

ebXML Registry Services and Protocols Version 3.0.1

Committee Draft, Aug 23, 2007

Document identifier:

regrep-rs

Location:

Latest Version: <http://docs.oasis-open.org/regrep-rs/latest/>

This Version: <http://docs.oasis-open.org/regrep-rs/v3.0.1/>

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

Editors:

Name	Affiliation
Kathryn Breininger	The Boeing Company
Farrukh Najmi	Wellfleet Software Corporation
Nikola Stojanovic	GS1 US

Contributors:

Name	Affiliation
Ivan Bedini	France Telecom
Ted Haas	GS1 US
Paul Macias	LMI
Carl Mattocks	MetLife
Monica Martin	Sun Microsystems
David Webber	Individual

Abstract:

This document defines the services and protocols for an ebXML Registry

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

Status:

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

Committee members should send comments on this specification to the regrep@lists.oasis-open.org list. Others should subscribe to and send comments to the regrep-comment@lists.oasis-open.org list. To subscribe, send an email message to regrep-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

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

Table of Contents

30	1 Introduction.....	12
31	1.1 Audience.....	12
32	1.2 Terminology.....	12
33	1.3 Notational Conventions.....	12
34	1.3.1 UML Diagrams.....	12
35	1.3.2 Identifier Placeholders.....	12
36	1.3.3 Constants.....	12
37	1.3.4 Bold Text.....	13
38	1.3.5 Example Values.....	13
39	1.4 XML Schema Conventions.....	13
40	1.4.1 Schemas Defined by ebXML Registry.....	13
41	1.4.2 Schemas Used By ebXML Registry.....	14
42	1.5 Registry Actors.....	15
43	1.6 Registry Use Cases.....	15
44	1.7 Registry Architecture.....	15
45	1.7.1 Registry Clients.....	16
46	1.7.1.1 Client API.....	16
47	1.7.2 Registry Service Interfaces.....	16
48	1.7.3 Service Interface: Protocol Bindings.....	16
49	1.7.4 Authentication and Authorization.....	17
50	1.7.5 Metadata Registry and Content Repository.....	17
51	2 Registry Protocols.....	18
52	2.1 Requests and Responses.....	18
53	2.1.1 RegistryRequestType.....	18
54	2.1.1.1 Syntax:.....	18
55	2.1.1.2 Parameters:.....	19
56	2.1.1.3 Returns:.....	19
57	2.1.1.4 Exceptions:.....	19
58	2.1.2 RegistryRequest.....	19
59	2.1.3 RegistryResponseType.....	19
60	2.1.3.1 Syntax:.....	20
61	2.1.3.2 Parameters:.....	20
62	2.1.4 RegistryResponse.....	20
63	2.1.5 RegistryErrorList.....	20
64	2.1.5.1 Syntax:.....	21
65	2.1.5.2 Parameters:.....	21
66	2.1.6 RegistryError.....	21
67	2.1.6.1 Syntax:.....	21
68	2.1.6.2 Parameters:.....	21
69	3 SOAP Binding.....	23
70	3.1 ebXML Registry Service Interfaces: Abstract Definition.....	23
71	3.2 ebXML Registry Service Interfaces SOAP Binding.....	23
72	3.3 ebXML Registry Service Interfaces SOAP Service Template.....	24
73	3.4 Mapping of Exception to SOAP Fault.....	24

