQueryRequest requests to be submitted such
1770 that each query requests a different subset of results within the result set. This feature enables the
1771 registry to handle queries that match a very large result set, in a scalable manner. The iterative query
1772 feature is accessed via the startIndex and maxResults parameters of the AdhocQueryRequest and the

1773 startIndex and totalResultCount parameters of the AdhocQueryResponse as described earlier.
 1774 The iterative queries feature is not a true Cursor capability as found in databases. The registry is not
 1775 required to maintain transactional consistency or state between iterations of a query. Thus it is possible
 1776 for new objects to be added or existing objects to be removed from the complete result set in between
 1777 iterations. As a consequence it is possible to have a result set element be skipped or duplicated
 1778 between iterations.
 1779 Note that while it is not required, an implementations MAY implement a transactionally consistent
 1780 iterative query feature.

1781 6.2.1 Query Iteration Example

1782 Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a query
 1783 that matches all 1007 Organizations. The user wishes to do the query iteratively such that
 1784 Organizations are retrieved in chunks of 100. The following table illustrates the parameters of the
 1785 AdhocQueryRequest and those of the AdhocQueryResponses for each iterative query in this example.
 1786

AdhocQueryRequest Parameters		AdhocQueryResponse Parameters		
startIndex	maxResults	startIndex	totalResultCount	# of Results
0	100	0	1007	100
100	100	100	1007	100
200	100	200	1007	100
300	100	300	1007	100
400	100	400	1007	100
500	100	500	1007	100
600	100	600	1007	100
700	100	700	1007	100
800	100	800	1007	100
900	100	900	1007	100
1000	100	1000	1007	7

1787

1788 6.3 Stored Query Support

1789 The AdhocQuery protocol allow clients to submit queries that may be as general or as specific as the
 1790 use case demands. As the queries get more specific they also get more complex. In these situations it
 1791 is desirable to hide the complexity of the query from the client using parameterized queries stored in
 1792 the registry. When using parameterized stored queries the client is only required to specify the identity
 1793 of the query and the parameters for the query rather than the query expression itself.

1794 Parameterized stored queries are useful to Registry Administrators because they provide a system
 1795 wide mechanism for the users of the registry to share a set of commonly used queries.

1796 Parameterized stored queries are useful to vertical standards because the standard can define domain
 1797 specific parameterized queries and require that they be stored within the registry.

1798 An ebXML Registry MUST support parameterized stored queries as defined by this section.

1799 6.3.1 Submitting a Stored Query

1800 A stored query is submitted using the standard SubmitObjectsRequest protocol where the object
 1801 submitted is an AdhocQueryType instance.

1802 6.3.1.1 Declaring Query Parameters

1803 When submitting a stored query, the submitter MAY declare zero or more parameters for that query. A
 1804 parameter MUST be declared using a parameter name that begins with the '\$' character followed

1805 immediately by a letter and then followed by any combination of letters and numbers. The following
1806 BNF defines how a parameter name MUST be declared.

1807

```
1808 QueryParameter := '$' [a-zA-Z] ( [a-zA-Z] | [0-9] )*
```

1809

1810 A query parameter MAY be used as a placeholder for any part of the stored query.

1811 The following example illustrates how a parameterized stored query may be submitted:

1812

```
1813 <SubmitObjectsRequest>  
1814   <rim:RegistryObjectList>  
1815     <rim:AdhocQuery id="{QUERY_ID}">  
1816       <rim:QueryExpression queryLanguage="{SQL_QUERY_LANG_ID}">  
1817         SELECT * from $tableName ro, Name_ nm, Description d  
1818         WHERE  
1819           objectType = '$objectType'  
1820           AND (nm.parent = ro.id AND UPPER ( nm.value ) LIKE UPPER  
1821 ( '$name' ) )  
1822           AND (d.parent = ro.id AND UPPER ( d.value ) LIKE UPPER  
1823 ( '$description' ) )  
1824           AND (ro.id IN ( SELECT classifiedObject FROM Classification  
1825 WHERE classificationNode IN ( SELECT id  
1826 FROM ClassificationNode WHERE path LIKE  
1827 '$classificationPath1%' ) ) )  
1828       </rim:QueryExpression>  
1829     </rim:AdhocQuery>  
1830   </rim:RegistryObjectList>  
1831 </SubmitObjectsRequest>
```

1832 Listing 1: Example of Stored Query Submission

1833

1834 The above query takes parameters *\$objectType*, *\$name*, *\$description* and *\$classificationPath1* and find
1835 all objects for that match specified objectType, name, description and classification.

1836 6.3.1.2 Canonical Context Parameters

1837 A query MAY contain one or more context parameters as defined in this section. Context parameters
1838 are special query parameters whose value does not need to be supplied by the client. Instead the
1839 value for a context parameter is supplied by the registry based upon the context within which the client
1840 request is being processed.

1841 When processing a query, a registry MUST replace all context parameters present in the query with
1842 the context sensitive value for the parameter. A registry MUST ignore any context parameter values
1843 supplied by the client.

1844

Context Parameter	Replacement Value
\$currentUser	Must be replaced with the id attribute of the user associated with the query.
\$currentTime	Must be replaced with the currentTime. The time format is same as the format defined for the timestamp attribute of AuditableEvent class.

1845

1846 6.3.2 Invoking a Stored Query

1847 A stored query is invoked using the AdhocQueryRequest with the following constraints:

- 1848 • The <rim:AdhocQuery> element MUST not contain a <rim:queryExpression> element.

- 1849 • The <rim:AdhocQuery> element's id attribute value MUST match the id attribute value of the stored
1850 query.
- 1851 • The <rim:AdhocQuery> element MAY have a Slot for each non-context parameter defined for the
1852 stored query being invoked. These Slots provide the value for the query parameters.

1853 **6.3.2.1 Specifying Query Invocation Parameters**

1854 A stored query MAY be defined with zero or more parameters. A client may specify zero or more of the
1855 parameters defined for the stored query when submitting the AdhocQueryRequest for the stored query.
1856 It is important to note that the client MAY specify fewer parameters than those declared for the stored
1857 query. A registry MUST prune any predicates of the stored query that contain parameters that were not
1858 supplied by the client during invocation of the stored query.

1859 In essence, the client may narrow or widen the specificity of the search by supplying more or less
1860 parameters.

1861 A client specifies a query invocation parameter by using a Slot whose name matches the parameter
1862 name and whose value MUST be a single value that matches the specified value for the parameter.

1863 A registry MUST ignore any parameters specified by the client for a stored query that do not match the
1864 parameters defined by the stored query.

1865 The following listing shows an example of how the stored query shown earlier is invoked. It shows:

- 1866 • The stored query being identified by the value of the id attribute of the <rim:AdhocQuery> element.
- 1867 • The value for the \$name parameter being supplied
- 1868 • The value of other parameters defined by the query not being supplied. This indicates that the client
1869 does not wish to use those parameters as search criteria.

1870

```
1871 <AdhocQueryRequest>  
1872   <query:ResponseOption returnComposedObjects="true"  
1873   returnType="LeafClassWithRepositoryItem"/>  
1874  
1875   <rim:AdhocQuery id="{STORED_QUERY_ID}">  
1876     <rim:Slot name="$name">  
1877       <rim:ValueList>  
1878         <rim:Value>%eBXML%</rim:Value>  
1879       </rim:ValueList>  
1880     </rim:Slot>  
1881   </rim:AdhocQuery>  
1882 </AdhocQueryRequest>
```

Listing 2: Example of Stored Query Invocation

1883 **6.3.3 Response to Stored Query Invocation**

1884 A registry MUST send a standard AdhocQueryResponse when a client invokes a stored query using an
1885 AdhocQueryRequest.

1886 **6.3.4 Access Control on a Stored Query**

1887 A stored query is a RegistryObject. Like all RegistryObjects, access to the stored query is governed by
1888 the Access Control Policy defined the stored query. By default a stored query is assigned the default
1889 Access Control Policy that allows any client to read and invoke that query and only the owner of the
1890 query and the Registry Administrator role to update or delete the query. The owner of the query may
1891 define a custom Access Control Policy for the query that restricts the visibility of the query, and ability
1892 to invoke it, to specific users, roles or groups. Thus the owner of the query or the Registry Administrator
1893 may control *who* gets to invoke *which* stored queries.

1894 **6.3.5 Canonical Query: Get Client's User Object**

1895 A registry MUST support a canonical stored query with

1896 `id="urn:oasis:names:tc:ebxml-regrep:query:GetCallersUser".`

1897 This query MUST return the User object associated with the client invoking the stored query. The client
1898 MUST not provide any parameters for this query. The stored query SHOULD use the canonical context
1899 parameter `$currentUser`.

1900 The following is a non-normative example of a stored SQL query that MAY be used by a registry for this
1901 canonical stored query:

1902

```
1903 <rim:AdhocQuery id="urn:oasis:names:tc:ebxml-  
1904 regrep:query:GetCallersUser">  
1905   <rim:QueryExpression  
1906     queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:SQL-  
1907     92">  
1908     SELECT u.* FROM User u WHERE u.id = $currentUser;  
1909   </rim:QueryExpression>  
1910 </rim:AdhocQuery>
```

1911 Note that a registry MAY use an equivalent stored filter query instead of a stored SQL query.

1912 **6.4 SQL Query Syntax**

1913 An ebXML Registry MAY support SQL as a supported query syntax within the `<rim:queryExpression>`
1914 element of `AdhocQueryRequest`. This section normatively defines the SQL syntax that an ebXML
1915 Registry MAY support. Note that the support for SQL syntax within a registry does not imply a
1916 requirement that the registry must use a relational database in its implementation.

1917 The registry SQL syntax is a proper subset of the "SELECT" statement of Intermediate level SQL as
1918 defined by ISO/IEC 9075:1992, Database Language SQL [SQL].

1919 The terms below enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query
1920 syntax conforms to the `<query specification>` with the following additional restrictions:

- 1921 1. A **<derived column>** MAY NOT have an **<as clause>**.
- 1922 2. A **<table expression>** does not contain the optional **<group by clause>** and **<having clause>**
1923 clauses.
- 1924 3. A **<table reference>** can only consist of **<table name>** and **<correlation name>**.
- 1925 4. A **<table reference>** does not have the optional AS between **<table name>** and **<correlation**
1926 **name>**.
- 1927 5. Restricted use of sub-queries is allowed by the syntax as follows. The **<in predicate>** allows for the
1928 right hand side of the **<in predicate>** to be limited to a restricted **<query specification>** as defined
1929 above.

1930 As defined by [SQL], a registry MUST process table names and attribute names in a case insensitive
1931 manner.

1932 **6.4.1 Relational Schema for SQL Queries**

1933 The normative Relational Schema definition that is the target of registry SQL queries can be found at
1934 the following location on the web:

1935 <http://www.oasis-open.org/committees/regrep/documents/3.0/sql/database.sql>

1936 **6.4.2 SQL Query Results**

1937 The result of an SQL query resolves to a collection of objects within the registry. It never resolves to
1938 partial attributes. The objects related to the result set may be returned as an `ObjectRef`, `RegistryObject`

1939 or leaf class depending upon the returnType attribute of the responseOption parameter specified by
1940 the client on the AdHocQueryRequest. The entire result set is returned as an <rim:RegistryObjectList>.

1941 **6.5 Filter Query Syntax**

1942 This section normatively defines an XML syntax for querying an ebXML Registry called *Filter Query*
1943 syntax. An ebXML Registry MUST support the Filter Query syntax as a supported query syntax within
1944 the <rim:queryExpression> element of AdhocQueryRequest.

1945 The Filter Query syntax is defined in [RR-QUERY-XSD] and is derived from a mapping from [ebRIM] to
1946 XML Schema following certain mapping patterns.

1947 The Filter Query operational model views the network of RegistryObjects in the registry as a virtual
1948 XML document and a query traverses a specified part of the tree and prunes or filters objects from the
1949 virtual document using filter expressions and ultimately returns a collection of objects that are left after
1950 filtering out all objects that do not match the filters specified in the query.

1951 Unlike SQL query syntax, the filter query syntax does not support joins across classes. This constrains
1952 the expressive capabilities of the query and may also be somewhat less efficient in processing.

1953 **6.5.1 Filter Query Structure**

1954 The <rim:queryExpression> element of AdhocQueryRequest MUST contain a *Query* element derived
1955 from the <query:RegistryObjectQueryType> type.

1956 A Query element MAY contain a <query:PrimaryFilter> element and MAY contain additional Filter,
1957 Branch and Query elements within it as shown in the abstract example below. The normative schema
1958 is defined by [RR-QUERY-XSD].

```
1959  
1960 <${QueryElement}>  
1961   <PrimaryFilter ... />  
1962   <${OtherFilterElement} ... />  
1963   <${BranchElement} ... />  
1964   <${QueryElement} ... />  
1965 </${QueryElement}>
```

1966

1967 The role of Query, Filter and Branch elements will be defined next.

1968 **6.5.2 Query Elements**

1969 A Query element is the top level element in the Filter Query syntax to query the registry. The [RR-
1970 QUERY-XSD] XML Schema defines a Query element for the RegistryObject class and all its
1971 descendant classes as defined by [ebRIM] using the following pattern:

- 1972 • For each class in model descendant from RegistryObject class define a complexType with name
1973 <class>QueryType. For example there is an OrganizationQueryType complexType defined for the
1974 Organization class in [ebRIM].
- 1975 • The QueryType of a descendant of RegistryObject class MUST extend the QueryType for its super
1976 class. For example the OrganizationQueryType extends the RegistryObjectQueryType.
- 1977 • For RegistryObject class and each of its descendants define an element with name <class>Query
1978 and with type <class>QueryType. For example the OrganizationQuery element is defined with type
1979 OrganizationQueryType.

1980 The class associated with a Query element is referred to as the *Query domain class*.

1981 The following example shows the Query syntax where the Query domain class is the Organization
1982 class defined by [ebRIM]:

1983

```
1984 <complexType name="OrganizationQueryType">  
1985   <complexContent>
```

```

1986     <extension base="tns:RegistryObjectQueryType">
1987         ...Relevant Filters, Queries and Branches are defined here...
1988     </extension>
1989     </complexContent>
1990 </complexType>
1991 <element name="OrganizationQuery" type="tns:OrganizationQueryType"/>

```

1992

1993 A Query element MAY have Filter, Branch or nested Query Elements. These are described in
 1994 subsequent sections.

1995 6.5.3 Filter Elements

1996 A Query element MAY contain one or more Filter sub-elements. A Filter element is used to *filter* or
 1997 select a subset of instances of a specific [ebRIM] class. The class that a Filter filters is referred to as
 1998 the *Filter domain class*. A Filter element specifies a restricted predicate clause over the attributes of the
 1999 Filter domain class.

2000 [RR-QUERY-XSD] XML Schema defines zero or more Filter elements within a Query element definition
 2001 using the following pattern:

- 2002 • **PrimaryFilter:** A Filter element is defined within the RegistryObjectQueryType with name
 2003 *PrimaryFilter*. This Filter is used to filter the instances of the Query domain class based upon the
 2004 value of its primitive attributes. The cardinality of the Filter element is zero or one. The *PrimaryFilter*
 2005 element is inherited by all descendant QueryTypes of RegistryObjectQueryType.
- 2006 • **Additional Filters:** Additional Filters in a Query element used to filter the instances of the Query
 2007 domain class based upon whether the candidate domain class instance has a referenced object that
 2008 satisfies the additional filter.

2009 Additional filter elements are defined for those attributes of the Query domain class that satisfy all of
 2010 the following criteria:

- 2011 • The attribute's domain is not a primitive type (e.g. string, float, dateTime, int etc.).
- 2012 • The attribute's domain class is not RegistryObject or its descendant.
- 2013 • The attribute's domain class does not have any reference attributes (use Branch or sub-Query if
 2014 attribute's domain class has reference attributes).

2015 The attribute for which the Filter is defined is referred to as the Filter domain attribute. The
 2016 domain class of the Filter domain attribute is the Filter domain class for such Filters. This type of
 2017 Filter is used to filter the instances of the Query domain class based upon the attribute values
 2018 within the Filter domain class.

- 2019 • The name of the Filter element is <Filter Domain Attribute Name>Filter.
- 2020 • The type of the Filter element is the FilterType complex type that is described in 6.5.3.1.
- 2021 • The cardinality of the Filter element matches the cardinality of the Filter domain attribute in the
 2022 Query domain class.

2023

2024 The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to
 2025 define Filters for the OrganizationQueryType for the Organization class defined by [ebRIM].

2026

```

2027 <complexType name="OrganizationQueryType">
2028   <complexContent>
2029     <extension base="tns:RegistryObjectQueryType">
2030       <sequence>
2031         <element maxOccurs="unbounded" minOccurs="0"
2032           name="AddressFilter" type="tns:FilterType"/>
2033         <element maxOccurs="unbounded" minOccurs="0"
2034           name="TelephoneNumberFilter" type="tns:FilterType"/>
2035         <element maxOccurs="unbounded" minOccurs="0"
2036           name="EmailAdresseFilter" type="tns:FilterType"/>
2037         ...Branches and sub-Queries go here...
2038       </sequence>

```

2039
2040
2041
2042
2043
2044
2045
2046
2047

```
</extension>  
</complexContent>  
</complexType>
```

The following UML class diagram describing the Filter class structure as defined in [RR-QUERY-XSD] XML Schema. Note that the classes whose name ends in “Type” map to complexTypes and other Filter classes map to elements in the [RR-QUERY-XSD] XML Schema.

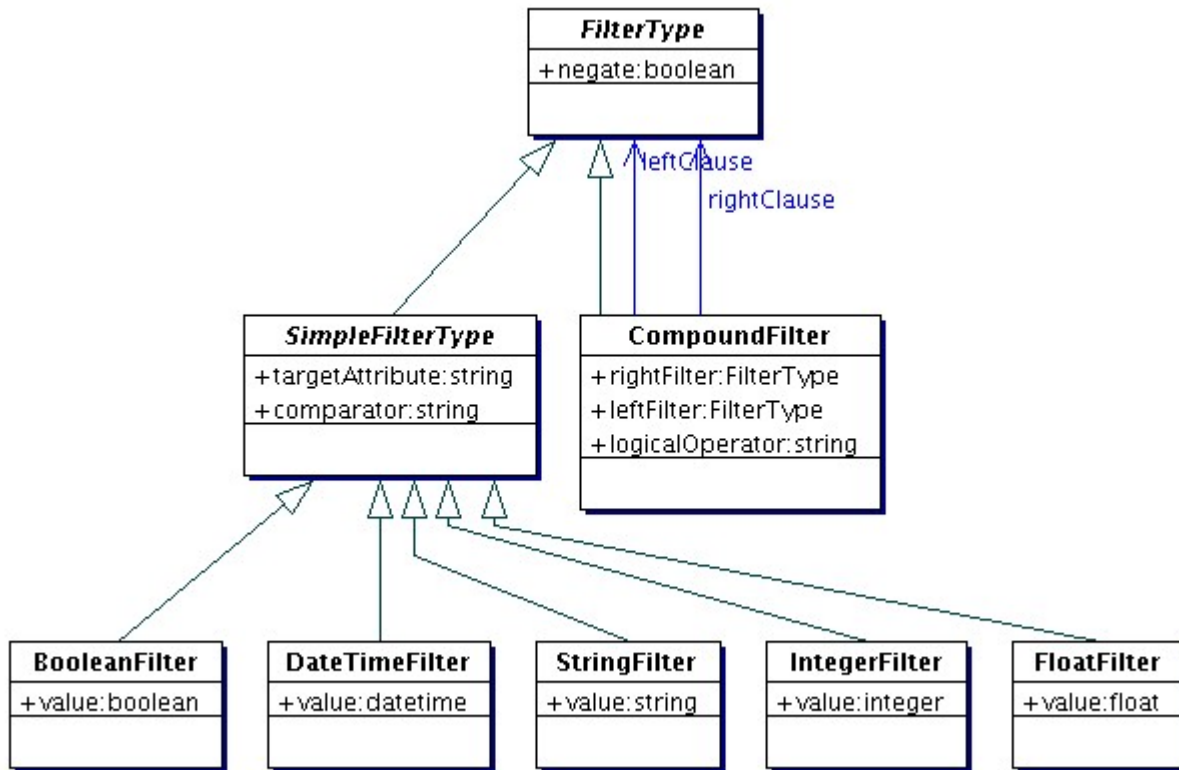


Figure 12: Filter Type Hierarchy

2048
2049

6.5.3.1 FilterType

2050

The FilterType is an abstract complexType that is the root type in the inheritance hierarchy for all Filter types.

2051
2052

6.5.3.1.1 Parameters:

2053

negate: This parameter specifies that the boolean value that the Filter evaluates to MUST be negated to complete the evaluation of the filter. It is functionally equivalent to the NOT operator in SQL syntax.

2054
2055
2056

6.5.3.2 SimpleFilterType

2057

The SimpleFilter is the abstract base type for several concrete Filter types defined for primitive type such as boolean, float, integer and string.

2058
2059

2060 **6.5.3.2.1 Parameters:**

2061 **domainAttribute:** This parameter specifies the attribute name of a primitive attribute
2062 within the Filter domain class. A registry MUST return an InvalidQueryException if this
2063 parameter's value does not match the name of primitive attribute within the Filter
2064 domain class. A registry MUST perform the attribute name match in a case insensitive
2065 manner.

2066 **comparator:** This parameter specifies the comparison operator for comparing the
2067 value of the attribute with the value supplied by the filter. The following comparators
2068 are defined:

- 2069 • LE: abbreviation for LessThanOrEqual
- 2070 • LT: abbreviation for LessThan
- 2071 • GE: abbreviation for GreaterThanOrEqual
- 2072 • GT: abbreviation for GreaterThan
- 2073 • EQ: abbreviation for Equal
- 2074 • NE: abbreviation for NotEqual
- 2075 • Like: Same as LIKE operator in SQL-92. MUST only be used in StringFilter.
- 2076 • NotLike: Same as NOT LIKE operator in SQL-92. MUST only be used in
2077 StringFilter.
- 2078

2079 **6.5.3.3 BooleanFilter**

2080 The BooleanFilter MUST only be used for matching primitive attributes whose domain is of type
2081 boolean.

2082 **6.5.3.3.1 Parameters:**

2083 **value:** This parameter specifies the value that MUST be compared with the attribute
2084 value being tested by the Filter. It MUST be a boolean value.

2085 The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the
2086 ClassificationScheme class defined by [ebRIM]:

```
2087 <BooleanFilter  
2088   domainAttribute="isInternal" comparator="EQ" value="true"/>
```

2090 **6.5.3.4 FloatFilter**

2091 The FloatFilter MUST only be used for matching primitive attributes whose domain is of type float.

2092 **6.5.3.4.1 Parameters:**

2093 **value:** This parameter specifies the value that MUST be compared with the attribute
2094 value being tested by the Filter. It MUST be a float value.

2095 The following example shows the use of a FloatFilter to match fictitious *amount* float attribute since
2096 [ebRIM] currently has no float attributes defined:

```
2097 <FloatFilter  
2098   domainAttribute="amount" comparator="GT" value="9.99"/>
```

2099

2100 6.5.3.5 IntegerFilter

2101 The IntegerFilter MUST only be used for matching primitive attributes whose domain is of type integer.

2102 6.5.3.5.1 Parameters:

2103 **value:** This parameter specifies the value that MUST be compared with the attribute
2104 value being tested by the Filter. It MUST be an integer value.

2105 The following example shows the use of a BooleanFilter to match a fictitious *count* integer attribute
2106 since [ebRIM] currently has no integer attributes defined:

```
2107 <IntegerFilter  
2108     domainAttribute="amount" comparator="LT" value="100"/>  
2109
```

2110 6.5.3.6 DateTimeFilter

2111 The DateTimeFilter MUST only be used for matching primitive attributes whose domain is of type
2112 datetime.

2113 6.5.3.6.1 Parameters:

2114 **value:** This parameter specifies the value that MUST be compared with the attribute
2115 value being tested by the Filter. It MUST be a datetime value.

2116 The following example shows the use of a DateTimeFilter to match a the *timestamp* attribute of the
2117 Auditable class defined by [ebRIM] where the timestamp value is greater than (later than) the specified
2118 datetime value:

```
2119 <DateTimeFilter  
2120     domainAttribute="timestamp"  
2121     comparator="GT" value="1997-07-16T19:20+01:00"/>  
2122
```

2123 6.5.3.7 StringFilter

2124 The StringFilter MUST only be used for matching primitive attributes whose domain is of type string.

2125 6.5.3.7.1 Parameters:

2126 **value:** This parameter specifies the value that MUST be compared with the attribute
2127 value being tested by the Filter. It MUST be a string value.

2128 The following example shows the use of a StringFilter to match a the *firstName* attribute of the Person
2129 class defined by [ebRIM] where the firstName value matches the pattern specified by the value:

```
2130 <StringFilter  
2131     domainAttribute="firstName"  
2132     comparator="Like" value="Farid%"/>  
2133
```

2134 6.5.3.8 CompoundFilter

2135 The CompoundFilter MAY be used to specify a boolean conjunction (AND) or disjunction (OR) between
2136 two Filters. It allows a query to express a combination of predicate clauses within a Filter Query.

2137 6.5.3.8.1 Parameters:

2138 **LeftFilter:** This parameter specifies the first of two Filters for the CompoundFilter.

2139 **RightFilter:** This parameter specifies the second of two Filters for the CompoundFilter.

2140 **logicalOperator:** This parameter specifies the logical operator. The value of this
2141 parameter MUST be "AND" or "OR"

2142 The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the
2143 ClassificationScheme class defined by [ebRIM]:

```
2144 <CompoundFilter logicalOperator="AND">  
2145   <LeftFilter domainAttribute="targetObject" comparator="EQ"  
2146     value="{REGISTRY_OBJECT_ID}" type="StringFilter"/>  
2147   <RightFilter domainAttribute="associationType" comparator="EQ"  
2148     value="{HAS_MEMBER_ASSOC_TYPE_NODE_ID}" type="StringFilter"/>  
2149 </CompoundFilter>
```

2150 6.5.4 Nested Query Elements

2151 A Query element MAY contain one or more nested Query sub-elements. The purpose of the nested
2152 Query element is to allow traversal of the branches within the network of relationships defined by the
2153 information model and prune or filter those branches that do not meet the predicates specified in the
2154 corresponding Branch element.

2155 The [RR-QUERY-XSD] XML Schema defines zero or more nested Query elements within a Query
2156 element definition using the following pattern:

2157 • A nested Query element is defined for each attribute of the Query domain class that satisfy all of the
2158 following criteria:

- 2159 • The attribute's domain class is a descendant type of the RegistryObjectType.
- 2160 • The attribute's domain class contains reference attributes that link the domain class to some
2161 third class via the reference.

2162 The attribute for which the nested Query is defined is referred to as the Nested Query domain
2163 attribute. The domain class of the nested Query domain attribute is the Query domain class for
2164 the nested Query element.

- 2165 • The name of the nested Query element is <Nested Query Domain Attribute Name>Query.
- 2166 • The type of the nested Query element matches the QueryType for the domain class for the Query
2167 domain attribute.
- 2168 • The cardinality of the nested Query element matches the cardinality of the nested Query domain
2169 attribute in the Query domain class.

2170 The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to
2171 define nested Query elements for the OrganizationQueryType for the Organization class defined by
2172 [ebRIM].

```
2173  
2174 <complexType name="OrganizationQueryType">  
2175   <complexContent>  
2176     <extension base="tns:RegistryObjectQueryType">  
2177       <sequence>  
2178         ...Filters and Branches go here ...  
2179         <element maxOccurs="1" minOccurs="0"  
2180           name="ParentQuery" type="tns:OrganizationQueryType"/>  
2181         <element maxOccurs="unbounded" minOccurs="0"  
2182           name="ChildOrganizationQuery" type="tns:OrganizationQueryType"/>  
2183         <element maxOccurs="1" minOccurs="0"  
2184           name="PrimaryContactQuery" type="tns:PersonQueryType"/>  
2185       </sequence>  
2186     </extension>  
2187   </complexContent>  
2188 </complexType>
```

2189 6.5.5 Branch Elements

2190 A Query element MAY contain one or more Branch sub-elements. A Branch element is similar to the
2191 nested Query element as it too can have sub-elements that are Filter, Branch and subQuery elements.

2192 However, it is different from Query elements because its type is not a descendant type of
2193 RegistryObjectQueryType. The purpose of the branch element is to allow traversal of the branches
2194 within the network of relationships defined by the information model and prune or filter those branches
2195 that do not meet the predicates specified in the corresponding Branch element.

2196 The [RR-QUERY-XSD] XML Schema defines zero or more Branch elements within a Query element
2197 definition using the following pattern:

2198 • A Branch element is defined for each attribute of the Query domain class that satisfies all of the
2199 following criteria:

- 2200 • The attribute's domain is not a primitive type (e.g. String, float, dateTime, int etc.).
- 2201 • The attribute's domain class contains reference attributes that link the domain class to some
2202 third class via the reference.

2203 The attribute for which the Branch is defined is referred to as the Branch domain attribute. The
2204 domain class of the Branch domain attribute is the Branch domain class for the Branch element.

- 2205 • The name of the Branch element is <Branch Domain Attribute Name>Branch.
- 2206 • The cardinality of the Branch element matches the cardinality of the Branch domain attribute in the
2207 Query domain class.

2208 The following example shows how the [RR-QUERY-XSD] XML Schema uses the above pattern to
2209 define Branches for the RegistryObjectQueryType for the RegistryObject class defined by [ebRIM].

2210

```
2211 <complexType name="RegistryObjectQueryType">  
2212   <complexContent>  
2213     <extension base="tns:FilterQueryType">  
2214       <sequence>  
2215         <element maxOccurs="unbounded" minOccurs="0"  
2216           name="SlotBranch" type="tns:SlotBranchType"/>  
2217         <element maxOccurs="1" minOccurs="0" name="NameBranch"  
2218           type="tns:InternationalStringBranchType"/>  
2219         <element maxOccurs="1" minOccurs="0" name="DescriptionBranch"  
2220           type="tns:InternationalStringBranchType"/>  
2221         ... Relevant Filters, queries go here...  
2222       </sequence>  
2223     </extension>  
2224   </complexContent>  
2225 </complexType>
```

2226

2227 6.6 Query Examples

2228 This section provides examples in both SQL and Filter Query syntax for some common query use
2229 cases. Each example gives the SQL syntax for the query followed by blank line followed by the
2230 equivalent Filter Query syntax for it.

2231 6.6.1 Name and Description Queries

2232 The following queries matches all RegistryObject instances whose name contains the word 'Acme' and
2233 whose description contains the word "bicycle".

2234

```
2235 SELECT ro.* from RegistryObject ro, Name nm, Description d WHERE  
2236   nm.value LIKE '%Acme%' AND  
2237   d.value LIKE '%bicycle%' AND  
2238   (ro.id = nm.parent AND ro.id = d.parent);  
2239  
2240 <RegistryObjectQuery>  
2241   <NameBranch>  
2242     <LocalizedStringFilter comparator="Like" domainAttribute="value"  
2243       value="%Acme%" xsi:type="StringFilterType"/>  
2244   </NameBranch>
```

```

2245 <DescriptionBranch>
2246   <LocalizedStringFilter comparator="Like" domainAttribute="value"
2247     value="%bicycle%" xsi:type="StringFilterType"/>
2248 </DescriptionBranch>
2249 </RegistryObjectQuery>
2250

```

2251

2252 6.6.2 Classification Queries

2253 This section describes various classification related queries.

2254 6.6.2.1 Retrieving ClassificationSchemes

2255 The following query retrieves the collection of all ClassificationSchemes. Note that the above query
 2256 may also specify additional Filters, Queries and Branches as search criteria if desired.

2257

```

2258 SELECT scheme.* FROM ClassificationScheme scheme;
2259
2260 <ClassificationSchemeQuery/>

```

2261

2262 6.6.2.2 Retrieving Children of Specified ClassificationNode

2263 The following query retrieves the children of a ClassificationNode given the "id" attribute of the parent
 2264 ClassificationNode:

2265

```

2266 SELECT cn.* FROM ClassificationNode cn WHERE parent = ${PARENT_ID};
2267
2268 <ClassificationNodeQuery>
2269   <PrimaryFilter comparator="Like" domainAttribute="parent"
2270     value="${PARENT_ID}" xsi:type="StringFilterType"/>
2271 </ClassificationNodeQuery>

```

2272

2273 6.6.2.3 Retrieving Objects Classified By a ClassificationNode

2274 The following query retrieves the collection of ExtrinsicObjects that are classified by the Automotive
 2275 Industry and the Japan Geography. Note that the query does not match ExtrinsicObjects classified by
 2276 descendant ClassificationNodes of the Automotive Industry and the Japan Geography. That would
 2277 require a slightly more complex query.

2278

```

2279 SELECT eo.* FROM ExtrinsicObject eo WHERE
2280   id IN (SELECT classifiedObject FROM Classification
2281     WHERE
2282       classificationNode IN (SELECT id FROM ClassificationNode
2283         WHERE path = '\${GEOGRAPHY_SCHEME_ID}/Asia/Japan'))
2284 AND
2285   id IN (SELECT classifiedObject FROM Classification
2286     WHERE
2287       classificationNode IN (SELECT id FROM ClassificationNode
2288         WHERE path = '\${INDUSTRY_SCHEME_ID}/Automotive'))
2289
2290 <ExtrinsicObjectQuery>
2291   <ClassificationQuery>
2292     <ClassificationNodeQuery>
2293       <PrimaryFilter comparator="EQ" domainAttribute="path"

```

```

2294         value="/${GEOGRAPHY_SCHEME_ID}/Asia/Japan"
2295         xsi:type="StringFilterType"/>
2296     </ClassificationNodeQuery>
2297 </ClassificationQuery>
2298 <ClassificationQuery>
2299     <ClassificationNodeQuery>
2300         <PrimaryFilter comparator="EQ" domainAttribute="path"
2301             value="/${INDUSTRY_SCHEME_ID}/Automotive"
2302             xsi:type="StringFilterType"/>
2303     </ClassificationNodeQuery>
2304 </ClassificationQuery>
2305 </ExtrinsicObjectQuery>

```

2306

2307 **6.6.2.4 Retrieving Classifications that Classify an Object**

2308 The following query retrieves the collection of Classifications that classify a object with id matching
2309 \${ID}:

2310

```

2311 SELECT c.* FROM Classification c
2312         WHERE c.classifiedObject = ${ID};
2313
2314 <ClassificationQuery>
2315     <PrimaryFilter comparator="EQ" domainAttribute="classifiedObject"
2316         value="${ID}" xsi:type="StringFilterType"/>
2317 </ClassificationQuery>

```

2318

2319 **6.6.3 Association Queries**

2320 This section describes various Association related queries.

2321 **6.6.3.1 Retrieving All Associations With Specified Object As Source**

2322 The following query retrieves the collection of Associations that have the object with id matching
2323 \${SOURCE_ID} as their source:

2324

```

2325 SELECT a.* FROM Association a WHERE sourceObject = ${SOURCE_ID}
2326
2327 <AssociationQuery>
2328     <PrimaryFilter comparator="EQ" domainAttribute="sourceObject"
2329         value="${SOURCE_ID}" xsi:type="StringFilterType"/>
2330 </AssociationQuery>

```

2331

2332 **6.6.3.2 Retrieving All Associations With Specified Object As Target**

2333 The following query retrieves the collection of Associations that have the object with id matching
2334 \${TARGET_ID} as their target:

2335

```

2336 SELECT a.* FROM Association a WHERE targetObject = ${TARGET_ID}
2337
2338 <AssociationQuery>
2339     <PrimaryFilter comparator="EQ" domainAttribute="targetObject"
2340         value="${TARGET_ID}" xsi:type="StringFilterType"/>
2341 </AssociationQuery>

```

2342

2343 6.6.3.3 Retrieving Associated Objects Based On Association Type

2344

2345 Select Associations whose associationType attribute value matches the value specified by the
2346 `{ASSOC_TYPE_ID}`. The `{ASSOC_TYPE_ID}` value MUST reference a ClassificationNode that is a
2347 descendant of the canonical AssociationType ClassificationScheme.

2348

```
2349 SELECT a.* FROM Association a WHERE  
2350     associationType = {ASSOC_TYPE_ID}
```

2351

```
2352 <AssociationQuery>
```

```
2353   <PrimaryFilter comparator="EQ" domainAttribute="associationType"
```

```
2354     value="{ASSOC_TYPE_ID}" xsi:type="StringFilterType"/>
```

```
2355 </AssociationQuery>
```

2356

2357

2358 6.6.3.4 Complex Association Query

2359 The various forms of Association queries may be combined into complex predicates. The following
2360 query selects Associations that match specified specific sourceObject, targetObject and
2361 associationType:

2362

```
2363 SELECT a.* FROM Association a WHERE  
2364     sourceObject = {SOURCE_ID} AND  
2365     targetObject = {TARGET_ID} AND  
2366     associationType = {ASSOC_TYPE_ID};
```

2367

```
2368 <AssociationQuery>
```

```
2369   <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
```

```
2370     <LeftFilter comparator="EQ" domainAttribute="sourceObject"
```

```
2371       xsi:type="StringFilterType" value="{SOURCE_ID}"/>
```

```
2372     <RightFilter logicalOperator="AND" xsi:type="CompoundFilterType">
```

```
2373       <LeftFilter comparator="EQ" domainAttribute="targetObject"
```

```
2374         xsi:type="StringFilterType" value="{TARGET_ID}"/>
```

```
2375       <RightFilter comparator="EQ" domainAttribute="associationType"
```

```
2376         xsi:type="StringFilterType" value="{ASSOC_TYPE_ID}"/>
```

```
2377     </RightFilter>
```

```
2378   </PrimaryFilter>
```

```
2379 </AssociationQuery>
```

2380

2381 6.6.4 Package Queries

2382 The following query retrieves all Packages that have as member the RegistryObject specified by
2383 `{REGISTRY_OBJECT_ID}`:

2384

```
2385 SELECT p.* FROM Package p, Association a WHERE  
2386     a.sourceObject = p.id AND  
2387     a.targetObject = {REGISTRY_OBJECT_ID} AND  
2388     a.associationType = {HAS_MEMBER_ASSOC_TYPE_NODE_ID};
```

2389

```
2390 <RegistryPackageQuery>
```

```
2391   <SourceAssociationQuery>
```

```
2392     <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
```

```
2393       <LeftFilter comparator="EQ" domainAttribute="targetObject"
```

```
2394         value="{REGISTRY_OBJECT_ID}"
```

```
2395         xsi:type="StringFilterType"/>
```

```
2396     <RightFilter comparator="EQ" domainAttribute="associationType"
2397         value="{HAS_MEMBER_ASSOC_TYPE_NODE_ID}"
2398         xsi:type="StringFilterType"/>
2399     </PrimaryFilter>
2400 </SourceAssociationQuery>
2401 </RegistryPackageQuery>
```

2402

2403 Note that the `{HAS_MEMBER_ASSOC_TYPE_NODE_ID}` is a placeholder for the value of the id
2404 attribute of the canonical HasMember AssociationType ClassificationNode.

2405 6.6.5 ExternalLink Queries

2406 The following query retrieves all ExternalLinks that serve as ExternalLink for the RegistryObject
2407 specified by `{REGISTRY_OBJECT_ID}`:

2408

```
2409 SELECT el.* From ExternalLink el, Association a WHERE
2410     a.sourceObject = el.id AND
2411     a.targetObject = {REGISTRY_OBJECT_ID} AND
2412     a.associationType = {EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};
2413
2414 <ExternalLinkQuery>
2415     <SourceAssociationQuery>
2416         <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2417             <LeftFilter comparator="EQ" domainAttribute="targetObject"
2418                 value="{REGISTRY_OBJECT_ID}"
2419                 xsi:type="StringFilterType"/>
2420             <RightFilter comparator="EQ" domainAttribute="associationType"
2421                 value="{EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
2422                 xsi:type="StringFilterType"/>
2423         </PrimaryFilter>
2424     </SourceAssociationQuery>
2425 </ExternalLinkQuery>
```

2426

2427 Note that the `{EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}` is a placeholder for the value of the
2428 id attribute of the canonical ExternallyLinks AssociationType ClassificationNode.

2429 The following query retrieves all ExtrinsicObjects that are linked to an ExternalLink specified by
2430 `{EXTERNAL_LINK_ID}`:

2431

```
2432 SELECT eo.* From ExtrinsicObject eo, Association a WHERE
2433     a.sourceObject = {EXTERNAL_LINK_ID} AND
2434     a.targetObject = eo.id AND
2435     a.associationType = {EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};
2436
2437 <ExtrinsicObjectQuery>
2438     <TargetAssociationQuery>
2439         <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2440             <LeftFilter comparator="EQ" domainAttribute="sourceObject"
2441                 value="{EXTERNAL_LINK_ID}"
2442                 xsi:type="StringFilterType"/>
2443             <RightFilter comparator="EQ" domainAttribute="associationType"
2444                 value="{EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
2445                 xsi:type="StringFilterType"/>
2446         </PrimaryFilter>
2447     </TargetAssociationQuery>
2448 </ExtrinsicObjectQuery>
```

2449

2450 6.6.6 Audit Trail Queries

2451 The following query retrieves all the AuditableEvents for the RegistryObject specified by
2452 \${REGISTRY_OBJECT_ID}:

2453

```
2454 SELECT ae.* FROM AuditableEvent ae, AffectedObject ao WHERE  
2455     ao.eventId = ae.id AND  
2456     ao.id = ${REGISTRY_OBJECT_ID}
```

2457

```
2458 <AuditableEventQuery>
```

```
2459   <AffectedObjectQuery>
```

```
2460     <PrimaryFilter comparator="EQ" domainAttribute="id"
```

```
2461       value="${REGISTRY_OBJECT_ID}" xsi:type="StringFilterType"/>
```

```
2462   </AffectedObjectQuery>
```

```
2463 </AuditableEventQuery>
```

2464

2465 **7 Event Notification Protocols**

2466 This chapter defines the Event Notification feature of the OASIS ebXML Registry.

2467 Event Notification feature allows OASIS ebXML Registries to notify its users and / or other registries
2468 about events of interest. It allows users to stay informed about registry events without being forced to
2469 periodically poll the registry. It also allows a registry to propagate internal changes to other registries
2470 whose content might be affected by those changes.

2471 ebXML registries support content-based Notification where interested parties express their interest in
2472 form of a query. This is different from subject-based (sometimes referred to as topic-based) notification,
2473 where information is categorized by subjects and interested parties express their interests in those
2474 predefined subjects.

2475 **7.1 Use Cases**

2476 The following use cases illustrate different ways in which ebXML registries notify users or other
2477 registries.

2478 **7.1.1 CPP Has Changed**

2479 A user wishes to know when the CPP [ebCPP] of a partner is updated or superseded by another CPP.
2480 When that happens he may wish to create a CPA [ebCPP] based upon the new CPP.

2481 **7.1.2 New Service is Offered**

2482 A user wishes to know when a new plumbing service is offered in her town and be notified every 10
2483 days. When that happens, she might try to learn more about that service and compare it with her
2484 current plumbing service provider's offering.

2485 **7.1.3 Monitor Download of Content**

2486 User wishes to know whenever his CPP [ebCPP] is downloaded in order to evaluate on an ongoing
2487 basis the success of his recent advertising campaign. He might also want to analyze who the
2488 interested parties are.

2489 **7.1.4 Monitor Price Changes**

2490 User wishes to know when the price of a product that she is interested in buying drops below a certain
2491 amount. If she buys it she would also like to be notified when the product has been shipped to her.

2492 **7.1.5 Keep Replicas Consistent With Source Object**

2493 In order to improve performance and availability of accessing some registry objects, a local registry
2494 MAY make replicas of certain objects that are hosted by another registry. The registry would like to be
2495 notified when the source object for a replica is updated so that it can synchronize the replica with the
2496 latest state of the source object.

2497 **7.2 Registry Events**

2498 Activities within a registry result in meaningful events. Typically, registry events are generated when a
2499 registry processes client requests. In addition, certain registry events may be caused by administrative
2500 actions performed by a registry operator. [ebRIM] defines the AuditableEvent class, instances of which
2501 represent registry events. When such an event occurs, an AuditableEvent instance is generated by the
2502 registry.

2503 7.3 Subscribing to Events

2504 A user MAY create a subscription with a registry if he or she wishes to receive notification for a specific
2505 type of event. A user creates a subscription by submitting a Subscription instance to a registry using
2506 the SubmitObjectsRequest. If a Subscription is submitted to a registry that does not support event
2507 notification then the registry MUST return an UnsupportedOperationException.

2508 The listing below shows a sample Subscription using a pre-defined SQL query as its selector that will
2509 result in an email notification to the user whenever a Service is created that is classified as a
2510 "Plumbing" service and located in "A Little Town."
2511

2512 The SQL query within the selector in plain English says the following:

2513 *Find all Services that are Created AND classified by ClassificationNode*
2514 *where ClassificationNode's Path ends with string "Plumbing", AND classified by ClassificationNode*
2515 *where ClassificationNode's Code contains string "A Little Town."*

2516

```
2517 <rim:Subscription id="{SUBSCRIPTION_ID}" selector="{QUERY_ID}">
2518   <!--
2519     The selector is a reference to a query object that has the
2520     following query defined
2521     SELECT * FROM Service s, AuditableEvent e, AffectedObject ao,
2522     Classification c1, Classification c2
2523     ClassificationNode cn1, ClassificationNode cn2 WHERE
2524     e.eventType = 'Created' AND ao.id = s.id AND ao.parent=e.id AND
2525     c1.classifiedObject = s.id AND c1.classificationNode = cn1.id
2526 AND
2527     cn1.path LIKE '%Plumbing' AND
2528     c2.classifiedObject = s.id AND c2.classificationNode = cn2.id
2529 AND
2530     cn2.path LIKE '%A Little Town%'
2531   -->
2532   <!-- Next endPoint is an email address -->
2533   <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-
2534   regrep:NotificationOptionType:Objects"
2535   endPoint="mailto:farrukh.najmi@sun.com"/>
2536   <!-- Next endPoint is a service via reference to its ServiceBinding
2537   object -->
2538   <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-
2539   regrep:NotificationOptionType:ObjectRefs"
2540   endPoint="urn:freebxml:registry:demoDB:serviceBinding:EpidemicAlertList
2541   enerServiceBinding"/>
2542 </rim:Subscription>
```

2543

2544 7.3.1 Event Selection

2545 In order to only be notified of specific events of interest, the user MUST specify a reference to a stored
2546 AdHocQuery object via the selector attribute within the Subscription instance. The query determines
2547 whether an event qualifies for that Subscription or not. For details on query syntax see chapter 6.

2548 7.3.2 Notification Action

2549 When creating a Subscription, a user MAY also specify Actions within the subscription that specify
2550 what the registry must do when an event matching the Subscription (subscription event) transpires.

2551 A user MAY omit specifying an Action within a Subscription if he does not wish to be notified by the
2552 registry. A user MAY periodically poll the registry and pull the pending Notifications.

2553 [ebRIM] defines two standard ways that a NotifyAction may be used:

2554 • Email NotifyAction that allows delivery of event notifications via email to a human user or to an
2555 email end point for a software component or agent.

2556 • Service NotifyAction that allows delivery of event notifications via a programmatic interface by
2557 invoking a specified listener web service.

2558 If the registry supports event notification, at some time after the successful processing of each request,
2559 it MUST check all registered and active Subscriptions and see if any Subscriptions match the event. If
2560 a match is found then the registry performs the Notification Actions required for the Subscription. A
2561 registry MAY periodically perform such checks and corresponding notification actions in a batch mode
2562 based upon registry specific policies.

2563 **7.3.3 Subscription Authorization**

2564 A registry operator or content owner MAY use custom Access Control Policies to decide which users
2565 are authorized to create a subscription and to what events. A Registry MUST return an
2566 AuthorizationException in the event that an unauthorized user submits a Subscription to a registry. It is
2567 up to registry implementations whether to honour the existing subscription if an access control policy
2568 governing subscriptions becomes more restrictive after subscription have already been created based
2569 on the older policy.

2570 **7.3.4 Subscription Quotas**

2571 A registry MAY use registry specific policies to decide an upper limit on the number of Subscriptions a
2572 user is allowed to create. A Registry MUST return a QuotaExceededException in the event that an
2573 authorized user submits more Subscriptions than allowed by their registry specific quota.

2574 **7.3.5 Subscription Expiration**

2575 Each subscription defines a startTime and an endTime attribute which determines the period within
2576 which a Subscription is active. Outside the bounds of the active period, a Subscription MAY exist in an
2577 expired state within the registry. A registry MAY remove an expired Subscription at any time. In such
2578 cases the identity of a RegistryOperator user MUST be used for the request in order to have sufficient
2579 authorization to remove a user's Subscription.

2580 A Registry MUST NOT consider expired Subscriptions when delivering notifications for an event to its
2581 Subscriptions. An expired Subscription MAY be renewed by submitting a new Subscription.

2582 **7.3.6 Subscription Rejection**

2583 A Registry MAY reject a Subscription if it is too costly to support. For instance a Subscription that
2584 wishes to be notified of any change in any object may be too costly for most registries. A Registry
2585 MUST return a SubscriptionTooCostlyException in the event that an Authorized User submits a
2586 Subscription that is too costly for the registry to process.

2587 **7.4 Unsubscribing from Events**

2588 A user MAY terminate a Subscription with a registry if he or she no longer wishes to be notified of
2589 events related to that Subscription. A user terminates a Subscription by deleting the corresponding
2590 Subscription object using the RemoveObjectsRequest to the registry.