74	4 HTTP Binding.....	26
75	4.1 HTTP Interface URL Pattern.....	26
76	4.2 RPC Encoding URL.....	26
77	4.2.1 Standard URL Parameters.....	26
78	4.2.2 QueryManager Binding.....	27
79	4.2.2.1 Sample getRegistryObject Request.....	27
80	4.2.2.2 Sample getRegistryObject Response.....	27
81	4.2.2.3 Sample getRepositoryItem Request.....	28
82	4.2.2.4 Sample getRepositoryItem Response.....	28
83	4.2.3 LifeCycleManager HTTP Interface.....	28
84	4.3 Submitter Defined URL.....	28
85	4.3.1 Submitter defined URL Syntax.....	29
86	4.3.2 Assigning URL to a RegistryObject	29
87	4.3.3 Assigning URL to a Repository Item	30
88	4.4 File Path Based URL.....	30
89	4.4.1 File Folder Metaphor.....	30
90	4.4.2 File Path of a RegistryObject.....	30
91	4.4.2.1 File Path Example.....	30
92	4.4.3 Matching URL To Objects.....	31
93	4.4.4 URL Matches a Single Object.....	31
94	4.4.5 URL Matches Multiple Object.....	32
95	4.4.6 Directory Listing.....	32
96	4.4.7 Access Control In RegistryPackage Hierarchy.....	32
97	4.5 URL Resolution Algorithm.....	33
98	4.6 Security Consideration.....	33
99	4.7 Exception Handling.....	33
100	5 Lifecycle Management Protocols.....	34
101	5.1 Submit Objects Protocol.....	34
102	5.1.1 SubmitObjectsRequest.....	34
103	5.1.1.1 Syntax:.....	34
104	5.1.1.2 Parameters:.....	35
105	5.1.1.3 Returns:.....	35
106	5.1.1.4 Exceptions:.....	35
107	5.1.2 Unique ID Generation.....	35
108	5.1.3 ID Attribute And Object References.....	35
109	5.1.4 Audit Trail.....	36
110	5.1.5 Sample SubmitObjectsRequest.....	36
111	5.2 The Update Objects Protocol.....	36
112	5.2.1 UpdateObjectsRequest.....	37
113	5.2.1.1 Syntax:.....	37
114	5.2.1.2 Parameters:.....	37
115	5.2.1.3 Returns:.....	38
116	5.2.1.4 Exceptions:.....	38
117	5.2.2 Audit Trail.....	38
118	5.3 The Approve Objects Protocol.....	38
119	5.3.1 ApproveObjectsRequest.....	38
120	5.3.1.1 Syntax:.....	39

121	5.3.1.2 Parameters:.....	39
122	5.3.1.3 Returns:.....	39
123	5.3.1.4 Exceptions:.....	39
124	5.3.2 Audit Trail.....	39
125	5.4 The Deprecate Objects Protocol.....	39
126	5.4.1 DeprecateObjectsRequest.....	40
127	5.4.1.1 Syntax:.....	40
128	5.4.1.2 Parameters:.....	40
129	5.4.1.3 Returns:.....	40
130	5.4.1.4 Exceptions:.....	41
131	5.4.2 Audit Trail.....	41
132	5.5 The Undeprecate Objects Protocol.....	41
133	5.5.1 UndeprecateObjectsRequest.....	41
134	5.5.1.1 Syntax:.....	41
135	5.5.1.2 Parameters:.....	42
136	5.5.1.3 Returns:.....	42
137	5.5.1.4 Exceptions:.....	42
138	5.5.2 Audit Trail.....	42
139	5.6 The Remove Objects Protocol.....	42
140	5.6.1 RemoveObjectsRequest.....	43
141	5.6.1.1 Syntax:.....	43
142	5.6.1.2 Parameters:.....	43
143	5.6.1.3 Returns:.....	44
144	5.6.1.4 Exceptions:.....	44
145	5.7 Registry Managed Version Control.....	44
146	5.7.1 Version Controlled Resources.....	44
147	5.7.2 Versioning and Object Identification.....	44
148	5.7.3 Logical ID.....	44
149	5.7.4 Version Identification.....	45
150	5.7.4.1 Version Identification for a RegistryObject.....	45
151	5.7.4.2 Version Identification for a RepositoryItem.....	45
152	5.7.5 Versioning of ExtrinsicObject and Repository Items.....	45
153	5.7.5.1 ExtrinsicObject and Shared RepositoryItem.....	46
154	5.7.6 Versioning and Composed Objects.....	46
155	5.7.7 Versioning and References.....	46
156	5.7.8 Versioning and Audit Trail.....	47
157	5.7.9 Inter-versions Association.....	47
158	5.7.10 Client Initiated Version Removal.....	47
159	5.7.11 Registry Initiated Version Removal.....	47
160	5.7.12 Locking and Concurrent Modifications.....	47
161	5.7.13 Version Creation.....	47
162	5.7.14 Versioning Override.....	48
163	6 Query Management Protocols.....	49
164	6.1 Ad Hoc Query Protocol.....	49
165	6.1.1 AdhocQueryRequest.....	50
166	6.1.1.1 Syntax:.....	50
167	6.1.1.2 Parameters:.....	50

168	6.1.1.3 Returns:.....	51
169	6.1.1.4 Exceptions:.....	51
170	6.1.2 AdhocQueryResponse.....	51
171	6.1.2.1 Syntax:.....	51
172	6.1.2.2 Parameters:.....	51
173	6.1.3 AdhocQuery.....	52
174	6.1.3.1 Syntax:.....	52
175	6.1.3.2 Parameters:.....	52
176	6.1.4 ReponseOption.....	52
177	6.1.4.1 Syntax:.....	52
178	6.1.4.2 Parameters:.....	52
179	6.2 Iterative Query Support.....	53
180	6.2.1 Query Iteration Example.....	53
181	6.3 Stored Query Support.....	54
182	6.3.1 Submitting a Stored Query.....	54
183	6.3.1.1 Declaring Query Parameters.....	54
184	6.3.1.2 Canonical Context Parameters.....	55
185	6.3.2 Invoking a Stored Query.....	55
186	6.3.2.1 Specifying Query Invocation Parameters.....	55
187	6.3.3 Response to Stored Query Invocation.....	56
188	6.3.4 Access Control on a Stored Query.....	56
189	6.3.5 Canonical Query: Get Client's User Object.....	56
190	6.4 SQL Query Syntax.....	57
191	6.4.1 Relational Schema for SQL Queries.....	57
192	6.4.2 SQL Query Results.....	57
193	6.5 Filter Query Syntax.....	58
194	6.5.1 Filter Query Structure.....	58
195	6.5.2 Query Elements.....	58
196	6.5.3 Filter Elements.....	59
197	6.5.3.1 FilterType.....	60
198	6.5.3.2 SimpleFilterType.....	60
199	6.5.3.3 BooleanFilter.....	61
200	6.5.3.4 FloatFilter.....	61
201	6.5.3.5 IntegerFilter.....	61
202	6.5.3.6 DateTimeFilter.....	62
203	6.5.3.7 StringFilter.....	62
204	6.5.3.8 CompoundFilter.....	62
205	6.5.4 Nested Query Elements.....	63
206	6.5.5 Branch Elements.....	63
207	6.6 Query Examples.....	64
208	6.6.1 Name and Description Queries.....	64
209	6.6.2 Classification Queries.....	64
210	6.6.2.1 Retrieving ClassificationSchemes.....	65
211	6.6.2.2 Retrieving Children of Specified ClassificationNode.....	65
212	6.6.2.3 Retrieving Objects Classified By a ClassificationNode.....	65
213	6.6.2.4 Retrieving Classifications that Classify an Object.....	65
214	6.6.3 Association Queries.....	66
215	6.6.3.1 Retrieving All Associations With Specified Object As Source.....	66