2591 Removal of a Subscription object follows the same rules as removal of any other object.

2592 **7.5 Notification of Events**

2593 A registry performs the *Actions* for a Subscription in order to actually deliver the events information to
2594 the subscriber. However, regardless of the specific delivery Action, the registry MUST communicate
2595 the Subscription events. The Subscription events are delivered within a Notification instance as
2596 described by [ebRIM]. In case of Service NotifyAction, the Notification is delivered to a handler service
2597 conformant to the RegistryClient interface. In case of an Email NotifyAction the notification is delivered

2598 an email address.

2599 The listing below shows a sample Notification matching the subscription example in section 7.3:

2600

```
2601 <rim:Notification subscription="{SUBSCRIPTION_ID}">
2602   <rim:RegistryObjectList>
2603     <rim:Service id="f3373a7b-4958-4e55-8820-d03a191fb76a">
2604       <rim:Name>
2605         <rim:LocalizedString value="A Little Town Plumbing"/>
2606       </rim:Name>
2607       <rim:Classification id="a3373a7b-4958-4e55-8820-d03a191fb76a"
2608 classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
2609       <rim:Classification id="b3373a7b-4958-4e55-8820-d03a191fb76a"
2610 classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>
2611     </rim:Service>
2612   </rim:RegistryObjectList>
2613 </rim:Notification>
```

2614

2615 A Notification MAY contain actual RegistryObjects or ObjectRefs to RegistryObjects within the
2616 <rim:RegistryObjectList>. A client MAY specify the whether they wish to receive RegistryObjects or
2617 ObjectRefs to RegistryObjects using the notificationOption attribute of the Action within the
2618 Subscription. The registry MAY override this notificationOption based upon registry specific operational
2619 policies.

2620 **7.6 Retrieval of Events**

2621 The registry provides asynchronous PUSH style delivery of Notifications via notify Actions as described
2622 earlier. However, a client MAY also use a PULL style to retrieve any pending events for their
2623 Subscriptions. Pulling of events is done using the AdHocQuery protocol and querying the Notification
2624 class. A registry SHOULD buffer undelivered notifications for some period to allow clients to PULL
2625 those notifications. The period that a registry SHOULD buffer undelivered notifications MAY be defined
2626 using registry specific policies.

2627 **7.7 Pruning of Events**

2628 A registry MAY periodically prune AuditableEvents in order to manage its resources. It is up to the
2629 registry when such pruning occurs. It is up to the registry to determine when undelivered events are
2630 purged. A registry SHOULD perform such pruning by removing the older information in its Audit Trail
2631 content. However, it MUST not remove the original Create Event at the beginning of the audit trail since
2632 the Create Event establishes the owner of the RegistryObject.

8 Content Management Services

2633

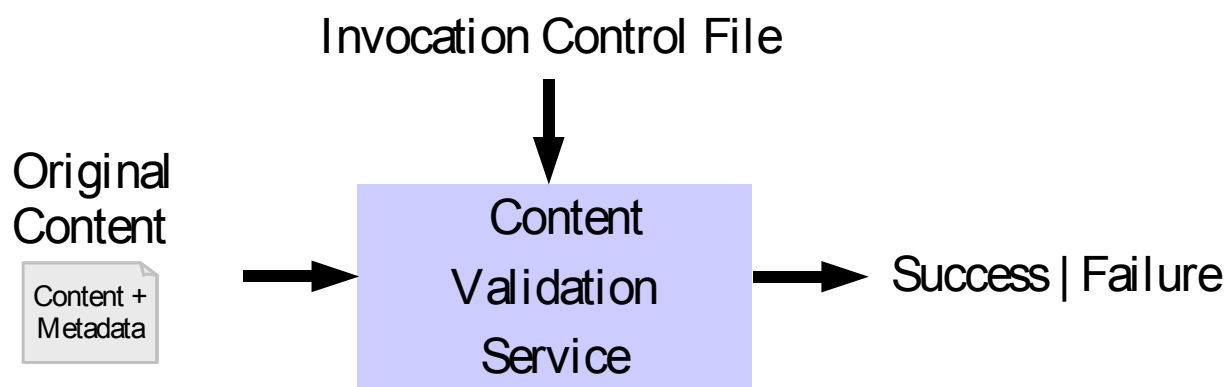
2634 This chapter describes the Content Management services of the ebXML Registry. Examples of Content
2635 Management Services include, but are not limited to, content validation and content cataloging.
2636 Content Management Services result in improved quality and integrity of registry content and metadata
2637 as well as improved ability for clients to discover that content and metadata.

2638 The Content Management Services facility of the registry is based upon a pluggable architecture that
2639 allows clients to publish and discover new Content Management Services as Service objects that
2640 conform to a normative web service interface specified in this chapter. Clients MAY configure a Content
2641 Management Service that is specialized for managing a specific type of content.

8.1 Content Validation

2642

2643 The Content Validation feature provides the ability to enforce domain specific validation rules upon
2644 submitted content and metadata in a content specific manner.



2645

2646

Figure 13: Content Validation Service

2647 A registry uses one or more Content Validation Services to automatically validate the RegistryObjects
2648 and repository items when they are submitted to the registry. A registry MUST reject a submission
2649 request in its entirety if it contains invalid data. In such cases a ValidationException MUST be returned
2650 to the client.

2651 Content Validation feature improves the quality of data in the registry.

8.1.1 Content Validation: Use Cases

2652

2653 The following use cases illustrate the Content Validation feature:

8.1.1.1 Validation of HL7 Conformance Profiles

2654

2655 The Healthcare Standards organization HL7 uses content validation to enforce consistency rules and
2656 semantic checks whenever an HL7 member submits an HL7 Conformance Profile. HL7 is also planning
2657 to use the feature to improve the quality of other types of HL7 artifacts.

8.1.1.2 Validation of Business Processes

2658

2659 Content validation may be used to enforce consistency rules and semantic checks whenever a
2660 Business Process is submitted to the registry. This feature may be used by organizations such as
2661 UN/CEFACT, OAGi, and RosettaNet.

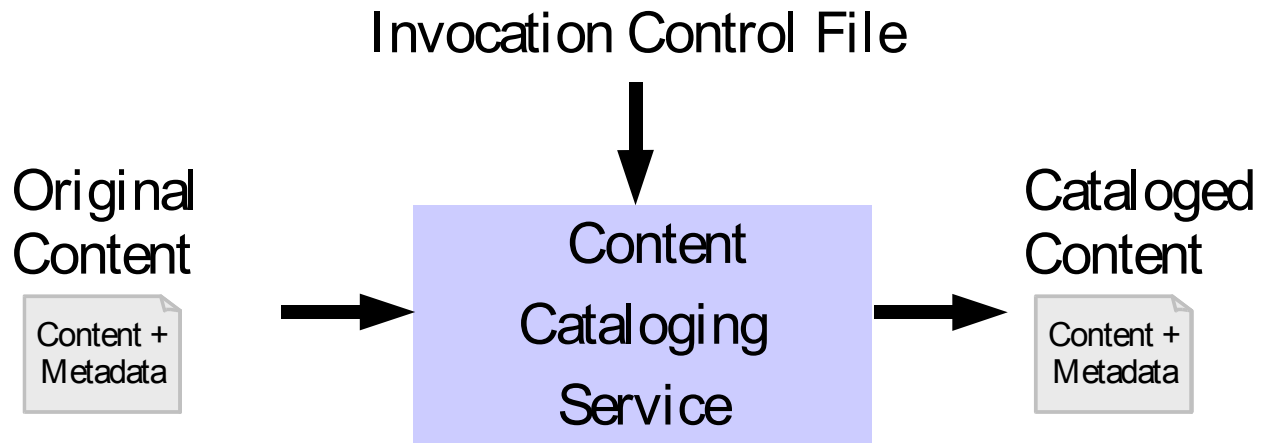
8.1.1.3 Validation of UBL Business Documents

2662

2663 Content validation may be used by the UBL technical committee to enforce consistency rules and
2664 semantic checks whenever a UBL business document is submitted to the registry.

2665 8.2 Content Cataloging

2666 The Content Cataloging feature provides the ability to selectively convert submitted RegistryObject and
2667 repository items into metadata defined by [ebRIM], in a content specific manner.



2668
2669

Figure 14: Content Cataloging Service

2670 A registry uses one or more Content Cataloging Services to automatically catalog RegistryObjects and
2671 repository items. Cataloging creates and/or updates RegistryObject metadata such as ExtrinsicObject
2672 or Classification instances. The cataloged metadata enables clients to discover the repository item
2673 based upon content from the repository item, using standard query capabilities of the registry. This is
2674 referred to as *Content-based Discovery*.

2675 The main benefit of the Content Cataloging feature is to enable Content-based Discovery.

2676 8.2.1 Content-based Discovery: Use Cases

2677 There are many scenarios where content-based discovery is necessary.

2678 8.2.1.1 Find All CPPs Where Role is “Buyer”

2679 A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to find
2680 CPPs for other companies where the Role element of the CPP is that of “Buyer”.

2681 8.2.1.2 Find All XML Schema’s That Use Specified Namespace

2682 A client may wish to discover all XML Schema documents in the registry that use an XML namespace
2683 containing the word “oasis”.

2684 8.2.1.3 Find All WSDL Descriptions with a SOAP Binding

2685 An ebXML registry client is attempting to discover all repository items that are WSDL descriptions that
2686 have a SOAP binding defined. Note that SOAP binding related information is content within the WSDL
2687 document and not metadata.

2688 8.3 Abstract Content Management Service

2689 This section describes in abstract terms how the registry supports pluggable, user-defined Content
2690 Management Services. A Content Management Service is invoked in response to content being
2691 submitted to the registry via the standard Submit/UpdateObjectsRequest method. The Service
2692 invocation is on a per request basis where one request may result in many invocations, one for each
2693 RegistryObject for which a Content Management Service is configured within the registry.

2694 The registry may perform such invocation in one of two ways.

2695

- 2696
- 2697
- 2698
- **Inline Invocation Model:** Content Management Service may be invoked inline with the processing of the Submit/UpdateObjectsRequest and prior to committing the content. This is referred to as Inline Invocation Model.
 - **Decoupled Invocation Model:** Content Management Service may be invoked decoupled from the processing of the Submit/UpdateObjectsRequest and some time after committing the content. This is referred to as Decoupled Invocation Model.
- 2699
- 2700
- 2701
- 2702

2703 8.3.1 Inline Invocation Model

2704 In an inline invocation model a registry MUST invoke a Content Management Service inline with
2705 Submit/UpdateObjectsRequest processing and prior to committing the Submit/UpdateObjectsRequest.
2706 All metadata and content from the original Submit/UpdateObjectsRequest request or from the Content
2707 Management Service invocation MUST be committed as an atomic transaction.

2708 Figure 15 shows an abstract Content Management Service and how it is used by an ebXML Registry
2709 using an inline invocation model. The steps are as follows:

2710

- 2711 1. A client submits a Content Management Service S1 to an ebXML Registry. The client
2712 typically belongs to an organization responsible for defining a specific type of content.
2713 For example the client may belong to RosettaNet.org and submit a Content Validation
2714 Service for validating RosettaNet PIPs. The client uses the standard
2715 Submit/UpdateObjectsRequest interface to submit the Service. This is a one-time step to
2716 configure this Content Management Service in the registry.
 - 2717 2. Once the Content Management Service has been submitted, a potentially different client
2718 may submit content to the registry that is of the same object type for which the Content
2719 Management Service has been submitted. The client uses the standard
2720 Submit/UpdateObjectsRequest interface to submit the content.
 - 2721 3. The registry determines there is a Content Management Service S1 configured for the
2722 object type for the content submitted. It invokes S1 using a
2723 ContentManagementServiceRequest and passes it the content.
 - 2724 4. The Content Management Service S1 processes the content and sends back a
2725 ContentManagementServiceResponse.
 - 2726 5. The registry then commits the content to the registry if there are no errors encountered.
 - 2727 6. The registry returns a RegistryResponse to the client for the
2728 Submit/UpdateObjectsRequest in step 2.
- 2729
- 2730
- 2731



Figure 15: Content Management Service: Inline Invocation Model

2732
2733

2734 **8.3.2 Decoupled Invocation Model**

2735 In a decoupled invocation model a registry **MUST** invoke a Content Management Service independent
2736 of or decoupled from the Submit/UpdateObjectsRequest processing. Any errors encountered during
2737 Content Management Service invocation **MUST NOT** have any impact on the original
2738 Submit/UpdateObjectsRequest processing.

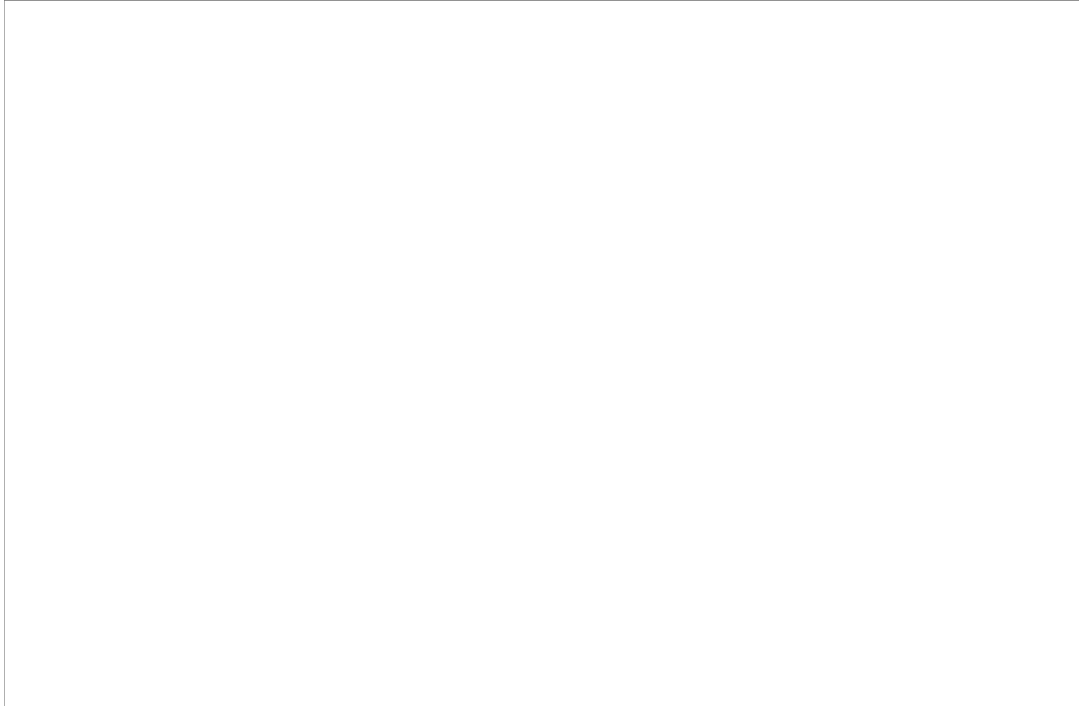
2739 All metadata and content from the original Submit/UpdateObjectsRequest request **MUST** be committed
2740 as an atomic transaction that is decoupled from the metadata and content that may be generated by
2741 the Content Management Service invocation.

2742 Figure 16 shows an abstract Content Management Service and how it is used by an ebXML Registry
2743 using a decoupled invocation model. The steps are as follows:

2744

- 2745 1. Same as in inline invocation model (Content Management Service is submitted).
- 2746 2. Same as in inline invocation model (client submits content using
2747 Submit/UpdateObjectsRequest).
- 2748 3. The registry processes the Submit/UpdateObjectsRequest and commits it to persistent
2749 store.
- 2750 4. The registry returns a RegistryResponse to the client for the
2751 Submit/UpdateObjectsRequest in step 2.
- 2752 5. The registry determines there is a Content Management Service S1 configured for the
2753 object type for the content submitted. It invokes S1 using a
2754 ContentManagementServiceRequest and passes it the content.
- 2755 6. The Content Management Service S1 processes the content and sends back a
2756 ContentManagementServiceResponse.

2757 7. If the ContentManagementServiceResponse includes any generated or modified content
2758 it is committed to the persistent store as separate transaction. If there are any errors
2759 encountered during decoupled invocation of a Content Management Service then these
2760 errors are logged by the registry in a registry specific manner and MUST NOT be
2761 reported back to the client.
2762



2763 **Figure 16: Content Management Service: Decoupled Invocation Model**
2764

2765 **8.4 Content Management Service Protocol**

2766 This section describe the abstract Content Management Service protocol that is the base- protocol for
2767 other concrete protocols such as Validate Content protocol and Catalog Content protocol. The concrete
2768 protocols will be defined later in this document.

2769 **8.4.1 ContentManagementServiceRequestType**

2770 The ContentManagementServiceRequestType MUST be the abstract base type for all requests sent
2771 from a registry to a Content Management Service.

2772 **8.4.1.1 Syntax:**

```
2773 <complexType name="ContentManagementServiceRequestType">  
2774   <complexContent>  
2775     <extension base="rs:RegistryRequestType">  
2776       <sequence>  
2777         <element name="OriginalContent"  
2778 type="rim:RegistryObjectListType"/>  
2779         <element name="InvocationControlFile"  
2780 type="rim:ExtrinsicObjectType" maxOccurs="unbounded" minOccurs="0"/>  
2781       </sequence>  
2782     </extension>  
2783   </complexContent>  
2784 </complexType>
```

2785

2786 **8.4.1.2 Parameters:**

2787 The following parameters are parameters that are either newly defined for this type or are inherited and
2788 have additional semantics beyond those defined in the base type description.

2789 *InvocationControlFile*: This parameter specifies the ExtrinsicObject for a repository item
2790 that the caller wishes to specify as the Invocation Control File. This specification does
2791 not specify the format of this file. There MUST be a corresponding repository item as an
2792 attachment to this request. The corresponding repository item SHOULD follow the
2793 same rules as attachments in Submit/UpdateObjectsRequest.

2794 *OriginalContent*: This parameter specifies the RegistryObjects that will be processed
2795 by the content management service. In case of ExtrinsicObject instances within the
2796 OriginalContent there MAY be repository items present as attachments to the
2797 ContentManagementServiceRequest. This specification does not specify the format of
2798 such repository items. The repository items SHOULD follow the same rules as
2799 attachments in Submit/UpdateObjectsRequest.
2800

2801 **8.4.1.3 Returns:**

2802 This request returns a ContentManagementServiceResponse. See section 8.4.2 for details.

2803 **8.4.1.4 Exceptions:**

2804 In addition to the exceptions returned by base request types, the following exceptions MAY be
2805 returned:

2806 *MissingRepositoryItemException*: signifies that the caller did not provide a repository
2807 item as an attachment to this request when the Service requires it.

2808 *InvocationControlFileException*: signifies that the InvocationControlFile(s) provided by
2809 the caller do not match the InvocationControlFile(s) expected by the Service.

2810 *UnsupportedContentException*: signifies that this Service does not support the content
2811 provided by the caller.
2812

2813 **8.4.2 ContentManagementServiceResponseType**

2814 The ContentManagementServiceResponseType is sent by a Content Management Service as a
2815 response to a ContentManagementServiceRequestType. The
2816 ContentManagementServiceResponseType is the abstract base type for all responses sent to a
2817 registry from a Content Management Service. It extends the RegistryResponseType and does not
2818 define any new parameters.
2819

2820 **8.4.2.1 Syntax:**

```
2821 <complexType name="ContentManagementServiceResponseType">  
2822   <complexContent>  
2823     <extension base="rs:RegistryResponseType">  
2824       <sequence>  
2825         </sequence>  
2826       </extension>  
2827     </complexContent>  
2828   </complexType>
```

2829

2830 **8.4.2.2 Parameters:**

2831 No new parameters are defined other than those inherited from RegistryResponseType.
2832

2833 **8.5 Publishing / Configuration of a Content Management Service**

2834 Any Submitter MAY submit an arbitrary Content Management Service to an ebXML Registry. The
2835 Content Management Service MUST be published using the standard LifecycleManager interface.

2836 The Submitter MUST use the standard Submit/UpdateObjectsRequest to publish:

- 2837 ○ A Service instance for the Content Management Service. In Figure 17 this is exemplified by the
2838 defaultXMLCatalogingService in the upper-left corner. The Service instance MUST have an
2839 Association with a ClassificationNode in the canonical ObjectType ClassificationScheme as
2840 defined by [ebRIM]. The Service MUST be the sourceObject while a ClassificationNode MUST
2841 be the targetObject. This association binds the Service to that specific ObjectType. The
2842 associationType for this Association instance MUST be "ContentManagementServiceFor." The
2843 Service MUST be classified by the canonical ContentManagementService
2844 ClassificationScheme as defined by [ebRIM]. For example it may be classified as a
2845 "ContentValidationService" or a "ContentCatalogingService."
- 2846 ○ The Service instance MAY be classified by a ClassificationNode under the canonical
2847 InvocationModel ClassificationScheme as defined by [ebRIM], to determine whether it uses the
2848 Inline Invocation model or the Decoupled Invocation model.
- 2849 ○ The Service instance MAY be classified by a ClassificationNode under the canonical
2850 ErrorHandlingModel ClassificationScheme as defined by [ebRIM], to determine whether the
2851 Service should fail on first error or simply log the error as a warning and continue. See section
2852 8.6.4 for details.
- 2853 ○ A ServiceBinding instance contained within the Service instance that MUST provide the
2854 accessURI to the Cataloging Service.
- 2855 ○ An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page
2856 describing:
 - 2857 The format of the supported content to be Cataloged
 - 2858 The format of the supported Invocation Control File
- 2859 Note that no SpecificationLink is required since this specification [ebRS] is implicit for Content
2860 Cataloging Services.
- 2861 ○ One or more Invocation Control File(s) consisting of an ExtrinsicObject and a repository item
2862 pair. The ExtrinsicObject for the Invocation Control File MUST have a required Association with
2863 associationType value that references a descendant ClassificationNode of the canonical
2864 ClassificationNode "InvocationControlFileFor." This is exemplified by the
2865 cppCatalogingServiceXSLT and the oagBODCatalogingServiceXSLT objects in Figure 17 (left
2866 side of picture). The Invocation Control File MUST be the sourceObject while a
2867 ClassificationNode in the canonical ObjectType ClassificationScheme MUST be the
2868 targetObject.
- 2869 ○



2870
2871

Figure 17: Cataloging Service Configuration

2872 Figure 17 shows an example of the configuration of the Canonical XML Cataloging Service associated
2873 with the objectType for XML content. This Cataloging Service may be used with any XML content that
2874 has its objectType attribute hold a reference to the xmlObjectType ClassificationNode or one of its
2875 descendants.

2876 The figure also shows two different Invocation Control Files, cppCatalogingServiceXSLT and
2877 oagBODCatalogingServiceXSLT that may be used to catalog ebXML CPP and OAG Business Object
2878 Documents (BOD) respectively.

2879 **8.5.1 Multiple Content Management Services and Invocation Control** 2880 **Files**

2881 This specification allows clients to submit multiple Content Management Services of the same type
2882 (e.g. validation, cataloging) and multiple Invocation Control Files for the same objectType. Content
2883 Management Services of the same type of service for the same ObjectType are referred to as peer
2884 Content Management Services.

2885

2886 When there are multiple Content Management Services and Invocation Control Files for the same
2887 ObjectType there MUST be an unambiguous association between a Content Management Service and
2888 its Invocation Control File(s). This MUST be defined by an Association instance with associationType
2889 value that references a ClassificationNode that is a descendant of the canonical ClassificationNode
2890 "InvocationControlFileFor" where the ExtrinsicObject for each Invocation Control File is the
2891 sourceObject and the Service is the targetObject.

2892 The order of invocation of peer Content Management Services is undefined and MAY be determined in
2893 a registry specific manner.

2894 **8.6 Invocation of a Content Management Service**

2895 This section describes how a registry invokes a Content Management Service.

2896 **8.6.1 Resolution Algorithm For Service and Invocation Control File**

2897 When a registry receives a submission of a RegistryObject, it MUST use the following algorithm to
2898 determine or resolve the Content Management Services and Invocation Control Files to be used for
2899 dynamic content management for the RegistryObject:

2900

- 2901 1. Get the objectType attribute of the RegistryObject.
- 2902 2. Query to see if the ClassificationNode referenced by the objectType is the targetObject of an Association
2903 with associationType of *ContentManagementServiceFor*. If the desired Association is not found for this
2904 ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired
2905 Association is found or until the parent is the ClassificationScheme. If desired Association(s) is found then
2906 repeat following steps for each such Association instance.
- 2907 3. Check if the sourceObject of the desired Association is a Service instance. If not, log an
2908 InvalidConfigurationException. If it is a Service instance, then use this Service as the Content
2909 Management service for the RegistryObject.
- 2910 4. Query to see if the objectType ClassificationNode is the targetObject of one or more Associations whose
2911 associationType value references a ClassificationNode that is a descendant of the canonical
2912 ClassificationNode *InvocationControlFileFor*. If desired Association is not found for this
2913 ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired
2914 Association is found or until the parent is the ClassificationScheme.
- 2915 5. If desired Association(s) is found then check if the sourceObject of the desired Association is an
2916 ExtrinsicObject instance. If not, log an InvalidConfigurationException. If sourceObject is an
2917 ExtrinsicObject instance, then use its repository item as an Invocation Control File. If there are multiple
2918 InvocationControlFiles then all of them MUST be provided when invoking the Service.

2919 The above algorithm allows for objectType hierarchy to be used to configure Content Management
2920 Services and Invocation Control Files with varying degrees of specificity or specialization with respect
2921 to the type of content.

2922 **8.6.2 Audit Trail and Cataloged Content**

2923 The Cataloged Content generated as a result of the invocation of a Content Management Service has
2924 an audit trail consistent with RegistryObject instances that are submitted by Registry Clients. However,
2925 since a Registry Client does not submit Cataloged Content, the user attribute of the AuditableEvent
2926 instances for such Cataloged Content references the Service object for the Content Management
2927 Service that generated the Cataloged Content. This allows an efficient way to distinguish Cataloged
2928 Content from content submitted by Registry Clients.

2929 **8.6.3 Referential Integrity**

2930 A registry MUST maintain referential integrity between the RegistryObjects and repository items
2931 invocation of a Content Management Service.

2932 **8.6.4 Error Handling**

2933 If the Content Management Service is classified by the "FailOnError" ClassificationNode under
2934 canonical ErrorHandlerModel ClassificationScheme as defined by [ebRIM], then the registry MUST
2935 stop further processing of the Submit/UpdateObjectsRequest and return status of "Failure" upon first
2936 error returned by a Content Management Service Invocation.

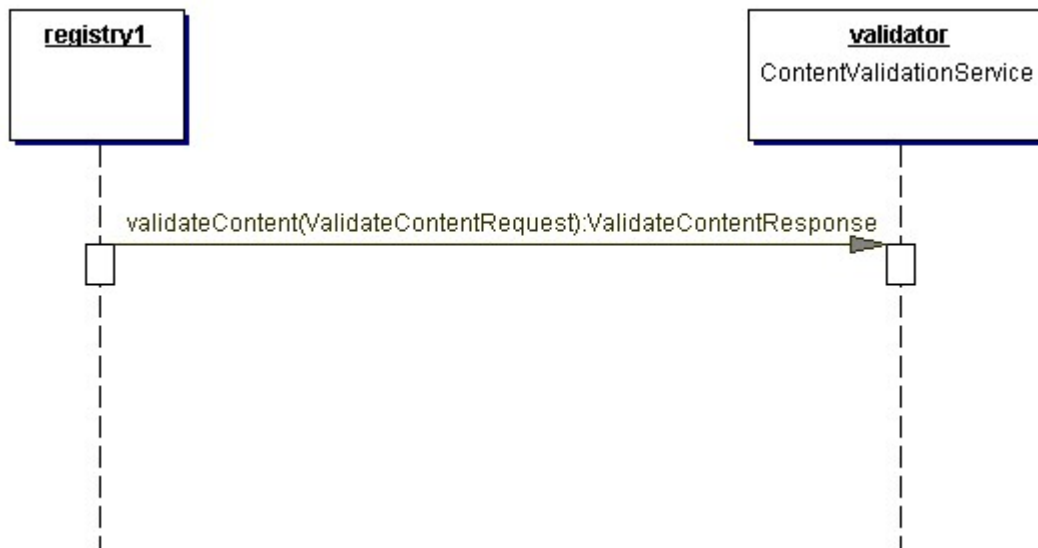
2937 If the Content Management Service is classified by the "LogErrorAndContinue" ClassificationNode
2938 under ErrorHandlerModel then the registry MUST continue to process the
2939 Submit/UpdateObjectsRequest and not let any Content Management Service invocation error affect the
2940 storing of the RegistryObjects and repository items that were submitted. Such errors SHOULD be
2941 logged as Warnings within the RegistryResponse returned to the client. In this case a registry MUST
2942 return a normal response with status of "Success" if the submitted content and metadata is stored
2943 successfully even when there are errors encountered during dynamic invocation of one or more
2944 Content Management Services.

2945 **8.7 Validate Content Protocol**

2946 The interface of a Content Validation Service MUST implement a single method called validateContent.
2947 The validateContent method accepts a ValidateContentRequest as parameter and returns a
2948 ValidateContentResponse as its response if there are no errors.

2949 The OriginalContent element within a ValidateContentRequest MUST contain exactly one
2950 RegistryObject that needs to be cataloged. The resulting ValidateContentResponse contains the status
2951 attribute that communicates whether the RegistryObject (and its content) are valid or not.

2952 The Validate Content protocol does not specify the implementation details of any specific Content
2953 Validation Service.



2954
2955

Figure 18: Validate Content Protocol

2956 **8.7.1 ValidateContentRequest**

2957 The ValidateContentRequest is used to pass content to a Content Validation Service so that it can
2958 validate the specified RegistryObject and any associated content. The RegistryObject typically is an
2959 ExternalLink (in the case of external content) or an ExtrinsicObject. The ValidateContentRequest
2960 extends the base type ContentManagementServiceRequestType.

2961 **8.7.1.1 Syntax:**

```
2962 <element name="ValidateContentRequest">  
2963   <complexType>  
2964     <complexContent>  
2965       <extension base="cms:ContentManagementServiceRequestType">  
2966         <sequence>  
2967         </sequence>  
2968       </extension>
```

2969
2970
2971

```
</complexContent>  
</complexType>  
</element>
```

2972

2973 **8.7.1.2 Parameters:**

2974 The following parameters are parameters that are either newly defined for this type or are inherited and
2975 have additional semantics beyond those defined in the base type description.

2976 *InvocationControlFile*: Inherited from base type. This parameter may not be present. If
2977 present its format is defined by the Content Validation Service.

2978 *OriginalContent*: Inherited from base type. This parameter MUST contain exactly one
2979 RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated
2980 content. This specification does not specify the format of the content. If it is an
2981 ExtrinsicObject then there MAY be a corresponding repository item as an attachment to
2982 this request that is the content. The corresponding repository item SHOULD follow the
2983 same rules as attachments in Submit/UpdateObjectsRequest.
2984

2985 **8.7.1.3 Returns:**

2986 This request returns a ValidateContentResponse. See section 8.7.2 for details.

2987 **8.7.1.4 Exceptions:**

2988 In addition to the exceptions returned by base request types, the following exceptions MAY be
2989 returned:

2990 *InvalidContentException*: signifies that the specified content was found to be invalid.
2991 The exception SHOULD include enough detail for the client to be able to determine
2992 how to make the content valid.
2993

2994 **8.7.2 ValidateContentResponse**

2995 The ValidateContentResponse is sent by the Content Validation Service as a response to a
2996 ValidateContentRequest.
2997

2998 **8.7.2.1 Syntax:**

2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009

```
<element name="ValidateContentResponse">  
  <complexType>  
    <complexContent>  
      <extension base="cms:ContentManagementServiceResponseType">  
        <sequence>  
        </sequence>  
      </extension>  
    </complexContent>  
  </complexType>  
</element>
```

3010 **8.7.2.2 Parameters:**

3011 The following parameters are parameters that are either newly defined for this type or are inherited and
3012 have additional semantics beyond those defined in the base type description.

3013 *status*: Inherited attribute. This enumerated value is used to indicate the status of the
3014 request. Values for status are as follows:

3015

- 3016 • Success - This status specifies that the content specified in the
3017 ValidateContentRequest was valid.
- 3018 • Failure - This status specifies that the request failed. If the error returned
3019 is an InvalidContentException then the content specified in the
3020 ValidateContentRequest was invalid. If there was some other failure
3021 encountered during the processing of the request then a different error
3022 MAY be returned.

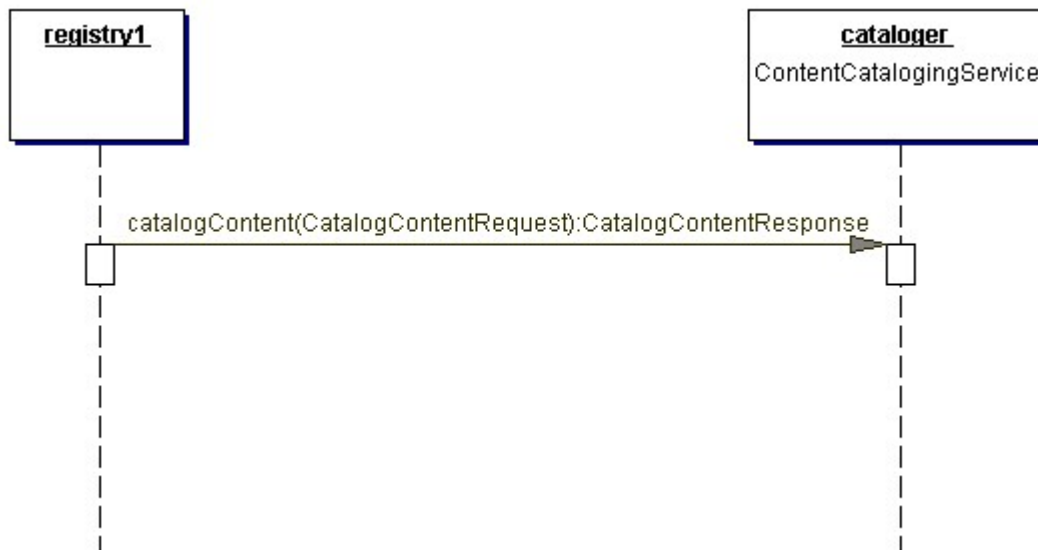
3023

3024 **8.8 Catalog Content Protocol**

3025 The interface of the Content Cataloging Service MUST implement a single method called
3026 catalogContent. The catalogContent method accepts a CatalogContentRequest as parameter and
3027 returns a CatalogContentResponse as its response if there are no errors.

3028 The CatalogContentRequest MAY contain repository items that need to be cataloged. The resulting
3029 CatalogContentResponse contains the metadata and possibly content that gets generated or updated
3030 by the Content Cataloging Service as a result of cataloging the specified repository items.

3031 The Catalog Content protocol does not specify the implementation details of any specific Content
3032 Cataloging Service.



3033

3034

Figure 19: Catalog Content Protocol

3035 **8.8.1 CatalogContentRequest**

3036 The CatalogContentRequest is used to pass content to a Content Cataloging Service so that it can
3037 create catalog metadata for the specified RegistryObject and any associated content. The
3038 RegistryObject typically is an ExternalLink (in case of external content) or an ExtrinsicObject. The
3039 CatalogContentRequest extends the base type ContentManagementServiceRequestType.

3040 **8.8.1.1 Syntax:**

3041

```
<element name="CatalogContentRequest">
```

3042
3043
3044
3045
3046
3047
3048
3049
3050

```
<complexType>
  <complexContent>
    <extension base="cms:ContentManagementServiceRequestType">
      <sequence>
      </sequence>
    </extension>
  </complexContent>
</complexType>
</element>
```

3051
3052

3053 **8.8.1.2 Parameters:**

3054 The following parameters are parameters that are either newly defined for this type or are inherited and
3055 have additional semantics beyond those defined in the base type description.

3056 *InvocationControlFile*: Inherited from base type. If present its format is defined by the
3057 Content Cataloging Service.

3058 *OriginalContent*: Inherited from base type. This parameter MUST contain exactly one
3059 RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated
3060 content. This specification does not specify the format of the content. If it is an
3061 ExtrinsicObject then there MAY be a corresponding repository item as an attachment to
3062 this request that is the content. The corresponding repository item SHOULD follow the
3063 same rules as attachments in Submit/UpdateObjectsRequest.
3064

3065 **8.8.1.3 Returns:**

3066 This request returns a CatalogContentResponse. See section 8.8.2 for details.

3067 **8.8.1.4 Exceptions:**

3068 In addition to the exceptions returned by base request types, the following exceptions MAY be
3069 returned:

3070 *CatalogingException*: signifies that an exception was encountered in the Cataloging
3071 algorithm for the service.
3072

3073 **8.8.2 CatalogContentResponse**

3074 The CatalogContentResponse is sent by the Content Cataloging Service as a response to a
3075 CatalogContentRequest.
3076

3077 **8.8.2.1 Syntax:**

3078
3079
3080
3081
3082
3083
3084
3085
3086
3087

```
<element name="CatalogContentResponse">
  <complexType>
    <complexContent>
      <extension base="cms:ContentManagementServiceResponseType">
        <sequence>
          <element name="CatalogedContent"
type="rim:RegistryObjectListType"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
```

3088
3089
3090

```
</complexType>  
</element>
```

3091 **8.8.2.2 Parameters:**

3092 The following parameters are parameters that are either newly defined for this type or are inherited and
3093 have additional semantics beyond those defined in the base type description.

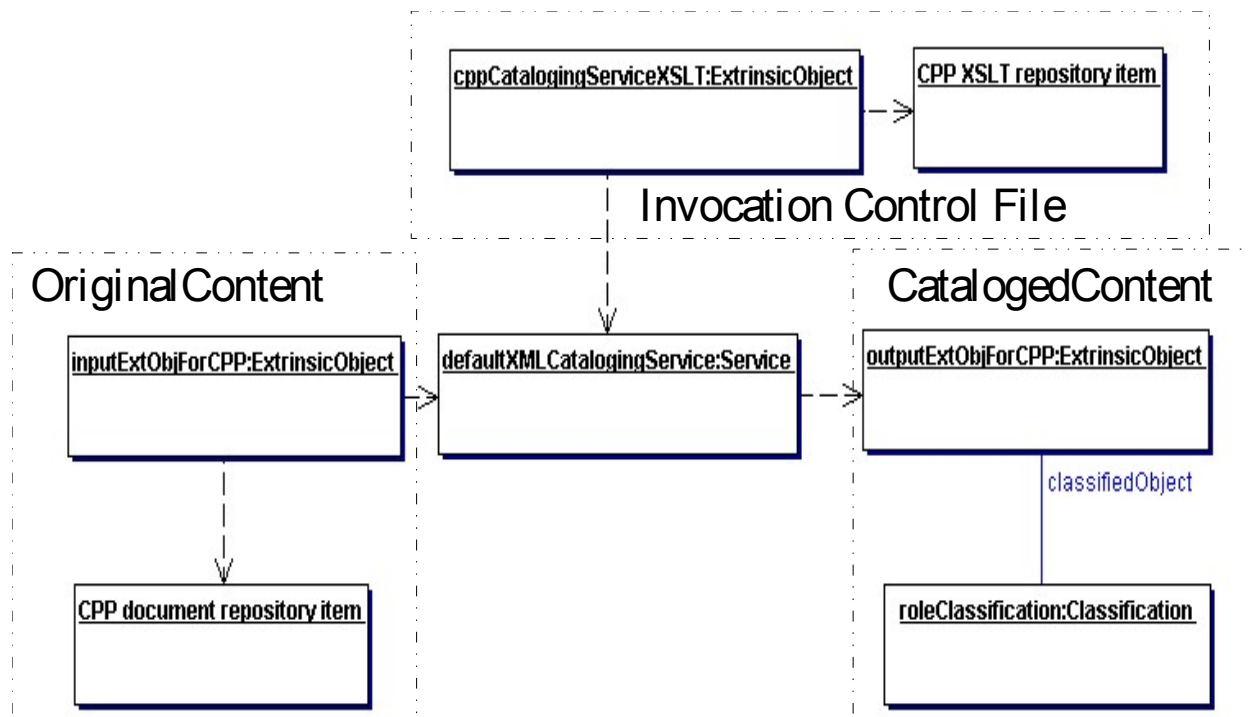
3094 *CatalogedContent:* This parameter specifies a collection of RegistryObject instances
3095 that were created or updated as a result of dynamic content cataloging by a content
3096 cataloging service. The Content Cataloging Service may add metadata such as
3097 Classifications, ExternalIdentifiers, name, description etc. to the CatalogedContent
3098 element. There MAY be an accompanying repository item as an attachment to this
3099 response message if the original repository item was modified by the request.

3100
3101

3102 **8.9 Illustrative Example: Canonical XML Cataloging Service**

3103 Figure 20 shows a UML instance diagram to illustrate how a Content Cataloging Service is used. This
3104 Content Cataloging Service is the normative Canonical XML Cataloging Service described in section
3105 8.10.

- 3106 ○ In the center we see a Content Cataloging Service name defaultXMLCataloger Service.
- 3107 ○ On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP being
3108 input as Original Content to the defaultXMLCataloging Service.
- 3109 ○ On top we see an XSLT style sheet repository item and its ExtrinsicObject that is configured as
3110 an Invocation Control File for the defaultXMLCataloger Service.
- 3111 ○ On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for the
3112 CPP. We also see a Classification roleClassification, which classifies the CPP by the Role
3113 element within the CPP. These are the Cataloged Content generated as a result of the
3114 Cataloging Service cataloging the CPP.



3115
3116 **Figure 20: Example of CPP cataloging using Canonical XML Cataloging Service**
3117

3118 **8.10 Canonical XML Content Cataloging Service**

3119 An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in
3120 service with the following constraints:

- 3121 • There is exactly one Service instance for the Canonical XML Content Cataloging Service
- 3122 • The Service is an XSLT engine
- 3123 • The Service may be invoked with exactly one Invocation Control File
- 3124 • The Original Content for the Service MUST be XML document(s)
- 3125 • The Cataloged Content for the Service MUST be XML document(s)
- 3126 • The Invocation Control File MUST be an XSLT style sheet
- 3127 • Each invocation of the Service MAY be with different Invocation Control File (XSLT style sheet)
3128 depending upon the objectType of the RegistryObject being cataloged. Each objectType
3129 SHOULD have its own unique XSLT style sheet. For example, ebXML CPP documents
3130 SHOULD have a specialized ebXML CPP Invocation Control XSLT style sheet.
- 3131 • The Service MUST have at least one input XML document that is a RegistryObject. Typically
3132 this is an ExtrinsicObject or an ExternalLink.
- 3133 • The Service MAY have at most one additional input XML document that is the content
3134 represented by the RegistryObject (e.g. a CPP document or an HL7 Conformance Profile). The
3135 optional second input MUST be referenced within the XSLT Style sheet by a using the
3136 “document” function with the document name specified by variable “repositoryItem” as in
3137 “document(\$repositoryItem).” A registry MUST define the variable “repositoryItem” when
3138 invoking the Canonical XML Cataloging Service.
- 3139 • The canonical XML Content Cataloging Service MUST apply the XSLT style sheet to the input
3140 XML instance document(s) in an XSLT transformation to generate the Cataloged Output.

3141 The Canonical XML Content Cataloging Service is a required normative feature of an ebXML Registry.

3142 **8.10.1 Publishing of Canonical XML Content Cataloging Service**

3143 An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in
3144 service. This built-in service MUST be published to the registry as part of the intrinsic bootstrapping of
3145 required canonical data within the registry.

9 Cooperating Registries Support

3146

3147 This chapter describes the capabilities and protocols that enable multiple ebXML registries to
3148 cooperate with each other to meet advanced use cases.

9.1 Cooperating Registries Use Cases

3149

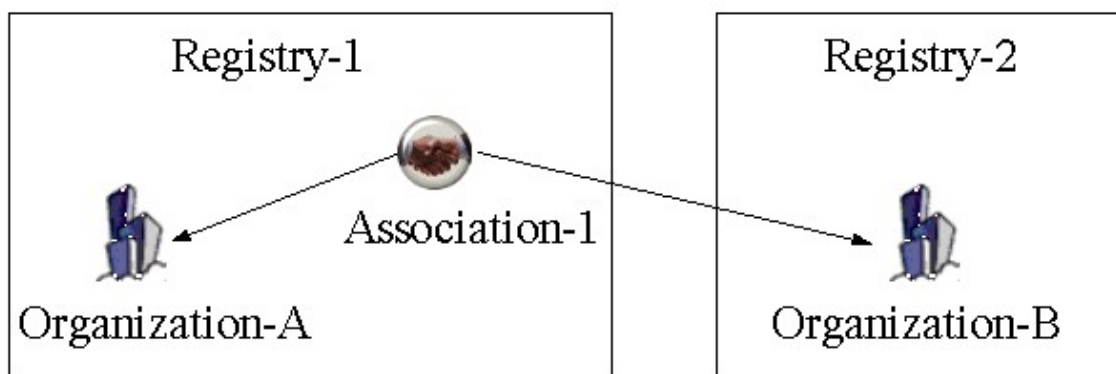
3150 The following is a list of use cases that illustrate different ways that ebXML registries cooperate with
3151 each other.

9.1.1 Inter-registry Object References

3152

3153 A Submitting Organization wishes to submit a RegistryObject to a registry such that the submitted
3154 object references a RegistryObject in another registry.

3155 An example might be where a RegistryObject in one registry is associated with a RegistryObject in
3156 another registry.



3157

3158

Figure 21: Inter-registry Object References

3159

9.1.2 Federated Queries

3160

3161 A client wishes to issue a single query against multiple registries and get back a single response that
3162 contains results based on all the data contained in all the registries. From the client's perspective it is
3163 issuing its query against a single logical registry that has the union of all data within all the physical
3164 registries.

9.1.3 Local Caching of Data from Another Registry

3165

3166 A destination registry wishes to cache some or all the data of another source registry that is willing to
3167 share its data. The shared dataset is copied from the source registry to the destination registry and is
3168 visible to queries on the destination registry even when the source registry is not available.

3169 Local caching of data may be desirable in order to improve performance and availability of accessing
3170 that object.

3171 An example might be where a RegistryObject in one registry is associated with a RegistryObject in
3172 another registry, and the first registry caches the second RegistryObject locally.

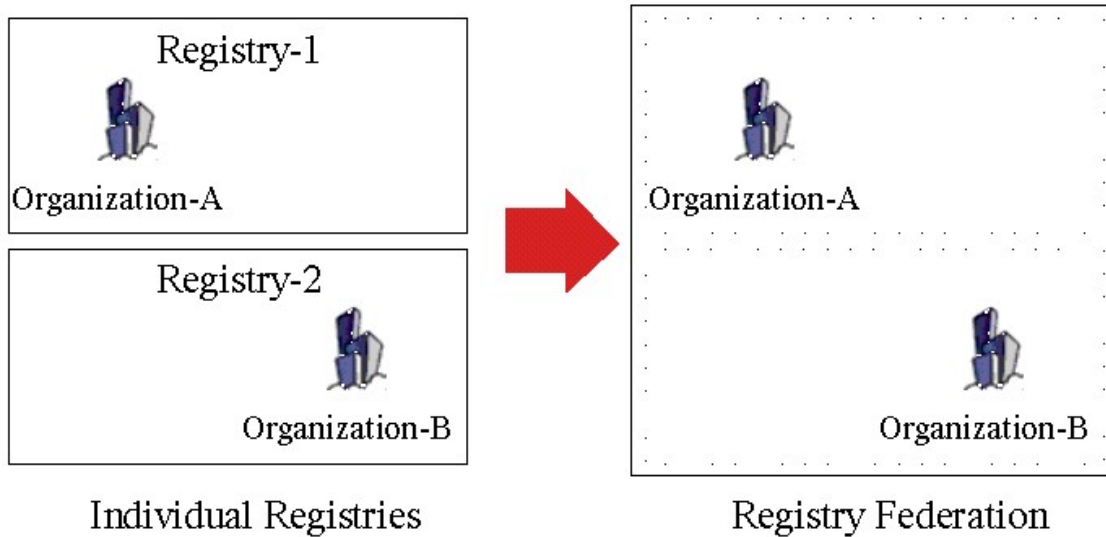
9.1.4 Object Relocation

3173

3174 A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from the
3175 registry where it was submitted to another registry.

3176 **9.2 Registry Federations**

3177 A registry federation is a group of registries that have voluntarily agreed to form a loosely coupled
3178 union. Such a federation may be based on common business interests and specialties that the
3179 registries may share. Registry federations appear as a single logical registry to registry clients.



3180
3181

Figure 22: Registry Federations

3182 Registry federations are based on a peer-to-peer (P2P) model where all participating registries are
3183 equal. Each participating registry is called a *registry peer*. There is no distinction between the registry
3184 operator that created a federation and those registry operators that joined that Federation later.

3185 Any registry operator MAY form a registry federation at any time. When a federation is created it MUST
3186 have exactly one registry peer which is the registry operated by the registry operator that created the
3187 federation.

3188 Any registry MAY choose to voluntarily join or leave a federation at any time.

3189 **9.2.1 Federation Metadata**

3190 The Registry Information model defines the Registry and Federation classes. Instances of these
3191 classes and the associations between these instances describe a federation and its members. Such
3192 instance data is referred to as Federation Metadata. The Registry and Federation classes are
3193 described in detail in [eBRIM].