216	6.6.3.2 Retrieving All Associations With Specified Object As Target.....	66
217	6.6.3.3 Retrieving Associated Objects Based On Association Type.....	66
218	6.6.3.4 Complex Association Query.....	67
219	6.6.4 Package Queries.....	67
220	6.6.5 ExternalLink Queries.....	67
221	6.6.6 Audit Trail Queries.....	68
222	7 Event Notification Protocols.....	69
223	7.1 Use Cases.....	69
224	7.1.1 CPP Has Changed.....	69
225	7.1.2 New Service is Offered.....	69
226	7.1.3 Monitor Download of Content.....	69
227	7.1.4 Monitor Price Changes.....	69
228	7.1.5 Keep Replicas Consistent With Source Object.....	69
229	7.2 Registry Events.....	69
230	7.3 Subscribing to Events.....	70
231	7.3.1 Event Selection.....	70
232	7.3.2 Notification Action.....	70
233	7.3.3 Subscription Authorization.....	71
234	7.3.4 Subscription Quotas.....	71
235	7.3.5 Subscription Expiration.....	71
236	7.3.6 Subscription Rejection.....	71
237	7.4 Unsubscribing from Events.....	71
238	7.5 Notification of Events.....	71
239	7.6 Retrieval of Events.....	72
240	7.7 Pruning of Events.....	72
241	8 Content Management Services.....	73
242	8.1 Content Validation.....	73
243	8.1.1 Content Validation: Use Cases.....	73
244	8.1.1.1 Validation of HL7 Conformance Profiles.....	73
245	8.1.1.2 Validation of Business Processes.....	73
246	8.1.1.3 Validation of UBL Business Documents.....	73
247	8.2 Content Cataloging.....	74
248	8.2.1 Content-based Discovery: Use Cases.....	74
249	8.2.1.1 Find All CPPs Where Role is "Buyer".....	74
250	8.2.1.2 Find All XML Schema's That Use Specified Namespace.....	74
251	8.2.1.3 Find All WSDL Descriptions with a SOAP Binding.....	74
252	8.3 Abstract Content Management Service.....	74
253	8.3.1 Inline Invocation Model.....	75
254	8.3.2 Decoupled Invocation Model.....	76
255	8.4 Content Management Service Protocol.....	77
256	8.4.1 ContentManagementServiceRequestType.....	77
257	8.4.1.1 Syntax:.....	77
258	8.4.1.2 Parameters:.....	78
259	8.4.1.3 Returns:.....	78
260	8.4.1.4 Exceptions:.....	78
261	8.4.2 ContentManagementServiceResponseType.....	78
262	8.4.2.1 Syntax:.....	78

263	8.4.2.2 Parameters:.....	78
264	8.5 Publishing / Configuration of a Content Management Service.....	79
265	8.5.1 Multiple Content Management Services and Invocation Control Files.....	80
266	8.6 Invocation of a Content Management Service.....	81
267	8.6.1 Resolution Algorithm For Service and Invocation Control File.....	81
268	8.6.2 Audit Trail and Cataloged Content.....	81
269	8.6.3 Referential Integrity.....	81
270	8.6.4 Error Handling.....	81
271	8.7 Validate Content Protocol.....	82
272	8.7.1 ValidateContentRequest.....	82
273	8.7.1.1 Syntax:.....	82
274	8.7.1.2 Parameters:.....	83
275	8.7.1.3 Returns:.....	83
276	8.7.1.4 Exceptions:.....	83
277	8.7.2 ValidateContentResponse.....	83
278	8.7.2.1 Syntax:.....	83
279	8.7.2.2 Parameters:.....	83
280	8.8 Catalog Content Protocol.....	84
281	8.8.1 CatalogContentRequest.....	84
282	8.8.1.1 Syntax:.....	84
283	8.8.1.2 Parameters:.....	85
284	8.8.1.3 Returns:.....	85
285	8.8.1.4 Exceptions:.....	85
286	8.8.2 CatalogContentResponse.....	85
287	8.8.2.1 Syntax:.....	85
288	8.8.2.2 Parameters:.....	86
289	8.9 Illustrative Example: Canonical XML Cataloging Service.....	86
290	8.10 Canonical XML Content Cataloging Service.....	87
291	8.10.1 Publishing of Canonical XML Content Cataloging Service.....	87
292	9 Cooperating Registries Support.....	88
293	9.1 Cooperating Registries Use Cases.....	88
294	9.1.1 Inter-registry Object References.....	88
295	9.1.2 Federated Queries.....	88
296	9.1.3 Local Caching of Data from Another Registry.....	88
297	9.1.4 Object Relocation.....	88
298	9.2 Registry Federations.....	89
299	9.2.1 Federation Metadata.....	89
300	9.2.2 Local Vs. Federated Queries.....	90
301	9.2.2.1 Local Queries.....	90
302	9.2.2.2 Federated Queries.....	90
303	9.2.2.3 Membership in Multiple Federations.....	91
304	9.2.3 Federated Lifecycle Management Operations.....	91
305	9.2.4 Federations and Local Caching of Remote Data.....	91
306	9.2.5 Caching of Federation Metadata.....	91
307	9.2.6 Time Synchronization Between Registry Peers.....	91
308	9.2.7 Federations and Security.....	92
309	9.2.8 Federation Lifecycle Management Protocols.....	92