3194 The Federation information model is summarized here as follows:

- 3195 ○ A Federation instance represents a registry federation.
- 3196 ○ A Registry instance represents a registry that is a member of the Federation.
- 3197 ○ An Association instance with associationType of *HasFederationMember* represents
3198 membership of the registry in the federation. This Association links the Registry instance and
3199 the Federation instance.

3200



3201
3202

Figure 23: Federation Metadata Example

3203 **9.2.2 Local Vs. Federated Queries**

3204 A federation appears to registry clients as a single unified logical registry. An AdhocQueryRequest sent
3205 by a client to a federation member MAY be local or federated. A new boolean attribute named
3206 *federated* is added to AdhocQueryRequest to indicate whether the query is federated or not.

3207 **9.2.2.1 Local Queries**

3208 When the federated attribute of AdhocQueryRequest has the value of *false* then the query is a local
3209 query. In the absence of a *federated* attribute the default value of *federated* attribute is *false*.

3210 A local AdhocQueryRequest is only processed by the registry that receives the request. A local
3211 AdhocQueryRequest does not operate on data that belongs to other registries.

3212 **9.2.2.2 Federated Queries**

3213 When the *federated* attribute of AdhocQueryRequest has the value of *true* then the query is a federated
3214 query.

3215 A federation member MUST route a federated query received by it to all other federation member
3216 registries on a best attempt basis. If a member is not reachable for any reason then it MAY be skipped.

3217 When a registry routes a federated query to other federation members it MUST set the federated
3218 attribute value to *false* and the *federation* attribute value to null to avoid infinite loops.

3219 A federated query operates on data that belongs to all members of the federation.

3220 When a client submits a federated query to a registry such that the query specifies no federation and
3221 no federations exist in the registry, then the registry MUST treat it as a local query.

3222 When a client submits a federated query that invokes a parameterized stored query, the registry MUST
3223 resolve the parameterized stored query into its non-stored form and MUST replace all variables with
3224 user-supplied parameters on registry supplied contextual parameters before routing it to a federation
3225 member.

3226 When a client submits a federated iterative query, the registry MUST use the *startIndex* attribute value
3227 of the original request as the *startIndex* attribute value of the routed request sent to each federation
3228 member. The response to the original request MUST be the *union* of the results from each routed

3229 query. In such cases the registry MUST return a *totalResultCount* attribute value on the federated query
3230 response to be equal to the *maximum* of all *totalResultCount* attribute values returned by each
3231 federation member.

3232 **9.2.2.3 Membership in Multiple Federations**

3233 A registry MAY be a member of multiple federations. In such cases if the *federated* attribute of
3234 AdhocQueryRequest has the value of *true* then the registry MUST route the federated query to *all*
3235 federations that it is a member of.

3236 Alternatively, the client MAY specify the id of a specific federation that the registry is a member of, as
3237 the value of the *federation* parameter. The type of the federation parameter is anyURI and identifies the
3238 "id" attribute of the desired Federation.

3239 In such cases the registry MUST route the federated query to the specified federation only.

3240 **9.2.3 Federated Lifecycle Management Operations**

3241 Details on how to create and delete federations and how to join and leave a federation are described in
3242 9.2.8.

3243 All lifecycle operations SHOULD be performed on a RegistryObject within its home registry using the
3244 operations defined by the LifecycleManager interface. Unlike query requests, lifecycle management
3245 requests do not support any federated capabilities.

3246 **9.2.4 Federations and Local Caching of Remote Data**

3247 A federation member is not required to maintain a local cache of replicas of RegistryObjects and
3248 repository items that belong to other members of the federation.

3249 A registry MAY choose to locally cache some or all data from any other registry whether that registry is
3250 a federation member or not. Data caching is orthogonal to registry federation and is described in
3251 section 9.3.

3252 Since by default there is minimal replication in the members of a federation, the federation architecture
3253 scales well with respect to memory and disk utilization at each registry.

3254 Data replication is often necessary for performance, scalability and fault-tolerance reasons.

3255 **9.2.5 Caching of Federation Metadata**

3256 A special case for local caching is the caching of the Federation and Registry instances and related
3257 Associations that define a federation and its members. Such data is referred to as federation metadata.
3258 A federation member is required to locally cache the federation metadata, from the federation home for
3259 each federation that it is a member of. The reason for this requirement is consistent with a Peer-to-
3260 Peer (P2P) model and ensures fault-tolerance in case the Federation home registry is unavailable.

3261 The federation member MUST keep the cached federation metadata synchronized with the master
3262 copy in the Federation home, within the time period specified by the replicationSyncLatency attribute of
3263 the Federation. Synchronization of cached Federation metadata may be done via synchronous polling
3264 or asynchronous event notification using the event notification feature of the registry.

3265 **9.2.6 Time Synchronization Between Registry Peers**

3266 Federation members are not required to synchronize their system clocks with each other. However,
3267 each Federation member SHOULD keep its clock synchronized with an atomic clock server within the
3268 latency described by the replicationSyncLatency attribute of the Federation.

3269 **9.2.7 Federations and Security**

3270 Federated operations abide by the same security rules as standard operations against a single registry.
3271 However, federation operations often require registry-to-registry communication. Such communication

3272 is governed by the same security rules as a Registry Client to registry communication. The only
3273 difference is that the requesting registry plays the role of Registry Client. Such registry-to-registry
3274 communication SHOULD be conducted over a secure channel such as HTTP/S. Federation members
3275 SHOULD be part of the same SAML Federation if member registries implement the Registry SAML
3276 Profile described in chapter 11.

3277 **9.2.8 Federation Lifecycle Management Protocols**

3278 This section describes the various operations that manage the lifecycle of a federation and its
3279 membership. Federation lifecycle operations are done using standard LifecycleManager interface of
3280 the registry in a stylized manner. Federation lifecycle operations are privileged operations. A registry
3281 SHOULD restrict Federation lifecycle operations to registry User's that have the RegistryAdministrator
3282 role.

3283 **9.2.8.1 Joining a Federation**

3284 The following rules govern how a registry joins a federation:

- 3285 • Each registry SHOULD have exactly one Registry instance within that registry for which it is a
3286 home. The Registry instance is owned by the RegistryOperator and may be placed in the
3287 registry using any operator specific means. The Registry instance SHOULD never change its
3288 home registry.
- 3289 • A registry MAY request to join an existing federation by submitting an instance of an
3290 Extramural Association that associates the Federation instance as sourceObject, to its Registry
3291 instance as targetObject, using an associationType of *HasFederationMember*. The home
3292 registry for the Association and the Federation objects MUST be the same.

3293

3294 **9.2.8.2 Creating a Federation**

3295 The following rules govern how a federation is created:

- 3296 • A Federation is created by submitting a Federation instance to a registry using
3297 SubmitObjectsRequest.
- 3298 • The registry where the Federation is submitted is referred to as the federation home.
- 3299 • The federation home may or may not be a member of that Federation.
- 3300 • A federation home MAY contain multiple Federation instances.

3301 **9.2.8.3 Leaving a Federation**

3302 The following rules govern how a registry leaves a federation:

3303 A registry MAY leave a federation at any time by removing its *HasFederationMember* Association
3304 instance that links it with the Federation instance. This is done using the standard
3305 RemoveObjectsRequest.

3306 **9.2.8.4 Dissolving a Federation**

3307 The following rules govern how a federation is dissolved:

- 3308 • A federation is dissolved by sending a RemoveObjectsRequest to its home registry and
3309 removing its Federation instance.
- 3310 • The removal of a Federation instance is controlled by the same Access Control Policies that
3311 govern any RegistryObject.
- 3312 • The removal of a Federation instance is controlled by the same lifecycle management rules
3313 that govern any RegistryObject. Typically, this means that a federation MUST NOT be
3314 dissolved while it has federation members. It MAY however be deprecated at any time. Once a

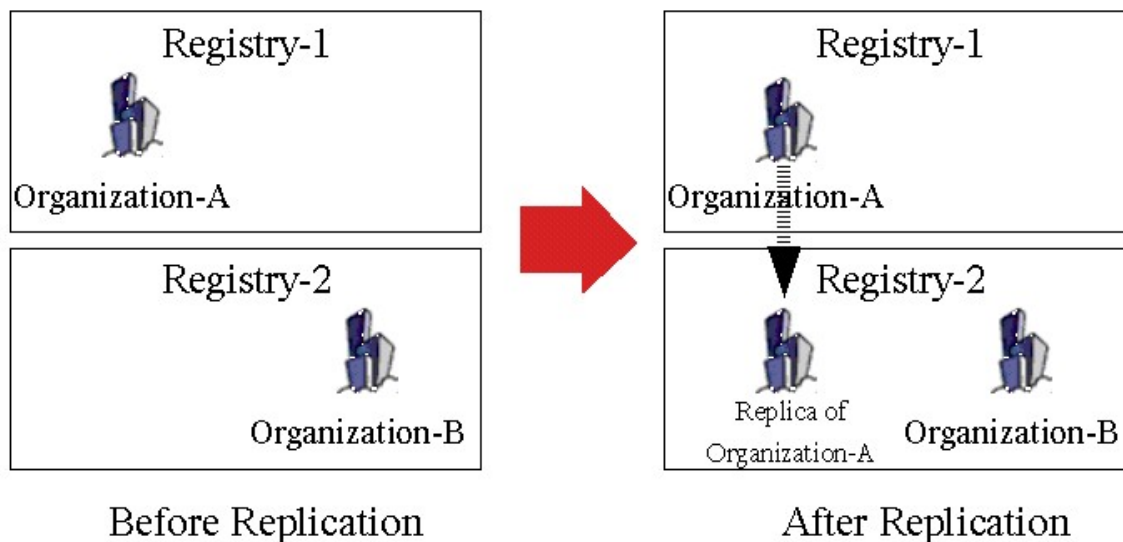
3315 Federation is deprecated no new members can join it.
3316

3317 9.3 Object Replication

3318 RegistryObjects within a registry MAY be replicated in another registry. A replicated copy of a remote
3319 object is referred to as its replica. The remote object MAY be an original object or it MAY be a replica.
3320 A replica from an original is referred to as a first-generation replica. A replica of a replica is referred to
3321 as a second-generation replica (and so on).

3322 The registry that replicates a remote object locally is referred to as the destination registry for the
3323 replication. The registry that contains the remote object being replicated is referred to as the source
3324 registry for the replication.

3325



3326

3327

3328

Figure 24: Object Replication

3329 9.3.1 Use Cases for Object Replication

3330 A registry MAY create a local replica of a remote object for a variety of reasons. A few sample use
3331 cases follow:

- 3332 ○ Improve access time and fault tolerance by locally caching remote objects. For example, a
3333 registry MAY automatically create a local replica when a remote ObjectRef is submitted to the
3334 registry.
- 3335 ○ Improve scalability by distributing access to hotly contested objects, such as NAICS scheme,
3336 across multiple replicas.
- 3337 ○ Enable cooperating registry features such as hierarchical registry topology and local caching of
3338 federation metadata.

3339 9.3.2 Queries And Replicas

3340 A registry MUST support client queries to consider a local replica of remote object as if it were a local
3341 object. Local replicas are considered within the extent of the data set of a registry as far as local
3342 queries are concerned.

3343 When a client submits a local query that retrieves a remote object by its id attribute, if the registry
3344 contains a local replica of that object then the registry SHOULD return the state defined by the local

3345 replica.

3346 **9.3.3 Lifecycle Operations And Replicas**

3347 LifeCycle operations on an original object **MUST** be performed at the home registry for that object.
3348 LifeCycle operations on replicas of an original object should result in an `InvalidRequestException`.

3349 **9.3.4 Object Replication and Federated Registries**

3350 Object replication capability is orthogonal to the registry federation capability. Objects **MAY** be
3351 replicated from any registry to any other registry without any requirement that the registries belong to
3352 the same federation.

3353 **9.3.5 Creating a Local Replica**

3354 Any Submitting Organization can create a replica by using the standard `SubmitObjectsRequest`. If a
3355 registry receives a `SubmitObjectsRequest` that has a `RegistryObjectList` containing a remote
3356 `ObjectRef`, then it **MUST** create a replica for that remote `ObjectRef`. In such cases the User that
3357 submitted the `ObjectRef` (via a `SubmitObjectsRequest`) owns the replica while the original
3358 `RegistryObject` is owned by its original owner.

3359 In addition to Submitting Organizations, a registry itself **MAY** create a replica under specific situations
3360 in a registry specific manner.

3361 Creating a local replica requires the destination registry to read the state of the remote object from the
3362 source registry and then create a local replica of the remote object.

3363 A registry **SHOULD** use standard `QueryManager` interface to read the state of a remote object (whether
3364 it is an original or a replica). No new APIs are needed to read the state of a remote object. Since query
3365 functionality does not need prior registration, no prior registration or contract is needed for a registry to
3366 read the state of a remote object.

3367 Once the state of the remote object has been read, a registry **MAY** use registry specific means to
3368 create a local replica of the remote object. Such registry specific means **MAY** include the use of the
3369 `LifeCycleManager` interface.

3370 A replica of a `RegistryObject` may be distinguished from an original since a replica **MUST** have its
3371 home attribute point to the remote registry where the original for the replica resides.

3372 **9.3.6 Transactional Replication**

3373 Transactional replication enables a registry to replicate events in another registry in a transactionally
3374 consistent manner. This is typically the case when entire registries are replicated to another registry.

3375 This specification defines a more loosely coupled replication model as an alternative to transactional
3376 replication for the following reasons:

- 3377 • Transactional replication requires a tight coupling between registries participating in the
3378 replication
- 3379 • Transactional replication is not a typical use case for registries
- 3380 • Loosely coupled replication as defined by this specification typically suffices for most use cases
- 3381 • Transaction replication is very complex and error prone

3382

3383 Registry implementations are not required to implement transactional replication.

3384 **9.3.7 Keeping Replicas Current**

3385 A registry **MUST** keep its replicas current within the latency specified by the value of the
3386 `replicationSyncLatency` attribute defined by the registry. This includes removal of the replica when its

3387 original is removed from its home registry.

3388 Replicas MAY be kept current using the event notification feature of the registry or via periodic polling.

3389 **9.3.8 Lifecycle Management of Local Replicas**

3390 Local Replicas are read-only objects. Lifecycle management actions are not permitted on local replicas
3391 with the exception of the Delete action which is used to remove the replica. All other lifecycle
3392 management actions MUST be performed on the original RegistryObject in the home registry for the
3393 object.

3394 **9.3.9 Tracking Location of a Replica**

3395 A local replica of a remote RegistryObject instance MUST have exactly one ObjectRef instance within
3396 the local registry. The home attribute of the ObjectRef associated with the replica tracks its home
3397 location. A RegistryObject MUST have exactly one home. The home for a RegistryObject MAY change
3398 via Object Relocation as described in section 9.4. It is optional for a registry to track location changes
3399 for replicas within it.

3400 **9.3.10 Remote Object References to a Replica**

3401 It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another
3402 RegistryObject. In such cases the home attribute of the ObjectRef contains the base URI to the home
3403 registry for the replica.

3404 **9.3.11 Removing a Local Replica**

3405 A client can remove a replica by using the RemoveObjectsRequest. If a registry receives a
3406 RemoveObjectsRequest that has an ObjectRefList containing a remote ObjectRef, then it MUST
3407 remove the local replica for that remote ObjectRef assuming that the client was authorized to remove
3408 the replica.

3409 **9.4 Object Relocation Protocol**

3410 Every RegistryObject has a home registry and a User within the home registry that is the Submitter or
3411 owner of that object. Initially, the home registry is the where the object is originally submitted. Initially,
3412 the owner is the User that submitted the object.

3413 A RegistryObject MAY be relocated from one home registry to another home registry using the Object
3414 Relocation protocol.

3415 Within the Object Relocation protocol, the new home registry is referred to as the *destination* registry
3416 while the previous home registry is called the *source* registry.

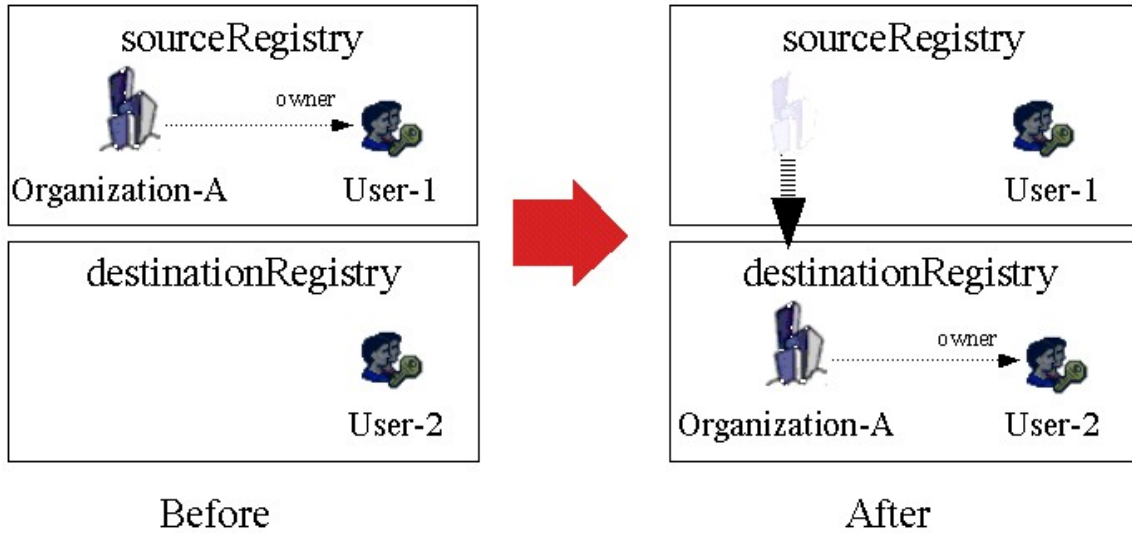


Figure 25: Object Relocation

3417
3418

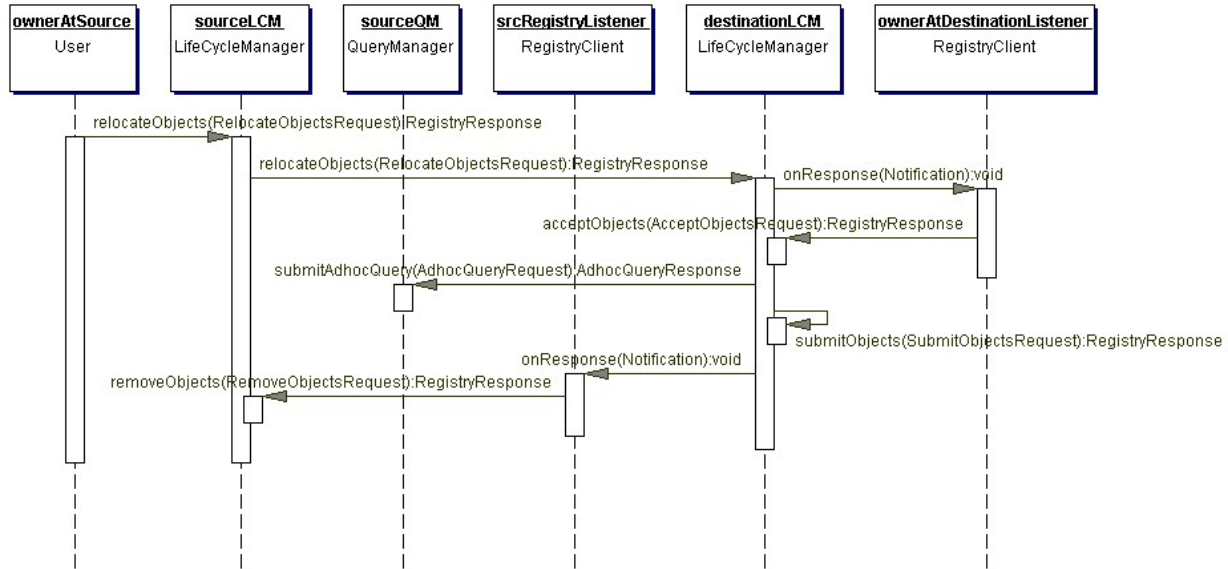
3419 The User at the source registry who owns the objects being relocated is referred to as the
3420 *ownerAtSource*. The User at the destination registry, who is the new owner of the objects, is referred to
3421 as the *ownerAtDestination*. While the *ownerAtSource* and the *ownerAtDestination* may often be the
3422 same, the Object Relocation protocol treats them as two distinct identities.

3423 A special case usage of the Object Relocation protocol is to transfer ownership of RegistryObjects from
3424 one User to another within the same registry. In such cases the protocol is the same except for the fact
3425 that the source and destination registries are the same.

3426 Following are some notable points regarding object relocation:

- 3427 • Object relocation does not require that the source and destination registries be in the same
3428 federation or that either registry have a prior contract with the other.
- 3429 • Object relocation MUST preserve object id. While the home registry for a RegistryObject MAY
3430 change due to object relocation, its id never changes.
- 3431 • ObjectRelocation MUST preserve referential integrity of RegistryObjects. Relocated objects
3432 that have references to an object that did not get relocated MUST preserve their reference.
3433 Similarly objects that have references to a relocated object MUST also preserve their
3434 reference. Thus, relocating an object may result in making the value of a reference attribute go
3435 from being a local reference to being a remote reference or vice versa.
- 3436 • AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque transactionId
3437 identifying the relocateObjects transaction.
- 3438 • The requests defined by the Relocate Objects protocol MUST be sent to the source or
3439 destination registry only.
- 3440 • When an object is relocated an AuditableEvent of type "Relocated" MUST be recorded by the
3441 sourceRegistry. Relocated events MUST have the source and destination registry's base URIs
3442 recorded as two Slots on the Relocated event. The names of these Slots are:
 - 3443 ○ urn:oasis:names:tc:ebxml-regrep:rs:events:sourceRegistry
 - 3444 ○ urn:oasis:names:tc:ebxml-regrep:rs:events:destinationRegistry

3445



3446
3447 **Figure 26: Relocate Objects Protocol**

3448 Figure 26 illustrates the Relocate Objects Protocol. The participants in the protocol are the
3449 ownerAtSource and ownerAtDestination User instances as well as the LifeCycleManager interfaces of
3450 the sourceRegistry and destinationRegistry.

3451 The steps in the protocol are described next:

- 3452 1. The protocol is initiated by the ownerAtSource sending a RelocateObjectsRequest message to
3453 the LifeCycleManager interface of the sourceRegistry. The sourceRegistry MUST make sure
3454 that the ownerAtSource is authorized to perform this request. The id of this
3455 RelocateObjectsRequest is used as the transaction identifier for this instance of the protocol.
3456 This RelocateObjectsRequest message MUST contain an ad hoc query that specifies the
3457 objects that are to be relocated.
- 3458 2. Next, the sourceRegistry MUST relay the same RelocateObjectsRequest message to the
3459 LifeCycleManager interface of the destinationRegistry. This message enlists the
3460 destinationRegistry to participate in relocation protocol. The destinationRegistry MUST store
3461 the request information until the protocol is completed or until a registry specific period after
3462 which the protocol times out.
- 3463 3. The destinationRegistry MUST relay the RelocateObjectsRequest message to the
3464 ownerAtDestination. This notification MAY be done using the event notification feature of the
3465 registry as described in chapter 7. The notification MAY be done by invoking a listener Service
3466 for the ownerAtDestination or by sending an email to the ownerAtDestination. This concludes
3467 the first phase of the Object Relocation protocol.
- 3468 4. The ownerAtDestination at a later time MAY send an AcceptObjectsRequest message to the
3469 destinationRegistry. This request MUST identify the object relocation transaction via the
3470 *correlationId*. The value of this attribute MUST be the id of the original
3471 RelocateObjectsRequest.
- 3472 5. The destinationRegistry sends an AdhocQueryRequest message to the sourceRegistry. The
3473 source registry returns the objects being relocated as an AdhocQueryResponse. In the event of
3474 a large number of objects this may involve multiple AdhocQueryRequest/responses as
3475 described by the iterative query feature described in section 6.2.
- 3476 6. The destinationRegistry submits the relocated data to itself assigning the identity of the
3477 ownerAtDestination as the owner. The relocated data MAY be submitted to the destination
3478 registry using any registry specific means or a SubmitObjectsRequest. However, the effect
3479 SHOULD be the same as if a SubmitObjectsRequest was used.

- 3480 7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been safely
3481 committed using the Event Notification feature of the registry as described in chapter 7.
- 3482 8. The sourceRegistry removes the relocated objects using any registry specific means and
3483 logging an AuditableEvent of type Relocated. This concludes the Object Relocation
3484 transaction.

3485 9.4.1 RelocateObjectsRequest

```
3486 <element name="RelocateObjectsRequest">  
3487   <complexType>  
3488     <complexContent>  
3489       <extension base="rs:RegistryRequestType">  
3490         <sequence>  
3491           <element name="Query" type="rim:AdhocQueryType"/>  
3492           <element name="SourceRegistry" type="rim:ObjectRefType"/>  
3493           <element name="DestinationRegistry"  
3494 type="rim:ObjectRefType"/>  
3495           <element name="OwnerAtSource" type="rim:ObjectRefType"/>  
3496           <element name="OwnerAtDestination"  
3497 type="rim:ObjectRefType"/>  
3498         </sequence>  
3499       </extension>  
3500     </complexContent>  
3501   </complexType>  
3502 </element>
```

3503

3504 9.4.1.1 Parameters:

- 3505 *id*: the attribute id provides the transaction identifier for this instance of the protocol.
- 3506 *AdhocQuery*: This element specifies an ad hoc query that selects the RegistryObjects that are
3507 being relocated.
- 3508 *sourceRegistry*: This element specifies the ObjectRef to the sourceRegistry Registry instance.
3509 The value of this attribute MUST be a local reference when the message is sent by the
3510 ownerAtSource to the sourceRegistry.
- 3511 *destinationRegistry*: This element specifies the ObjectRef to the destinationRegistry Registry
3512 instance.
- 3513 *ownerAtSource*: This element specifies the ObjectRef to the ownerAtSource User instance.
- 3514 *ownerAtDestination*: This element specifies the ObjectRef to the ownerAtDestination User
3515 instance.
- 3516

3517 9.4.1.2 Returns:

3518 This request returns a RegistryResponse. See section 2.1.4 for details.

3519 9.4.1.3 Exceptions:

- 3520 In addition to the exceptions common to all requests, the following exceptions MAY be returned:
- 3521 *ObjectNotFoundException*: signifies that the specified Registry or User was not found in
3522 the registry.
- 3523

3524 9.4.2 AcceptObjectsRequest

```
3525 <element name="AcceptObjectsRequest">
```

3526
3527
3528
3529
3530
3531
3532
3533
3534
3535

```
<complexType>
  <complexContent>
    <extension base="rs:RegistryRequestType">
      <attribute name="correlationId" use="required"
type="{http://www.w3.org/2001/XMLSchema}anyURI" />
    </extension>
  </complexContent>
</complexType>
</element>
```

3536 **9.4.2.1 Parameters:**

3537 *correlationId*: Provides the transaction identifier for this instance of the protocol.
3538

3539 **9.4.2.2 Returns:**

3540 This request returns a RegistryResponse. See section 2.1.4 for details.

3541 **9.4.2.3 Exceptions:**

3542 In addition to the exceptions common to all requests, the following exceptions MAY be returned:
3543 *InvalidRequestException*: signifies that the specified correlationId was not found to
3544 match an ongoing RelocateObjectsRequest in the registry.
3545

3546 **9.4.3 Object Relocation and Remote ObjectRefs**

3547 The following scenario describes what typically happens when a person moves:

- 3548 1. When a person moves from one house to another, other persons may have their old postal
3549 addresses.
- 3550 2. When a person moves, they leave their new address as the forwarding address with the post
3551 office.
- 3552 3. The post office forwards their mail for some time to their new address.
- 3553 4. Eventually the forwarding request expires and the post office no longer forwards mail for that
3554 person.
- 3555 5. During this forwarding interval the person notifies interested parties of their change of address.

3556 The Object Relocation feature supports a similar model for relocation of RegistryObjects. The following
3557 steps describe the expected behavior when an object is relocated.

- 3558 1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other
3559 RegistryObjects may have remote ObjectRefs to O1.
- 3560 2. The registry R1 MUST create an AuditableEvent of type Relocated that includes the home URI
3561 for the new registry R2.
- 3562 3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it MUST
3563 forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the client. The
3564 object O1 MUST include the home URI to R2 within the optional home attribute of
3565 RegistryObject. Clients are advised to check the home attribute and update the home attribute
3566 of their local ObjectRef to match the new home URI value for the object.
- 3567 4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no longer
3568 required to relay requests for O1 to R2 transparent to the client. Instead R1 MUST return an
3569 ObjectNotFoundException.

- 3570 5. Clients that are interested in the relocation of O1 and being notified of its new address may
3571 choose to be notified by having a prior subscription using the event notification facility of the
3572 registry. For example a Registry that has a remote ObjectRefs to O1 may create a subscription
3573 on relocation events for O1. This however, is not required behavior.

3574 **9.4.4 Notification of Object Relocation To ownerAtDestination**

3575 This section describes how the destinationRegistry uses the event notification feature of the registry to
3576 notify the ownerAtDestination of a Relocated event.

3577 The destinationRegistry MUST send a Notification with the following required characteristics:

- 3578 • The notification MUST be an instance of a Notification element.
- 3579 • The Notification instance MUST have at least one Slot as follows:
 - 3580 ○ The Slot MUST have the name:
3581 urn:oasis:names:tc:ebxml-regrep:rs:events:correlationId
 - 3582 ○ The Slot MUST have the correlationId for the Object Relocation transaction as the
3583 value of the Slot.

3584

3585 **9.4.5 Notification of Object Commit To sourceRegistry**

3586 This section describes how the destinationRegistry uses the event notification feature of the registry to
3587 notify the sourceRegistry that it has completed committing the relocated objects.

3588 The destinationRegistry MUST send a Notification with the following required characteristics:

- 3589 • The notification MUST be an instance of a Notification element.
- 3590 • The Notification instance MUST have at least one Slot as follows:
 - 3591 ○ The Slot MUST have the name
3592 urn:oasis:names:tc:ebxml-regrep:rs:events:objectsCommitted
 - 3593 ○ The Slot MUST have the value of *true*.

3594

3595 **9.4.6 Object Ownership and Owner Reassignment**

3596 A registry MUST determine the ownership of a RegistryObject based upon the most recent
3597 AuditableEvent that has the eventType matching the canonical EventType ClassificationNode for
3598 Create or Relocate events.

3599 A special case of Object Relocation is when an ObjectRelocationRequest to a registry specifies the
3600 same registry as sourceRegistry and destinationRegistry. In such cases the request is effectively to
3601 change the owner of the specified objects from current owner to a new owner.

3602 In such case if the client does not have the RegistryAdministrator role then the protocol requires the
3603 ownerAtDestination to issue an AcceptObjectsRequest as described earlier.

3604 However, if the client does have the RegistryAdministrator role then the registry MUST change the
3605 owner of the object to the user specified as ownerAtDestination without the ownerAtDestination to
3606 issue an AcceptObjectsRequest.

3607 **9.4.7 Object Relocation and Timeouts**

3608 No timeouts are specified for the Object Relocation protocol. Registry implementations MAY cleanup
3609 incomplete Object Relocation transactions in a registry specific manner as an administrative task using
3610 registry specific policies.

3611

3612 **10 Registry Security**

3613 This chapter describes the security features of ebXML Registry. A glossary of security terms can be
3614 referenced from [RFC 2828]. The registry security specification incorporates by reference the following
3615 specifications:

- 3616 • [WSI-BSP] WS-I Basic Security Profile 1.0
- 3617 • [WSS-SMS] Web Services Security: SOAP Message Security 1.0
- 3618 • [WSS-SWA] Web Services Security: SOAP Messages with Attachments (SwA) Profile 1.0

3619 This chapter provides registry specific details not present in above specifications.

3620 **10.1 Security Use Cases**

3621 This section describes various use cases that require security features from the registry. Subsequent
3622 sections describe specific registry mechanisms that enable each of these use cases.

3623 **10.1.1 Identity Management**

3624 An organization deploys an ebXML Registry and needs to define the set of users and services that are
3625 authorized to use the services offered by the registry. They require that the registry provide some
3626 mechanism for registering and subsequently managing the identity and credentials associated with
3627 such authorized users and services.

3628 **10.1.2 Message Security**

3629 A Registered User sends a request message to the registry and receives a response back from the
3630 registry. The user requires that the message integrity be protected during transmission from tampering
3631 (man-in-the-middle attack). The user may also require that the message communication is not
3632 available to unauthorized parties (confidentiality).

3633 **10.1.3 Repository Item Security**

3634 A Registered User submits a repository item to the registry. The user requires that the registry provide
3635 mechanisms to protect the integrity of the repository item during transmission on the wire and as long
3636 as it is stored in the registry. The user may also require that the content of the RepositoryItem is not
3637 available to unauthorized parties (confidentiality).

3638 **10.1.4 Authentication**

3639 An organization that deploys an ebXML Registry requires that when a Registered User sends a request
3640 to the registry, the registry checks the credentials provided by the user to ensure that the user is a
3641 Registered User and to unambiguously determine the user's identity.

3642 **10.1.5 Authorization and Access Control**

3643 An organization that deploys an ebXML Registry requires that the registry provide a mechanism that
3644 protect its resources from unauthorized access. Specifically, when a Registry Requestor sends a
3645 request to the registry, the registry restricts the actions of the requestor to specific actions on specific
3646 resources for which the requestor is authorized.

3647 **10.1.6 Audit Trail**

3648 An organization that deploys an ebXML Registry requires that the registry keep a journal or Audit Trail
3649 of all significant actions performed by Registry Requestors on registry resources. This provides a basic
3650 form of non-repudiation where a Registry Requestor cannot repudiate that they performed actions
3651 that are logged in the Audit Trail.

3652 **10.2 Identity Management**

3653 An ebXML Registry **MUST** provide an Identity Management mechanism that allows identities and
3654 credentials to be registered for authorized users of the registry and subsequently managed.

3655 If a registry implements the Registry SAML Profile as described in chapter 11 then the Identity
3656 Management capability **MUST** be provided by an Identity Provider service that integrates with the
3657 registry using the SAML 2.0 protocols as defined by [SAMLCore].

3658 If a registry does not implement the Registry SAML Profile then it **MUST** provide User Registration and
3659 Identity Management functionality in an implementation specific manner.

3660 **10.3 Message Security**

3661 A registry **MUST** provide mechanisms to securely exchange messages between a Registry Requestor
3662 and the registry to ensure data and source integrity as described in this section.

3663 **10.3.1 Transport Layer Security**

3664 A registry **MUST** support HTTP/S communication between an HTTP Requestor and its HTTP interface
3665 binding. A registry **MUST** also support HTTP/S communication between a SOAP Requestor and its
3666 SOAP interface binding when the underlying transport protocol is HTTP.

3667 HTTP/S support **SHOULD** allow for both SSL and TLS as transport protocols.

3668 **10.3.2 SOAP Message Security**

3669 A registry **MUST** support signing and verification of all registry protocol messages (requests and
3670 responses) between a SOAP Requestor and its SOAP binding. Such mechanisms **MUST** conform to
3671 [WSI-BSP], [WSS-SMS], [WSS-SWA] and [XMLDSIG]. The reader should refer to these specifications
3672 for details on these message security mechanisms.

3673 **10.3.2.1 Request Message Signature**

3674 When a Registered User sends a request message to the registry, the requestor **SHOULD** sign the
3675 request message with a Message Signature. This ensures the integrity of the message and also
3676 enables the registry to perform authentication and authorization for the request. If the registry receives
3677 a request that does not include a Message signature then it **MUST** implicitly treat the request as
3678 coming from a Registry Guest. A Registered User need not sign a request message with a Message
3679 Signature when the SOAP communication is conducted over HTTP/S as the message security is
3680 handled by the transport layer security provided by HTTP/S in this case.

3681 When a Registered User sends a request message to the registry that contains a RepositoryItem as a
3682 SOAP Attachment, the requestor **MUST** also reference and sign the RepositoryItem from the message
3683 signature. This **MUST** conform to [RFC2392] and [WSS-SWA].

3684 If the registry receives a request containing an unsigned RepositoryItem then it **MUST** return an
3685 UnsignedRepositoryItemException.

3686 **10.3.2.2 Response Message Signature**

3687 When a Registered User sends a request message to the registry, the registry **MAY** use a pre-
3688 established preference policy or a default policy to determine whether the response message **SHOULD**
3689 be signed with a Message Signature. When a Registry Guest sends a request, the Registration
3690 Authority **MAY** use a default policy to determine whether the response contains a header signature. A
3691 registry need not sign a response message with a Message Signature when the SOAP communication
3692 is conducted over HTTP/S as the message security is handled by the transport layer security provided
3693 by HTTP/S in this case.

3694 When a registry sends a signed response message to a Registry Client that contains a RepositoryItem
3695 as a SOAP Attachment, the registry **MUST** also reference and sign the RepositoryItem from the
3696 message signature. This **MUST** conform to [RFC2392] and [WSS-SWA].

3697 If the Registry Client receives a signed response with a RepositoryItem that does not include a
3698 RepositoryItem Signature then it SHOULD not trust the integrity of the response and treat it as an error
3699 condition.

3700 10.3.2.3 KeyInfo Requirements

3701 The sender of a registry protocol message (Registry Requestor and Registry) SHOULD provide their
3702 public key under the <wsse:Security> element. If provided, it MUST be contained in a
3703 <wsse:BinarySecurityToken> element and MUST be referenced from the <ds:KeyInfo> element in the
3704 Message Signature. The value of wsu:Id attribute of the <wsse:BinarySecurityToken> containing the
3705 senders public key MUST be **urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert**.
3706 The <wsse:BinarySecurityToken> SHOULD contain a X509 Certificate.

3707 Listing 3 shows an example of Message signature including specifying the KeyInfo.

3708 10.3.2.4 Message Signature Validation

3709 Signature validation ensures message and attached RepositoryItems integrity and security, concerning
3710 both data and source.

3711 If the registry receives a request containing a Message Signature then it MUST validate the Message
3712 Signature as defined by [WSS-SMS]. In case the request contains an attached RepositoryItem it MUST
3713 validate the RepositoryItems signature as defined by [WSS-SWA].

3714 If the Registry Requestor receives a response containing a Message Signature then it SHOULD
3715 validate the Message Signature as defined by [WSS-SMS]. In case the response contains an attached
3716 RepositoryItem then it SHOULD validate the RepositoryItem signature as defined by [WSS-SWA].

3717 10.3.2.5 Message Signature Example

3718 The following example shows the format of a Message Signature:

```
3719 <soap:Envelope>  
3720 <soap:Header>  
3721 <wsse:Security>  
3722 <wsse:BinarySecurityToken EncodingType="http://docs.oasis-  
3723 open.org/wss/2004/01/oasis-200401-wss-soap-message-security-  
3724 1.0#Base64Binary" ValueType="http://docs.oasis-  
3725 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"  
3726 wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">  
3727 lui+Jy4WYKJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnW  
3728 SWkXm9jAEEdsm/  
3729 hs+f3NwvK23bh46mNmncQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yT  
3730 cI7XU7xZT54S9  
3731 hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrsFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI  
3732 9WzxPCfHdalN4  
3733 rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUC9QY3VjwNALgGDaEAT7gpURkCI8  
3734 5HjdnSA5SM4cY  
3735 7jAsYX/CiPEkRJcBULLTEFrBZIBYDPzRWLSdsJRJngF7yCoGWJ+/HYOyP8P4OM5  
3736 9FDi0kM8GwOE0  
3737 WgYrJHH92qaVhoiPTLi7  
3738 </wsse:BinarySecurityToken>  
3739 <ds:Signature>  
3740 <!--The Message Signature -->  
3741 <ds:SignedInfo>  
3742 <ds:CanonicalizationMethod  
3743 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">  
3744 <c14n:InclusiveNamespaces PrefixList="wsse soap"  
3745 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />  
3746 </ds:CanonicalizationMethod>  
3747 <ds:SignatureMethod  
3748 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />  
3749 <ds:Reference URI="#TheBody">
```

```

3750         <ds:Transforms>
3751             <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
3752 exc-c14n#">
3753                 <c14n:InclusiveNamespaces PrefixList=""
3754 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3755             </ds:Transform>
3756         </ds:Transforms>
3757         <ds:DigestMethod
3758 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
3759         <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValu
3760 e>
3761     </ds:Reference>
3762 </ds:SignedInfo>
3763 <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureVa
3764 lue>
3765     <ds:KeyInfo>
3766         <wsse:SecurityTokenReference>
3767             <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
3768 regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
3769 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
3770         </wsse:SecurityTokenReference>
3771     </ds:KeyInfo>
3772 </ds:Signature>
3773 </wsse:Security>
3774 </soap:Header>
3775 <soap:Body wsu:Id="TheBody">
3776     <lcm:SubmitObjectsRequest/>
3777 </soap:Body>
3778 </soap:Envelope>

```

Listing 3: Message Signature Example

10.3.2.6 Message With RepositoryItem: Signature Example

The following example shows the format of a Message Signature that also signs the attached RepositoryItem:

```

3784 Content-Type: multipart/related; boundary="BoundaryStr" type="text/xml"
3785 --BoundaryStr
3786 Content-Type: text/xml
3787 <soap:Envelope>
3788     <soap:Header>
3789         <wsse:Security>
3790             <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
3791 open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
3792 1.0#Base64Binary" ValueType="http://docs.oasis-
3793 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
3794 wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">
3795         lui+Jy4WYKJW5xM3aHnLxOpGVIPzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnW
3796         SWkXm9jAEdsm/
3797         hs+f3NwwK23bh46mNmCQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yT
3798         cI7XU7xZT54S9
3799         hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrsFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI
3800         9WzxPCfHdalN4
3801         rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUc9QY3VjwNALgGDaEAT7gpURkCI8
3802         5HjdnSA5SM4cY
3803         7jAsYX/CiPEkrJcBULL1TEFrBZIBYDPzRWLSdsJRJngF7yCoGWJ+/HYOyP8P4OM5
3804         9FDi0kM8GwOE0
3805         WgYrJHH92qaVhoiPTLi7
3806     </wsse:BinarySecurityToken>
3807     <ds:Signature>
3808     <!-- The Message Signature -->

```

```

3809     <ds:SignedInfo>
3810         <ds:CanonicalizationMethod
3811 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
3812             <cl4n:InclusiveNamespaces PrefixList="wsse soap"
3813 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3814         </ds:CanonicalizationMethod>
3815         <ds:SignatureMethod
3816 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
3817         <ds:Reference URI="#TheBody">
3818             <ds:Transforms>
3819                 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
3820 exc-c14n#">
3821                     <cl4n:InclusiveNamespaces PrefixList=""
3822 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3823                 </ds:Transform>
3824             </ds:Transforms>
3825             <ds:DigestMethod
3826 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
3827             <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValu
3828 e>
3829         </ds:Reference>
3830     </ds:SignedInfo>
3831
3832     <!--A reference to a RepositoryItem (one for each
3833 RepositoryItem) -->
3834     <ds:SignedInfo>
3835         <ds:CanonicalizationMethod
3836 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
3837             <cl4n:InclusiveNamespaces PrefixList="wsse soap"
3838 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3839         </ds:CanonicalizationMethod>
3840         <ds:SignatureMethod
3841 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
3842         <ds:Reference URI="cid:${REPOSITORY_ITEM1_ID}">
3843             <ds:Transforms>
3844                 <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-
3845 exc-c14n#">
3846                     <ds:Transform Algorithm="http://docs.oasis-
3847 open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-
3848 Content-Only-Transform" />
3849                 </ds:Transform>
3850             </ds:Transforms>
3851             <ds:DigestMethod
3852 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
3853             <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValu
3854 e>
3855         </ds:Reference>
3856     </ds:SignedInfo>
3857
3858     <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureVa
3859 lue>
3860
3861     <ds:KeyInfo>
3862         <wsse:SecurityTokenReference>
3863             <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
3864 regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
3865 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
3866         </wsse:SecurityTokenReference>
3867     </ds:KeyInfo>
3868
3869 </ds:Signature>
3870 </wsse:Security>
3871 </soap:Header>

```

```

3872     <soap:Body wsu:Id="TheBody">
3873         <lcm:SubmitObjectsRequest/>
3874     </soap:Body>
3875 </soap:Envelope>
3876 --BoundaryStr
3877 Content-Type: image/png
3878 Content-ID: <${REPOSITORY_ITEM1_ID}>
3879 Content-Transfer-Encoding: base64
3880 the repository item (e.g. PNG Image) goes here..

```

3881 Listing 4: RepositoryItem Signature Example

3882 10.3.2.7 SOAP Message Security and HTTP/S

3883 When using HTTP/S between a Registry Client and a registry, SOAP message security MUST NOT be
3884 used. Specifically:

- 3885 • The Registry Client MUST NOT sign the request message or any repository items in the request.
- 3886 • The registry MUST NOT verify request or RepositoryItem signatures.
- 3887 • The registry MUST NOT sign the response message or any repository items in the response.
- 3888 • The Registry Client MUST NOT verify response or RepositoryItem signatures.

3889 10.3.3 Message Confidentiality

3890 A registry SHOULD support encryption of protocol messages as defined section 9 of [WSI-BSP] as a
3891 mechanism to support confidentiality of protocol messages during transmission on the wire.

3892 A Registry Client MAY use encryption of RepositoryItems as defined by [WSS-SWA] as a mechanism
3893 to support confidentiality of RepositoryItems during transmission on the wire.

3894 A registry SHOULD support the submission of encrypted repository items.

3895 10.3.4 Key Distribution Requirements

3896 The registry and Registered Users MUST mutually exchange their public keys. This is necessary to
3897 enable:

- 3898 • Mutual Authentication of Registry Client and registry using SSL/TLS handshake for transport
3899 layer security over HTTP/S
- 3900 • Validation of Message Signature and RepositoryItem Signature (described in section).
- 3901 • Decryption of encrypted messages

3902 In order to enable Message Security the following requirements MUST be met:

- 3903 1. A Certificate is associated with the registry.
- 3904 2. A Certificate is associated with Registry Client.
- 3905 3. A Registry Client registers its public key certificate with the registry. This is typically done during
3906 User Registration and is implementation specific.
- 3907 4. Registry Client obtains the registry's public key certificate and stores it in its own local key store.
3908 This is done in an implementation specific manner.

3909

3910 10.4 Authentication

3911 The Registry MUST be able to authenticate the identity of the User associated with client requests in
3912 order to perform authorization and access control and to maintain an Audit Trail of registry access. In
3913 security terms a service that provides the ability to authenticate requestors is referred to as an
3914 Authentication Authority.

3915 A registry MUST provide one or more of the following Authentication mechanisms:

- 3916 • Registry as Authentication Authority
- 3917 • External Authentication Authority

3918

3919 **10.4.1 Registry as Authentication Authority**

3920 A registry MAY provide authentication capability by serving as an Authentication Authority. In this role
3921 the registry uses the <ds:KeyInfo> in the Message Signature as credentials to authenticate the
3922 requestor. This typically requires checking that the public key supplied in the <ds:KeyInfo> of the
3923 Message Signature matches the public key of a Registered User. This also requires that the registry
3924 maintain a “registry keystore” that contains the public keys of Registered Users. The remaining details
3925 of registry as an authentication authority are implementation specific.

3926 Alternatively, if the Registry Client communicates with the registry over HTTP/S, the registry MUST
3927 authenticate the Registry Client User if a registered certificate is provided through SSL Client
3928 Authentication. If the certificate is not known to the registry then the Registry MUST assign the
3929 RegistryGuest principal with the Registry Client.

3930 **10.4.2 External Authentication Authority**

3931 A registry MAY also use an external Authentication Authority to authenticate client requests. The use of
3932 an external Authentication Authority requires that the registry implement the Registry SAML Profile as
3933 described in chapter 11.

3934 **10.4.3 Authenticated Session Support**

3935 Once a request is authenticated a Registry SHOULD establish an authenticated session using
3936 implementation specific means to avoid having to re-authenticate subsequent request from the same
3937 requestor. When the underlying transport protocol is HTTP, a registry SHOULD implement
3938 authenticated session support based upon HTTP session capability as defined by [RFC2965].

3939 **10.5 Authorization and Access Control**

3940 Once a registry has authenticated the identity of the Registered User associated with a client request it
3941 MUST perform authorization and subsequently enforce access control rules based upon the
3942 authorization decision.

3943 Authorization and access control is an operation conducted by the registry that decides WHO can do
3944 WHAT ACTION on WHICH RESOURCE.

- 3945 • The WHO is the User determined by the authentication step.
- 3946 • The WHAT ACTION is determined by the registry protocol request sent by the client.
- 3947 • The WHICH RESOURCE consists of the RegistryObjects and RepositoryItems impacted by the
3948 registry protocol request.

3949 The Access Control Policy associated with the resource that is impacted by the action determines
3950 authorization and access control.

3951 A registry MUST provide an access control and authorization mechanism based upon chapter titled
3952 “Access Control Information Model” in [eBRIM]. This model defines a default access control policy that
3953 MUST be supported by the registry. In addition it also defines a binding to [XACML] that allows fine-
3954 grained access control policies to be defined.