310	9.2.8.1	Joining a Federation.....	92
311	9.2.8.2	Creating a Federation.....	92
312	9.2.8.3	Leaving a Federation.....	92
313	9.2.8.4	Dissolving a Federation.....	92
314	9.3	Object Replication.....	93
315	9.3.1	Use Cases for Object Replication.....	93
316	9.3.2	Queries And Replicas.....	94
317	9.3.3	Lifecycle Operations And Replicas.....	94
318	9.3.4	Object Replication and Federated Registries.....	94
319	9.3.5	Creating a Local Replica.....	94
320	9.3.6	Transactional Replication.....	94
321	9.3.7	Keeping Replicas Current.....	95
322	9.3.8	Lifecycle Management of Local Replicas.....	95
323	9.3.9	Tracking Location of a Replica.....	95
324	9.3.10	Remote Object References to a Replica.....	95
325	9.3.11	Removing a Local Replica.....	95
326	9.4	Object Relocation Protocol.....	95
327	9.4.1	RelocateObjectsRequest.....	98
328	9.4.1.1	Parameters:.....	98
329	9.4.1.2	Returns:.....	98
330	9.4.1.3	Exceptions:.....	98
331	9.4.2	AcceptObjectsRequest.....	98
332	9.4.2.1	Parameters:.....	99
333	9.4.2.2	Returns:.....	99
334	9.4.2.3	Exceptions:.....	99
335	9.4.3	Object Relocation and Remote ObjectRefs.....	99
336	9.4.4	Notification of Object Relocation To ownerAtDestination.....	100
337	9.4.5	Notification of Object Commit To sourceRegistry.....	100
338	9.4.6	Object Ownership and Owner Reassignment.....	100
339	9.4.7	Object Relocation and Timeouts.....	100
340	10	Registry Security.....	101
341	10.1	Security Use Cases.....	101
342	10.1.1	Identity Management.....	101
343	10.1.2	Message Security.....	101
344	10.1.3	Repository Item Security.....	101
345	10.1.4	Authentication.....	101
346	10.1.5	Authorization and Access Control.....	101
347	10.1.6	Audit Trail.....	101
348	10.2	Identity Management.....	102
349	10.3	Message Security.....	102
350	10.3.1	Transport Layer Security.....	102
351	10.3.2	SOAP Message Security.....	102
352	10.3.2.1	Request Message Signature.....	102
353	10.3.2.2	Response Message Signature.....	102
354	10.3.2.3	KeyInfo Requirements.....	103
355	10.3.2.4	Message Signature Validation.....	103
356	10.3.2.5	Message Signature Example.....	103