3955 **10.6 Audit Trail**

3956 Once a registry has performed authorization checks, enforced access control and allowed a client
3957 request to proceed it services the client request. A registry MUST create an Audit Trail of all

3958 LifeCycleManager operations. A registry MAY create an Audit Trail of QueryManager operations. To
3959 conserve storage resources, a registry MAY prune the Audit Trail information it stores in an
3960 implementation specific manner. A registry SHOULD perform such pruning by removing the older
3961 information in its Audit Trail content. However, it MUST not remove the original Create Event at the
3962 beginning of the audit trail since the Create Event establishes the owner of the RegistryObject.
3963 Details of how a registry maintains an Audit Trail of client requests is described in the chapter title
3964 "Event Information Model" of [ebRIM].

3965

11 Registry SAML Profile

3966
3967
3968

This chapter defines the Registry SAML Profile that a registry MAY implement in order to support SAML 2.0 protocols defined by [SAMLCore]. A specific focus of the Registry SAML Profile is the Web Single Sign On (SSO) profile defined by [SAMLProf].

3969

11.1 Terminology

3970
3971
3972

The reader should refer to the SAML Glossary [SAMLGloss] for various terms used in the Registry SAML profile. A few terms are described here for convenience:

Term	Definition
Authentication Authority	An Authentication Authority is a system entity (typically a service) that enables other system entities (typically a user or service) to establish an authenticated session by proving their identity by providing necessary credentials (e.g. username / password, certificate alias / password). An Authentication Authority produces authentication assertions as a result of successful authentication.
Enhanced Client Proxy (ECP)	Describes a client that operates under certain constraints such as not being able to support HTTP Redirect protocol. Typically these are clients that do not have a Web Browser environment. In this document the main example of an ECP is a Registry Client that uses SOAP to communicate with the registry (SOAP Requestor).
Identity Provider (IdP)	A kind of <i>service provider</i> that creates, maintains, and manages identity information for <i>principals</i> (e.g. users). An Identity Provider is usually also an Authentication Authority.
Principal	A system entity whose identity can be authenticated. This maps to User in [ebRIM].
SAML Requestor	A <i>system entity</i> that utilizes the SAML protocol to request services from another system entity (a <i>SAML authority</i> , a <i>responder</i>). The term "client" for this notion is not used because many system entities simultaneously or serially act as both clients and servers.
Service Provider (SP)	A role donned by a system entity where the system entity provides services to principals or other system entities. The Registry Service is a SP
Single Sign On (SSO)	The ability to share a single authenticated session across multiple SSO enabled services and application. The client may establish the authenticated session by authenticating with any Authentication Authority within the system. The client may then perform secure operations with any SSO enabled service within the system using the authenticated session.
Single Logout	The ability to logout nearly simultaneously from multiple Service Providers within a federated system.

3973

3974

11.2 Use Cases for SAML Profile

3975
3976

The Registry SAML Profile is intended to address following use cases using the protocols defined by [SAMLCore].

3977

11.2.1 Registry as SSO Participant:

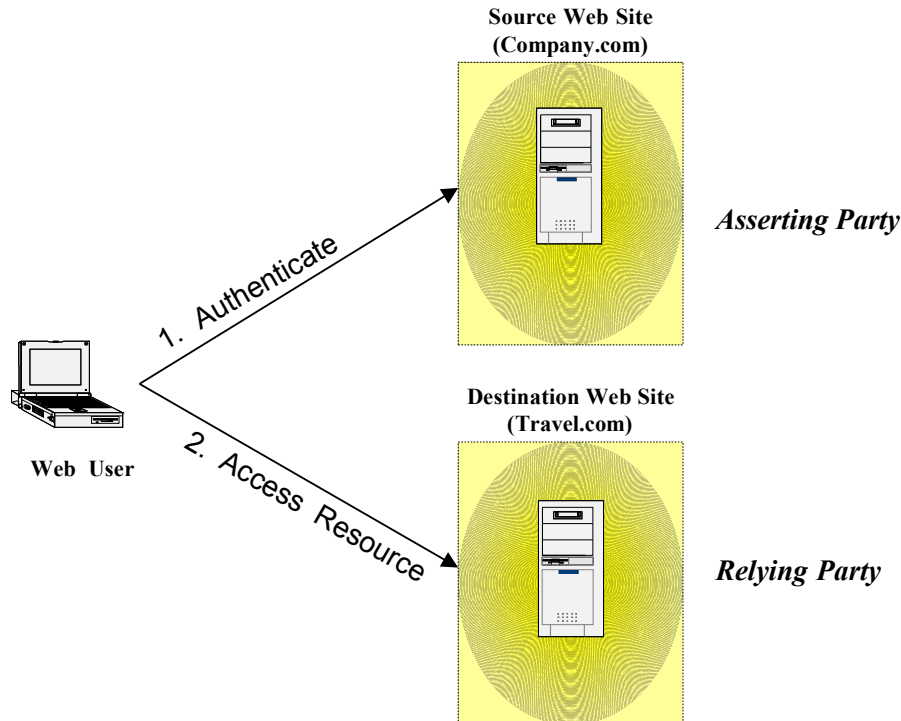
3978
3979

A large enterprise is deploying an ebXML Registry. The enterprise already has an existing Identity Provider (e.g. an Access Manager service) where it maintains user information and credentials. The

3980 enterprise also has an existing Authentication Authority (which may be the same service as the Identity
3981 Provider) that is used to authenticate users and enable Single Sign On (SSO) across all their
3982 enterprise services applications.

3983 The enterprise wishes to use its existing Identity Provider to manage registry users and to avoid
3984 duplicating the user database contained in the Identity Provider within the registry. The enterprise also
3985 wishes to use its existing Authentication Authority to authenticate registry users and expects the
3986 registry to participate in SSO capability provided by their Authentication Authority service.

3987



3988
3989

Figure 27: SAML SSO Typical Scenario

3990 11.3 SAML Roles Played By Registry

3991 In order to conform to the registry SAML Profile an ebXML Registry plays the Service Provider (SP) role
3992 based upon conformance with SAML 2.0 protocols.

3993 11.3.1 Service Provider Role

3994 The Service Provider role enables the registry to participate in SAML protocols. Specifically it allows
3995 the registry to utilize an Identity Provider to perform client authentication on its behalf.

3996 11.3.1.1 Service Provider Requirements

3997 The following are a list of requirements for the Service Provider role of the registry:

- 3998 • MUST support the protocols, messages and bindings that are the responsibility of the Service
3999 Provider as defined by Web SSO Profile in [SAMLProf]. Specifically it MUST be able to initiate
4000 and participate in the Authentication Request Protocol with an Identity Provider.
- 4001 • MUST be able to use a SAML Identity Provider to authenticate client requests.
- 4002 • MUST support the ability to maintain a security context for registry clients across multiple client
4003 requests.

4004

4005 **11.4 Registry SAML Interface**

4006 In order to conform to the registry SAML Profile an ebXML Registry MUST implement a new SAML
 4007 interface in addition to its service interfaces such as QueryManager and LifeCycleManager.

4008 Details of the registry's SAML interface are not described by this specification. Instead they are
 4009 described by the SAML 2.0 specifications and MUST support SAML HTTP and SOAP requests.

4010 A registry uses its SAML interface to participate in SAML protocols with SAML Clients and SAML
 4011 Identity Providers. Specifically, an IdentityProvider uses the registry's SAML Service Provider interface
 4012 to deliver the Response to an Authentication Request.

4013 **11.5 Requirements for Registry SAML Profile**

4014 In order to conform to the Registry SAML Profile a registry MUST implement specific SAML protocol
 4015 that support specific SAML protocol message exchanges using specific protocol bindings.

4016 Table 7 lists the matrix of SAML Profiles, Protocols Messages and their Bindings that a registry MUST
 4017 support in order to conform to the registry SAML Profile.

4018 The reader should refer to:

- 4019 • [SAMLProf] for description of profiles listed
- 4020 • [SAMLCore] for description of Message Flows listed
- 4021 • [SAMLBind] for description of Bindings listed

4022

Profile	Message Flows	Binding	Implementation Requirement
Web SSO	<AuthnRequest> from Registry to IdentityProvider	HTTP redirect	MUST
	IdentityProvider <Response> to Registry	HTTP POST	MUST
		HTTP artifact	MUST
Single Logout	<LogoutRequest>	HTTP redirect	MUST
		SOAP	MAY
	<LogoutResponse>	HTTP redirect	MUST
		SOAP	MAY
Artifact Resolution	<ArtifactResolve>,	SOAP	MUST
	<ArtifactResponse>	SOAP	MUST
Enhanced Client/Proxy SSO	ECP to Registry, Registry to ECP to IdentityProvider	PAOS	MUST
	IdentityProvider to ECP to Registry, Registry to ECP	PAOS	MUST

4023

4024

Table 7: Required SAML Profiles, Protocols and Bindings

4025 **11.6 SSO Operation**

4026 This section describes the interaction sequence for various types of SSO operations.

4027 **11.6.1 Scenario Actors**

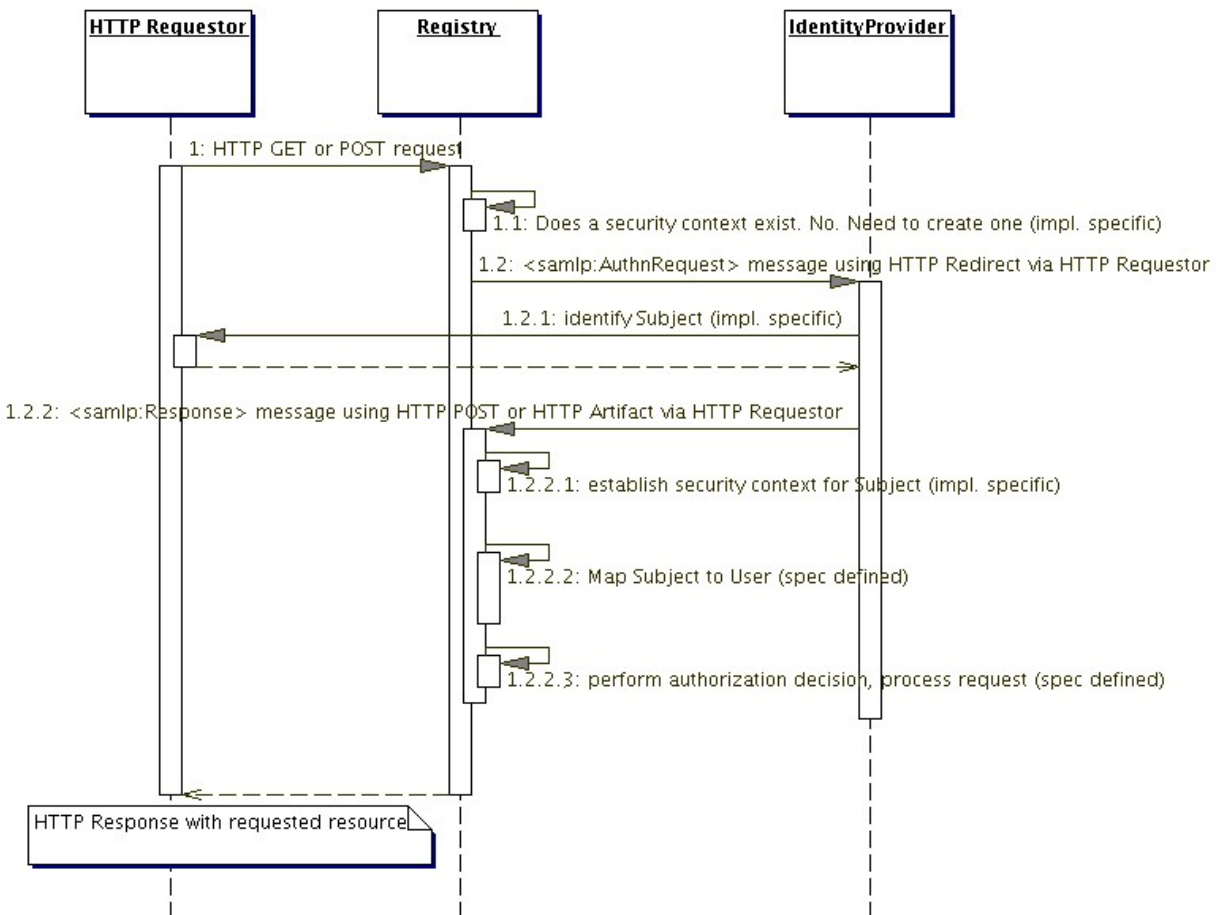
4028 The following are the actors that will be participating the various SSO Operation scenarios described in

4029 subsequent section:

- 4030 • HTTP Requestor: This represents a Registry Client that accesses the registry using the HTTP
4031 binding of the registry protocols typically through a User Agent such as a Web Browser.
- 4032 • SOAP Requestor: This represents a Registry Client that accesses the registry using the SOAP
4033 binding of the registry protocols.
- 4034 • Registry: This represents a Registry and includes all Registry interfaces such as
4035 QueryManager, LifeCycleManager and the registry's SAML Service Provider. The Registry
4036 participates in ebXML Registry protocols as well as SAML protocols.
- 4037 • IdentityProvider: This represents the IdentityProvider used by the registry to perform
4038 Authentication on its behalf.

4039 11.6.2 SSO Operation – Unauthenticated HTTP Requestor

4040 Figure 28 shows a high level view of the Single Sign On (SSO) operation when the SOAP Requestor is
4041 unauthenticated and accesses the registry over HTTP via a User Agent such as a Web Browser.



4042
4043

Figure 28: SSO Operation – Unauthenticated HTTP Requestor

4044 11.6.2.1 Scenario Sequence

4045 Figure 28 shows the following sequence of steps for the operation:

- 4046 1 The HTTP Requestor sends a HTTP GET or POST request to a Registry interface such as the
4047 QueryManager or LifeCycleManager.
- 4048 1.1 The Registry checks to see if it already has a security context established for the Subject
4049 associated with the request. It determines that there is no pre-existing security context.
- 4050 1.2 In order to establish a security context, the Registry therefor initiates the <saml:AuthnRequest>
4051 protocol with the IdentityProvider. The <AuthnRequest> is sent using HTTP Redirect via the User
4052 Agent (e.g. Web Browser) used by the HTTP Requestor.
- 4053 1.2.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this
4054 requires communicating with the User Agent being used by the HTTP Requestor to get the
4055 credentials associated with the Subject and then using the credentials to authenticate that the
4056 IdentityProvider knows the Subject. In case of SSL/TLS based communication the credetials
4057 are acquired without any user intervention directly from the User Agent. The figure assumes
4058 that the IdentityProvider is able to authenticate the Subject.
- 4059 1.2.2 The IdentityProvider sends a <saml:Response> message containing a
4060 <saml:AuthenticationStatement> to the Registry using either HTTP POST or HTTP Artifact
4061 SAML Binding via the User Agent.
- 4062 1.2.2.1 The Registry uses implementation specific means to establish a security context for the
4063 Subject authenticated by the IdentityProvider based upon the information contained about the
4064 Subject in the <saml:Response> message. This may include creating an HTTP Session for
4065 the HTTP Requestor.
- 4066 1.2.2.2 The Registry maps the information about the Subject in the <saml:Response> message into
4067 a <rim:User> instance. This establishes the <rim:User>context for the security context.
- 4068 1.2.2.3 The Registry then performs authorization decision based upon the original HTTP request and
4069 the <rim:User>. The figure assumes that authorization decision was to allow the request to be
4070 processed. The Registry processes the request and subsequently return the requested
4071 resource to the HTTP Requestor via the HTTP response.
- 4072

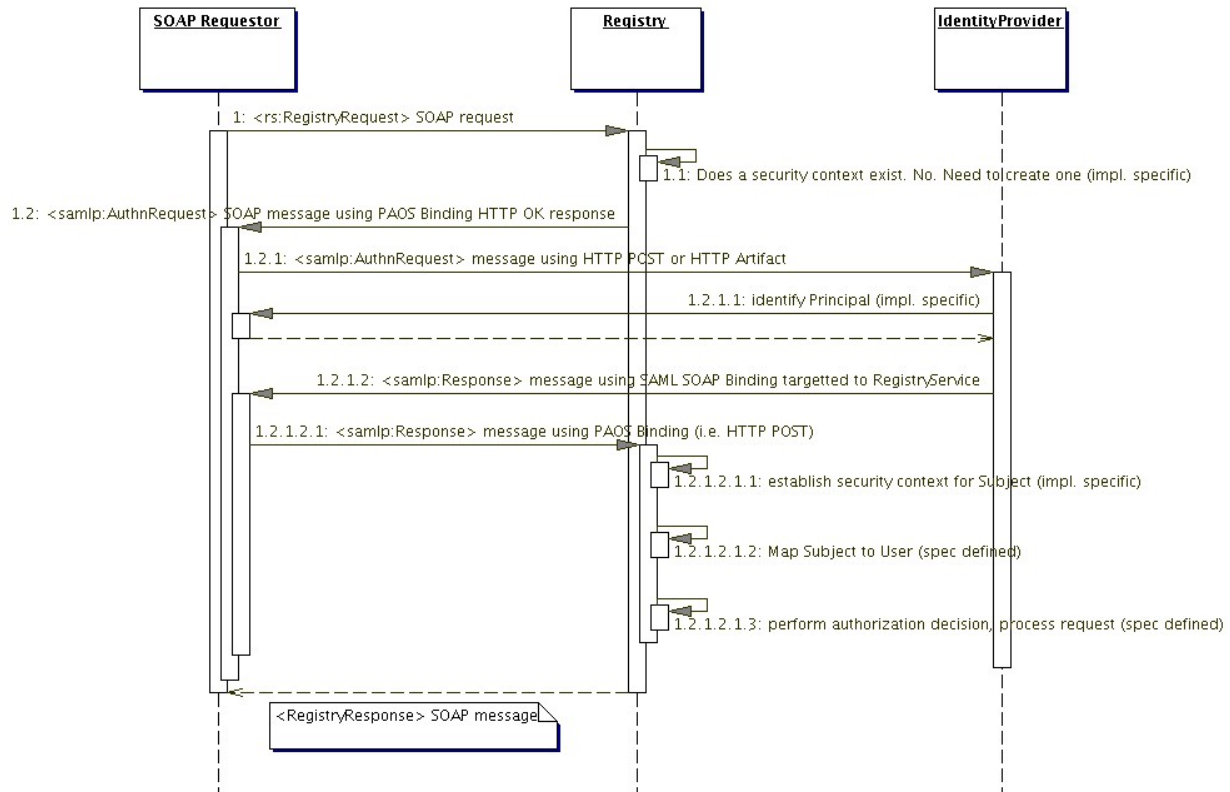
4073 **11.6.3 SSO Operation – Authenticated HTTP Requestor**

4074 This is the case where the HTTP Requestor first authenticates with an IdentityProvider and then
4075 accesses the registry over HTTP via a User Agent such as a Web Browser.

4076 Currently there are no standard means defined for carrying SAML Assertions resulting from the
4077 Registry Requestor authenticating with an IdentityProvider over HTTP protocol to a Service Provider
4078 such as the registry. A registry MAY support this scenario in an implementation specific manner.
4079 Typically, the Identity Provider will define any such implementation specific manner.

4080 **11.6.4 SSO Operation – Unauthenticated SOAP Requestor**

4081 This is the case where an unauthenticated Registry Requestor accesses the registry over SOAP.
4082 Figure 29 shows the steps involved.



4083
4084

Figure 29: SSO Operation - Unauthenticated SOAP Requestor

4085 **11.6.4.1 Scenario Sequence**

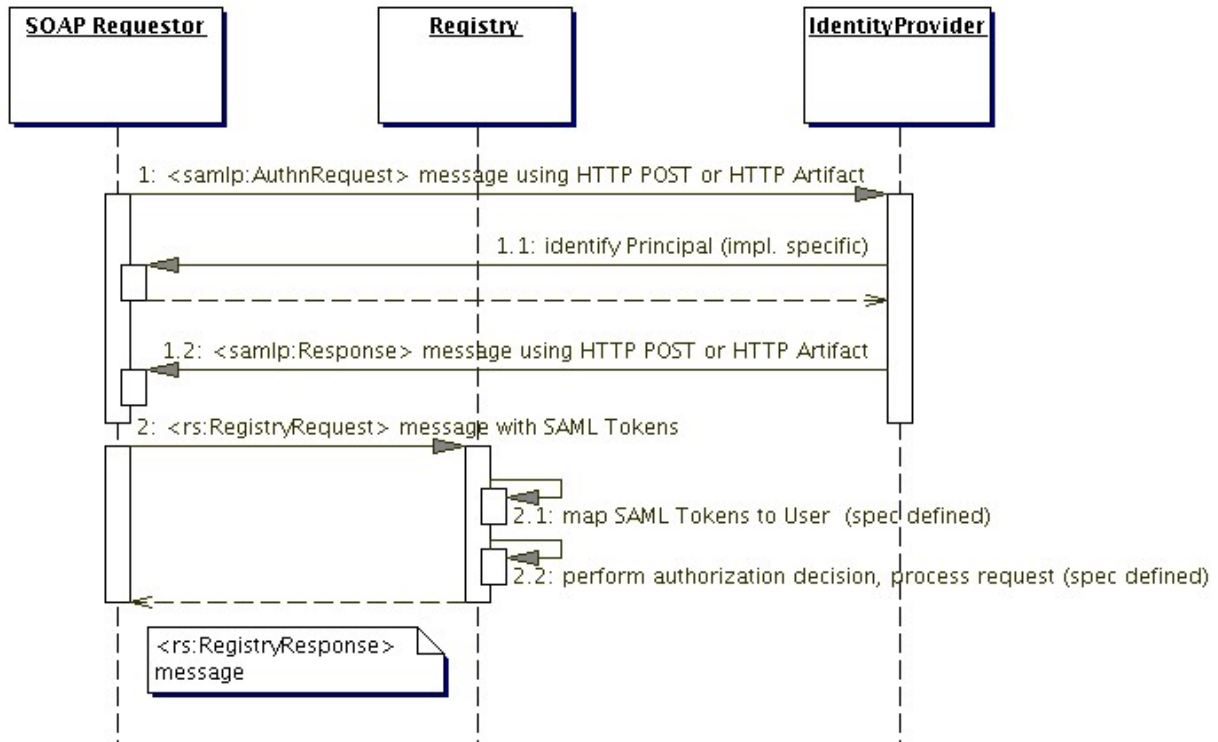
4086 Figure 29 shows the following sequence of steps for the operation:

- 4087 1 The SOAP Requestor sends a `<rs:RegistryRequest>` SOAP message such as a
4088 `<lcm:SubmitObjectsRequest>` to a Registry interface such as the `LifeCycleManagerManager`. In
4089 the request header the SOAP Requestor declares that it is an ECP requestor as defined by the
4090 ECP Profile in `[SAMLProf]`.
- 4091 1.1 The Registry checks to see if it already has a security context established for the Subject
4092 associated with the request. It determines that there is no pre-existing security context.
- 4093 1.2 Because the request is from an ECP client, the registry uses the ECP Profile defined by
4094 `[SAMLProf]` and sends a `<samlp:AuthnRequest>` SOAP message as response to the
4095 `<rs:RegistryRequest>` SOAP message to the SOAP Requestor using the PAOS Binding as
4096 defined by `[SAMLBind]`. The response has an HTTP Response status of OK.
- 4097 1.2.1 The SOAP Requestor then initiates the `<samlp:AuthnRequest>` protocol with the
4098 IdentityProvider. The `<samlp:AuthnRequest>` is sent using HTTP POST or Artifact Binding
4099 directly to the IdentityProvider.
- 4100 1.2.1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically
4101 this requires communicating with the SOAP Requestor to get the credentials associated with
4102 the Subject and then using the credentials to authenticate that the IdentityProvider knows the
4103 Subject. In case of SSL/TLS based communication the credentials are acquired without any
4104 user intervention directly from the SOAP Requestor. The figure assumes that the
4105 IdentityProvider is able to authenticate the Subject.
- 4106 1.2.1.2 The IdentityProvider sends a `<samlp:Response>` message containing a
4107 `<saml:AuthenticationStatement>` to the SOAP Requestor using SAML SOAP Binding. The

4108 HTTP header specifies the Registry as the ultimate target of the response.
4109 1.2.1.2.1 The SOAP Requestor forwards the <saml:Response> message containing a
4110 <saml:AuthenticationStatement> to the Registry using PAOS Binding via HTTP POST.
4111 1.2.1.2.1.1 The Registry uses implementation specific means to establish a security context for the
4112 Subject authenticated by the IdentityProvider based upon the information contained about
4113 the Subject in the <samlp:Response> message. This may include creating an HTTP
4114 Session for the HTTP Requestor.
4115 1.2.1.2.1.2 The Registry maps the information about the Subject in the <samlp:Response> message
4116 into a <rim:User> instance. This establishes the <rim:User> context for the security
4117 context.
4118 1.2.1.2.1.3 The Registry then performs authorization decision based upon the original SOAP request
4119 and the <rim:User>. The figure assumes that authorization decision was to allow the
4120 request to be processed. The Registry processes the request and subsequently return a
4121 <rs:RegistryResponse> SOAP message as response to the original <rs:RegistryRequest>
4122 SOAP request.
4123

4124 **11.6.5 SSO Operation – Authenticated SOAP Requestor**

4125 This is the case where the Registry Requestor first authenticates with an IdentityProvider directly and
4126 then makes a request to the registry using SOAP.