357	10.3.2.6 Message With RepositoryItem: Signature Example.....	104
358	10.3.2.7 SOAP Message Security and HTTP/S.....	105
359	10.3.3 Message Confidentiality.....	106
360	10.3.4 Key Distribution Requirements.....	106
361	10.4 Authentication.....	106
362	10.4.1 Registry as Authentication Authority.....	106
363	10.4.2 External Authentication Authority.....	107
364	10.4.3 Authenticated Session Support.....	107
365	10.5 Authorization and Access Control.....	107
366	10.6 Audit Trail.....	107
367	11 Registry SAML Profile.....	108
368	11.1 Terminology.....	108
369	11.2 Use Cases for SAML Profile.....	108
370	11.2.1 Registry as SSO Participant:	109
371	11.3 SAML Roles Played By Registry.....	109
372	11.3.1 Service Provider Role.....	109
373	11.3.1.1 Service Provider Requirements.....	109
374	11.4 Registry SAML Interface.....	110
375	11.5 Requirements for Registry SAML Profile	110
376	11.6 SSO Operation.....	111
377	11.6.1 Scenario Actors.....	111
378	11.6.2 SSO Operation – Unauthenticated HTTP Requestor.....	111
379	11.6.2.1 Scenario Sequence.....	112
380	11.6.3 SSO Operation – Authenticated HTTP Requestor.....	113
381	11.6.4 SSO Operation – Unauthenticated SOAP Requestor.....	113
382	11.6.4.1 Scenario Sequence.....	114
383	11.6.5 SSO Operation – Authenticated SOAP Requestor.....	114
384	11.6.5.1 Scenario Sequence.....	115
385	11.6.6 <samlp:AuthnRequest> Generation Rules.....	116
386	11.6.7 <samlp:Response> Processing Rules.....	116
387	11.6.8 Mapping Subject to User.....	116
388	11.7 External Users.....	117
389	12 Native Language Support (NLS).....	118
390	12.1 Terminology.....	118
391	12.2 NLS and Registry Protocol Messages.....	118
392	12.3 NLS Support in RegistryObjects	118
393	12.3.1 Character Set of LocalizedString.....	120
394	12.3.2 Language of LocalizedString.....	120
395	12.4 NLS and Repository Items	120
396	12.4.1 Character Set of Repository Items.....	120
397	12.4.2 Language of Repository Items.....	120
398	13 Conformance.....	121
399	13.1 Conformance Profiles.....	121
400	13.2 Feature Matrix.....	121
401	14 References.....	125
402	14.1 Normative References.....	125

403 14.2 Informative.....127
404

Illustration Index

Figure 1: Simplified View of ebXML Registry Architecture.....	16
Figure 2: Registry Protocol Request-Response Pattern.....	18
Figure 3: Example Registry Package Hierarchy.....	31
Figure 4: Example of a Directory Listing.....	32
Figure 5: Submit Objects Protocol.....	34
Figure 6: Update Objects Protocol.....	37
Figure 7: Approve Objects Protocol.....	38
Figure 8: Deprecate Objects Protocol.....	40
Figure 9: Undeprecate Objects Protocol.....	41
Figure 10: Remove Objects Protocol.....	43
Figure 11: Ad Hoc Query Protocol.....	50
Figure 12: Filter Type Hierarchy.....	60
Figure 13: Content Validation Service.....	73
Figure 14: Content Cataloging Service.....	74
Figure 15: Content Management Service: Inline Invocation Model.....	76
Figure 16: Content Management Service: Decoupled Invocation Model.....	77
Figure 17: Cataloging Service Configuration.....	80
Figure 18: Validate Content Protocol.....	82
Figure 19: Catalog Content Protocol.....	84
Figure 20: Example of CPP cataloging using Canonical XML Cataloging Service.....	86
Figure 21: Inter-registry Object References.....	88
Figure 22: Registry Federations.....	89
Figure 23: Federation Metadata Example.....	90
Figure 24: Object Replication.....	93
Figure 25: Object Relocation.....	96
Figure 26: Relocate Objects Protocol.....	97
Figure 27: SAML SSO Typical Scenario.....	109
Figure 28: SSO Operation – Unauthenticated HTTP Requestor.....	112
Figure 29: SSO Operation - Unauthenticated SOAP Requestor.....	114
Figure 30: SSO Operation - Authenticated SOAP Requestor.....	115

1 Introduction

406

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

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

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

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

1.1 Audience

417

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

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

1.2 Terminology

421

422 The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT,
423 RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in IETF RFC
424 2119 [RFC2119].