4127
4128

Figure 30: SSO Operation - Authenticated SOAP Requestor

4129 **11.6.5.1 Scenario Sequence**

4130 The figure shows the following sequence of steps for the operation:

- 4131 1 The SOAP Requestor then initiates the <samlp:AuthnRequest> protocol directly with the
4132 IdentityProvider. The <samlp:AuthnRequest> is sent using HTTP POST or Artifact Binding.
- 4133 1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this
4134 requires communicating with the SOAP Requestor to get the credentials associated with the
4135 Subject and then using the credentials to authenticate that the IdentityProvider knows the
4136 Subject. In case of SSL/TLS based communication the credentials are acquired without any user
4137 intervention directly from the SOAP Requestor. The figure assumes that the IdentityProvider is
4138 able to authenticate the Subject.
- 4139 1.2 The IdentityProvider sends a <samlp:Response> message containing a
4140 <saml:AuthenticationStatement> to the SOAP Requestor using SAML HTTP POST or HTTP
4141 Artifact Binding.
- 4142 2 The SOAP Requestor sends a <rs:RegistryRequest> SOAP message such as a
4143 <lcm:SubmitObjectsRequest> to a Registry interface such as the LifeCycleManagerManager. The

4144 <rs:RegistryRequest> SOAP message includes SAML Tokens in the <soap:Header> of the SOAP
4145 message as defined by [WSS-SAML]. The SAML Tokens are based upon the <saml:Response>
4146 during authentication.

4147 2.1 The registry maps the SAML Tokens from the <soap:Header> of the <rs:RegistryRequest> to a
4148 <rim:User> instance. This establishes the <rim:User> context for the request.

4149 2.2 The Registry then performs authorization decision based upon the original SOAP request and
4150 the <rim:User>. The figure assumes that authorization decision was to allow the request to be
4151 processed. The Registry processes the request and subsequently return a
4152 <rs:RegistryResponse> SOAP message as response to the original <rs:RegistryRequest> SOAP
4153 request.
4154

4155 **11.6.6 <saml:AuthnRequest> Generation Rules**

4156 The following rules MUST be observed when the registry or Registry Client issues a
4157 <saml:AuthnRequest>:
4158

- 4159 • A registry MUST specify a NameIDPolicy within the <saml:AuthRequest>
- 4160 • The Format of the NameIDPolicy MUST be urn:oasis:names:tc:SAML:2.0:nameid-
4161 format:persistent as defined by section in [SAMLCore]. Note that it is the Persistent Identifier
4162 that maps to the id attribute of <rim:User>.

4163 —

4164 **11.6.7 <saml:Response> Processing Rules**

4165 This section describes how the registry processes the <saml:Response> to a <saml:AuthnRequest>:

4166 **<saml:Response> Processing**

- 4167 • Response Processing: The registry MUST verify the <ds:Signature> for the <saml:Response>
4168 if present.
- 4169 • The registry MUST check the <saml:Status> associated with <saml:Response> for errors. If
4170 the <saml:Status> has a top level <saml:StatusCode> whose value is NOT
4171 urn:oasis:names:tc:SAML:2.0:status:Success then the registry MUST throw
4172 an AuthenticationException. The AuthenticationException message SHOULD include the
4173 information from the StatusCode, StatusMessage and StatusDetail from the <saml:Status>.

4174 **<saml:Assertion> Processing**

- 4175 • The registry SHOULD check the <saml:Assertion> for Conditions and honour any standard
4176 Conditions defined by [SAMLCore] if any are specified.

4177 **<saml:AuthnStatement> Processing**

- 4178 • The registry MUST check the SessionNotOnOrAfter attribute of the <saml:AuthnStatement> for
4179 validity of the authenticated session.

4180 **<saml:Subject> Processing**

- 4181 • A registry MUST map the <saml:Subject> to a <rim:User> instance as described in 11.6.8.

4182 **11.6.8 Mapping Subject to User**

4183 As required by [SAMLCore] a <saml:Response> to a <saml:AuthnRequest> MUST contain a
4184 <saml:Subject> that identifies the Subject that was authenticated by the IdentityProvider. In addition it
4185 MUST contain a <saml:AuthnStatement> which asserts that the IdentityProvider indeed authenticated
4186 the Subject.

4187 The following table defines the mapping between a <saml:Subject> and a <rim:User>:
4188

– Subject Attribute	– User Attribute	– Description
– NameID content	– id attribute	NameID Format MUST be “urn:oasis:names:tc:SAML:1.1:nameid-format:persistent”

4189 **Table 8: Mapping Subject to User**

4190 Note that any attribute of Subject not specified above SHOULD be ignored when mapping Subject to
4191 User. Note that any attribute of User not specified above MUST be left unspecified when mapping
4192 Subject to User.

4193 **11.7 External Users**

4194 The SAML Profile allows registry Users to be registered in an Identity Provider external to the registry.
4195 These are referred to as “External Users”. A registry dynamically creates such External Users by
4196 mapping a SAML Subject to a User instance dynamically.

4197 The following are some restrictions on External User instances:

- 4198 • External User instances are transient from the registry’s perspective and MUST not be stored
4199 within the registry as User instances
- 4200 • A RegistryObject MUST not have a reference to an External User unless it is composed within
4201 that RegistryObject. Composed RegistryObjects such as Classification instances are allowed to
4202 reference their parent External User instance.
- 4203 • Since External User instances are transient they MUST not match a registry Query.

4204
4205
4206
4207
4208

4209 12 Native Language Support (NLS)

4210 This chapter describes the Native Languages Support (NLS) features of ebXML Registry.

4211 12.1 Terminology

4212 The following terms are used in NLS.

NLS Term	Description
Coded Character Set (CCS)	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.
Character Encoding Scheme (CES)	CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are ISO-2022, UTF-8.
Character Set (charset)	<ul style="list-style-type: none">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.A list of registered character sets can be found at [IANA].

4213

4214 12.2 NLS and Registry Protocol Messages

4215 For the accurate processing of data in both registry client and registry services, it is essential for the
4216 recipient of a protocol message to know the character set being used by it.

4217 A Registry Client SHOULD specify charset parameter in MIME header when they specify text/xml as
4218 Content-Type. A registry MUST specify charset parameter in MIME header when they specify text/xml
4219 as Content-Type.

4220 The following is an example of specifying the character set in the MIME header.

4221

```
4222 Content-Type: text/xml; charset=ISO-2022-JP
```

4223

4224

4225 If a registry receives a protocol message with the charset parameter omitted then it MUST use the
4226 default charset value of "us-ascii" as defined in [RFC 3023].

4227 Also, when an application/xml entity is used, the charset parameter is optional, and registry client and
4228 registry services MUST follow the requirements in Section 4.3.3 of [REC-XML] which directly address
4229 this contingency.

4230 If another Content-Type is used, then usage of charset MUST follow [RFC 3023].

4231 12.3 NLS Support in RegistryObjects

4232 The information model XML Schema [RR-RIM-XSD] defines the <rim:InternationalStringType> for
4233 defining elements that contains a locale sensitive string value.

4234

```
4235 <complexType name="InternationalStringType">  
4236 <sequence maxOccurs="unbounded" minOccurs="0">
```

```
4237     <element ref="tns:LocalizedString"/>
4238   </sequence>
4239 </complexType>
```

4240

4241 An InternationalStringType may contain zero or more LocalizedStrings within it where each
4242 LocalizedString contain a string value is a specified local language and character set.

4243

```
4244 <complexType name="LocalizedStringType">
4245   <attribute ref="xml:lang" default="en-US"/>
4246   <attribute default="UTF-8" name="charset"/>
4247   <attribute name="value" type="tns:FreeFormText" use="required"/>
4248 </complexType>
```

4249

4250 Examples of such attributes are the “name” and “description” attributes of the RegistryObject class
4251 defined by [ebRIM] as shown below.

```
4252   <complexType name="InternationalStringType">
4253     <sequence maxOccurs="unbounded" minOccurs="0">
4254       <element ref="tns:LocalizedString"/>
4255     </sequence>
4256   </complexType>
4257   <element name="InternationalString"
4258 type="tns:InternationalStringType"/>
4259   <element name="Name" type="tns:InternationalStringType"/>
4260   <element name="Description" type="tns:InternationalStringType"/>
4261
4262   <complexType name="LocalizedStringType">
4263     <attribute ref="xml:lang" default="en-US"/>
4264     <!--attribute name = "lang" default = "en-US" form = "qualified"
4265 type = "language"/-->
4266     <attribute default="UTF-8" name="charset"/>
4267     <attribute name="value" type="tns:FreeFormText" use="required"/>
4268   </complexType>
4269   <element name="LocalizedString" type="tns:LocalizedStringType"/>
```

4270

4271 An element InternationalString is capable of supporting multiple locales within its collection of
4272 LocalizedStrings.

4273 The above schema allows a single RegistryObject instance to include values for any NLS sensitive
4274 element in multiple locales.

4275 The following example illustrates how a single RegistryObject can contain NLS sensitive <rim:Name>
4276 and “<rim:Description>” elements with their value specified in multiple locales. Note that the
4277 <rim:Name> and <rim:Description> use the <rim:InternationalStringType> as their type.

```
4278   <rim:ExtrinsicObject id="{ID}" mimeType="text/xml">
4279     <rim:Name>
4280       <rim:LocalizedString xml:lang="en-US" value="customACP1.xml"/>
4281       <rim:LocalizedString xml:lang="fi-FI" value="customACP1.xml"/>
4282       <rim:LocalizedString xml:lang="pt-BR" value="customACP1.xml"/>
4283     </rim:Name>
4284     <rim:Description>
4285       <rim:LocalizedString xml:lang="en-US" value="A sample custom
4286 ACP"/>
4287       <rim:LocalizedString xml:lang="fi-FI" value="Esimerkki custom
4288 ACP"/>
4289       <rim:LocalizedString xml:lang="pt-BR" value="Exemplo de ACP
4290 customizado
4291 "/>
4292     </rim:Description>
```

4293 `</rim:ExtrinsicObject>`

4294

4295 Since locale information is specified at the sub-element level there is no language or character set
4296 associated with a specific RegistryObject instance.

4297 **12.3.1 Character Set of *LocalizedString***

4298 The character set used by a locale specific String (*LocalizedString*) is defined by the charset attribute.
4299 Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of
4300 *LocalizedStrings* for maximum interoperability.

4301 **12.3.2 Language of *LocalizedString***

4302 The language MAY be specified in `xml:lang` attribute (Section 2.12 [REC-XML]).

4303 **12.4 NLS and Repository Items**

4304 While a single instance of an *ExtrinsicObject* is capable of supporting multiple locales, it is always
4305 associated with a single repository item. The repository item MAY be in a single locale or MAY be in
4306 multiple locales. This specification does not specify any NLS requirements for repository items.

4307 **12.4.1 Character Set of Repository Items**

4308 When a submitter submits a repository item, they MAY specify the character set used by the
4309 repository item using the MIME *Content-Type* mime header for the mime multipart containing the
4310 repository item as shown below:

4311 `Content-Type: text/xml; charset="UTF-8"`

4312

4313

4314
4315 Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of
4316 *LocalizedStrings* for maximum interoperability. A registry MUST preserve the charset of a repository
4317 item as it is originally specified when it is submitted to the registry.

4318 **12.4.2 Language of Repository Items**

4319 The *Content-language* mime header for the mime bodypart containing the repository item MAY specify
4320 the language for a locale specific repository item. The value of the *Content-language* mime header
4321 property MUST conform to [RFC 1766].

4322 This document currently specifies only the method of sending the information of character set and
4323 language, and how it is stored in a registry. However, the language information MAY be used as one of
4324 the query criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation
4325 procedure, like registry client is asking a favorite language for messages from registry services, could
4326 be another functionality for the future revision of this document.

4327 **13 Conformance**

4328 This chapter defines the technical conformance requirements for ebXML Registry. Note that it does not
4329 define specific conformance tests to verify compliance with various conformance profiles.

4330 **13.1 Conformance Profiles**

4331 An ebXML Registry **MUST** comply with one of the following conformance profiles:

- 4332 • Registry Lite – This conformance profile requires the registry to implement a minimal set of core
4333 features defined by this specification.
- 4334 • Registry Full – This conformance profile requires the registry to implement additional set of
4335 features in addition to those required by the Registry Lite conformance profile.

4336 **13.2 Feature Matrix**

4337 The following table identifies the implementation requirements for each feature defined by this
4338 specification for each conformance profile defined above.

Table 9: Feature Conformance Matrix

Feature	Registry Lite	Registry Full
SOAP Binding		
QueryManager binding	MUST	MUST
LifeCycleManager binding	MUST	MUST
HTTP Binding		
RPC Encoded URL	MUST	MUST
User Defined URL	MAY	MUST
File Path URL	MAY	MUST
LifeCycleManager		
SubmitObjects Protocol	MUST	MUST
UpdateObjects Protocol	MUST	MUST
ApproveObjects Protocol	MUST	MUST
DeprecateObjects Protocol	MUST	MUST
UnderprecateObjects Protocol	MUST	MUST
RemoveObjects Protocol	MUST	MUST
Registry Managed Version Control	MAY	MUST
QueryManager		
SQL Query	MAY	MUST
Filter Query	MUST	MUST
Stored Parameterized Query	MAY	MUST
Iterative Query	MAY	MUST
Event Notification	MAY	MUST
Content Management Services		
Validate Content Protocol	MAY	MUST
Catalog Content Protocol	MAY	MUST
Canonical XML Cataloging Service	MAY	MUST
Cooperating Registries		
Remote object references	MAY	MUST
Federated queries	MAY	MUST
Object Replication	MAY	MUST
Object Relocation	MAY	MUST
Registry Security		
Identity Management	MUST	MUST
Message Security		
Transport layer security	MAY	MUST
SOAP Message Security	MUST	MUST
Repository Item Security	MUST	MUST
Authorization and Access Control		
Default Access Control Policy	MUST	MUST
Custom Access Control Policies	MAY	MUST
Audit Trail	MUST	MUST

Feature	Registry Lite	Registry Full
Registry SAML Profile	MAY	MUST
NLS	MUST	MUST

4339

14 References

4340

14.1 Normative References

4341

- 4342 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF
4343 RFC 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 4344 **[ebRIM]** ebXML Registry Information Model Version 3.0
4345 [http://www.oasis-open.org/committees/regrep/documents/3.0/specs/regrep-rim-](http://www.oasis-open.org/committees/regrep/documents/3.0/specs/regrep-rim-3.0-cs-01.pdf)
4346 [3.0-cs-01.pdf](http://www.oasis-open.org/committees/regrep/documents/3.0/specs/regrep-rim-3.0-cs-01.pdf)
- 4347 **[REC-XML]** W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
4348 <http://www.w3.org/TR/REC-xml>
- 4349 **[RFC 1766]** IETF (Internet Engineering Task Force). RFC 1766:
4350 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
4351 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 4352 **[RFC 2130]** IETF (Internet Engineering Task Force). RFC 2130
4353 The Report of the IAB Character Set Workshop held 29 February - 1 March,
4354 1996
4355 <http://www.faqs.org/rfcs/rfc2130.html>
- 4356 **[RFC 2277]** IETF (Internet Engineering Task Force). RFC 2277:
4357 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
4358 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 4359 **[RFC 2278]** IETF (Internet Engineering Task Force). RFC 2278:
4360 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
4361 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 4362 **[RFC2616]** IETF (Internet Engineering Task Force). RFC 2616:
4363 Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1* . 1999.
4364 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 4365 **[RFC2965]** IETF (Internet Engineering Task Force). RFC 2965:
4366 D. Kristol et al. *HTTP State Management Mechanism*. 2000.
4367 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 4368 **[RR-CMS-XSD]** ebXML Registry Content Management Services XML Schema
4369 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>
- 4370 **[RR-LCM-XSD]** ebXML Registry LifeCycleManager XML Schema
4371 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/lcm.xsd>
- 4372 **[RR-RIM-XSD]** ebXML Registry Information Model XML Schema
4373 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>
- 4374 **[RR-RS-XSD]** ebXML Registry Service Protocol XML Schema
4375 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rs.xsd>
- 4376 **[RR-QM-XSD]** ebXML Registry QueryManager XML Schema
4377 [http://www.oasis-](http://www.oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd)
4378 [open.org/committees/regrep/documents/3.0/schema/query.xsd](http://www.oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd)
- 4379 **[SAMLBind]** S. Cantor et al., *Bindings for the OASIS Security Assertion Markup Language*
4380 (*SAML*) *V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-
4381 bindings-2.0-cd-03.
4382 <http://www.oasis-open.org/committees/security/>.
- 4383 Note: when this document is finalized, this URL will be updated.
- 4384 **[SAMLConform]** P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion*
4385 *Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID
4386 sstc-saml-conformance-2.0-cd-03.
4387 <http://www.oasis-open.org/committees/security/>.

4388		Note: when this document is finalized, this URL will be updated.
4389	[SAMLCore]	S. Cantor et al., <i>Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, December 2004. Document ID sstc-saml-core-2.0-cd-03.
4390		
4391		http://www.oasis-open.org/committees/security/ .
4392		
4393		Note: when this document is finalized, this URL will be updated.
4394	[SAMLProf]	S. Cantor et al., <i>Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, September 2004. Document ID sstc-saml-profiles-2.0-cd-03.
4395		
4396		http://www.oasis-open.org/committees/security/ .
4397		
4398		Note: when this document is finalized, this URL will be updated.
4399	[SAMLXSD]	S. Cantor et al., SAML protocols schema. OASIS SSTC, September 2004.
4400		Document ID sstc-saml-schema-protocol-2.0.
4401		http://www.oasis-open.org/committees/security/ .
4402		Note: when this document is finalized, this URL will be updated.
4403	[SAMLXSD]	S. Cantor et al., SAML assertions schema. OASIS SSTC, September 2004.
4404		Document ID sstc-saml-schema-assertion-2.0.
4405		http://www.oasis-open.org/committees/security/ .
4406		Note: when this document is finalized, this URL will be updated.
4407	[SOAP11]	W3C Note. Simple Object Access Protocol, May 2000
4408		http://www.w3.org/TR/SOAP
4409	[SwA]	W3C Note: SOAP with Attachments, Dec 2000
4410		http://www.w3.org/TR/SOAP-attachments
4411	[SQL]	Structured Query Language (FIPS PUB 127-2)
4412		http://www.itl.nist.gov/fipspubs/fip127-2.htm
4413	[SQL/PSM]	Database Language SQL — Part 4: Persistent Stored Modules
4414		(SQL/PSM) [ISO/IEC 9075-4:1996]
4415	[UUID]	DCE 128 bit Universal Unique Identifier
4416		http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
4417	[WSDL]	W3C Note. Web Services Description Language (WSDL) 1.1
4418		http://www.w3.org/TR/wsdl
4419	[XML]	T. Bray, et al. Extensible Markup Language (XML) 1.0 (Second Edition). World
4420		Wide Web Consortium, October 2000.
4421		http://www.w3.org/TR/REC-xml
4422	[XMLDSIG]	XML-Signature Syntax and Processing
4423		http://www.w3.org/TR/2001/PR-xmldsig-core-20010820/
4424	[WSI-BSP]	WS-I: Basic Security Profile 1.0
4425		http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html
4426		Note: when this document is finalized, this URL will be updated.
4427	[WSS-SMS]	Web Services Security: SOAP Message Security 1.0
4428		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
4429		
4430	[WSS-SWA]	Web Services Security: SOAP Message with Attachments (SwA) Profile 1.0
4431		http://www.oasis-open.org/apps/org/workgroup/wss/download.php/10902/wss-swa-profile-1.0-cd-01.pdf
4432		
4433		Note: when this document is finalized, this URL will be updated.

4434 14.2 Informative

4435	[ebBPSS]	ebXML Business Process Specification Schema
4436		http://www.ebxml.org/specs

4437	[ebCPP]	ebXML Collaboration-Protocol Profile and Agreement Specification http://www.ebxml.org/specs/
4438		
4439	[ebMS]	ebXML Messaging Service Specification, Version 1.0 http://www.ebxml.org/specs/
4440		
4441	[DeltaV]	Versioning Extension to WebDAV, IETF RFC 3253 http://www.webdav.org/deltav/protocol/rfc3253.html
4442		
4443	[XPT]	XML Path Language (XPath) Version 1.0 http://www.w3.org/TR/xpath
4444		
4445	[IANA]	IANA (Internet Assigned Numbers Authority). Official Names for Character Sets, ed. Keld Simonsen et al. http://www.iana.org/
4446		
4447		
4448	[RFC2392]	E. Levinson, Content-ID and Message-ID Uniform Resource Locators, IETF RFC 2392, http://www.ietf.org/rfc/rfc2392.txt
4449		
4450		
4451	[RFC 2828]	IETF (Internet Engineering Task Force). RFC 2828: Internet Security Glossary, ed. R. Shirey. May 2000. http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html
4452		
4453		
4454	[RFC 3023]	IETF (Internet Engineering Task Force). RFC 3023: XML Media Types, ed. M. Murata. 2001. ftp://ftp.isi.edu/in-notes/rfc3023.txt
4455		
4456		
4457	[SAMLMeta]	S. Cantor et al., <i>Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, September 2004. Document ID sstc-saml-metadata-2.0-cd-02. http://www.oasis-open.org/committees/security/ .
4458		
4459		
4460		
4461	[SAMLGloss]	J. Hodges et al., <i>Glossary for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, September 2004. Document ID sstc-saml-glossary-2.0-cd-02. http://www.oasis-open.org/committees/security/ .
4462		
4463		
4464		
4465	[SAMLSecure]	F. Hirsch et al., <i>Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0</i> . OASIS SSTC, September 2004. Document ID sstc-saml-sec-consider-2.0-cd-02. http://www.oasis-open.org/committees/security/ .
4466		
4467		
4468		
4469	[SAMLTech]	J.Hughes et al., Technical Overview of the OASIS Security Assertion Markup Language (SAML)V2.0. http://www.oasis-open.org/committees/download.php/7874/sstc-saml-tech-overview-2.0-draft-01.pdf
4470		
4471		
4472		
4473	[UML]	Unified Modeling Language http://www.uml.org http://www.omg.org/cgi-bin/doc?formal/03-03-01
4474		
4475		
4476		

4477

A. Acknowledgments

4478

The editors would like to acknowledge the contributions of the OASIS ebXML Registry Technical Committee, whose voting members at the time of publication are listed as contributors on the title page of this document.

4479

4480

4481

- Finally, the editors wish to acknowledge the following people for their contributions of material used as input to the OASIS ebXML Registry specifications:

4482

4483

Name	Affiliation
Aziz Abouelfoutouh	Government of Canada
Ed Buchinski	Government of Canada
Asuman Dogac	Middle East Technical University, Ankara Turkey
Michael Kass	NIST
Richard Lessard	Government of Canada
Evan Wallace	NIST
David Webber	Individual

4484

•

4485

B. Notices

4486 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that
4487 might be claimed to pertain to the implementation or use of the technology described in this document
4488 or the extent to which any license under such rights might or might not be available; neither does it
4489 represent that it has made any effort to identify any such rights. Information on OASIS's procedures
4490 with respect to rights in OASIS specifications can be found at the OASIS website. Copies of claims of
4491 rights made available for publication and any assurances of licenses to be made available, or the result
4492 of an attempt made to obtain a general license or permission for the use of such proprietary rights by
4493 implementors or users of this specification, can be obtained from the OASIS Executive Director.

4494 OASIS invites any interested party to bring to its attention any copyrights, patents or patent
4495 applications, or other proprietary rights which may cover technology that may be required to implement
4496 this specification. Please address the information to the OASIS Executive Director.

4497 **Copyright © OASIS Open 2004. All Rights Reserved.**

4498 This document and translations of it may be copied and furnished to others, and derivative works that
4499 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published
4500 and distributed, in whole or in part, without restriction of any kind, provided that the above copyright
4501 notice and this paragraph are included on all such copies and derivative works. However, this
4502 document itself does not be modified in any way, such as by removing the copyright notice or
4503 references to OASIS, except as needed for the purpose of developing OASIS specifications, in which
4504 case the procedures for copyrights defined in the OASIS Intellectual Property Rights document must be
4505 followed, or as required to translate it into languages other than English.

4506 The limited permissions granted above are perpetual and will not be revoked by OASIS or its
4507 successors or assigns.

4508 This document and the information contained herein is provided on an "AS IS" basis and OASIS
4509 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
4510 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS
4511 OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR
4512 PURPOSE.