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

1.3 Notational Conventions

428

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

1.3.1 UML Diagrams

431

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

1.3.2 Identifier Placeholders

434

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

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

440

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

1.3.3 Constants

442

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

445 1.3.4 Bold Text

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

449

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

454

455 1.3.5 Example Values

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

458

```
459 <rim:Slot name="urn:oasis:names:tc:ebxml-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 text
467 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 specification, the
469 schema documents take precedence. Note that in some cases the normative text of this specification
470 imposes constraints beyond those indicated by the schema documents.

471 Conventional XML namespace prefixes are used throughout this specification to stand for their respective
472 namespaces as follows, whether or not a namespace declaration is present in the example. The use of
473 these namespace prefixes in instance documents is non-normative. However, for consistency and
474 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 it's 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 as
486 such supports an open-ended and broad set of use cases. The following are some common use cases that
487 are being addressed by ebXML Registry.

- 488 • Web Services Registry: publish, management, discovery and reuse of web service discriptions in WSDL,
489 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 Process
494 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.
500

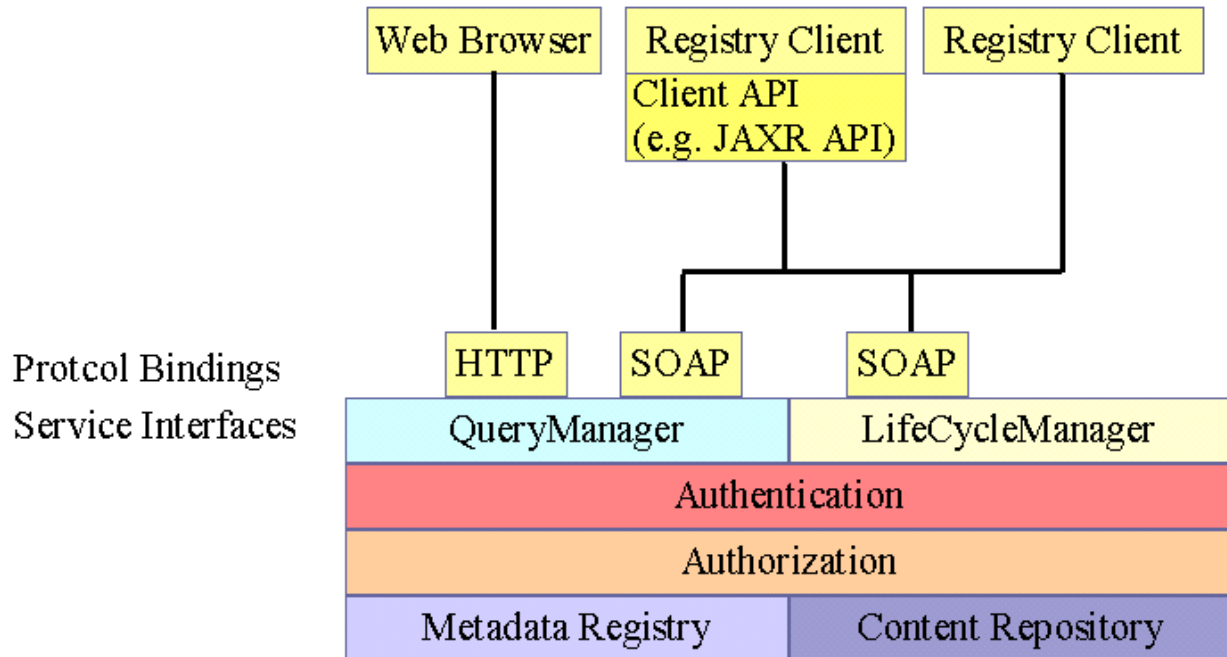


Figure 1: Simplified View of ebXML Registry Architecture

502 1.7.1 Registry Clients

503 A Registry Client is a software program that interacts with the registry using registry protocols. The Registry
 504 Client MAY be a Graphical User Interface (GUI), software service or agent. The Registry Client typically
 505 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 may
 507 accessed by a web browser. In either case the Registry Client interacts with the registry using registry
 508 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 such
 511 as the Java API for XML Registries [JAXR] to access the registry. Client APIs such as [JAXR] provide
 512 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 and
 517 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 interfaces
 521 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 the
 524 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 Service
527 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 protocol.
529 Additional bindings may be defined in the future as needed by the community.

530 **1.7.4 Authentication and Authorization**

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

535 **1.7.5 Metadata Registry and Content Repository**

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

2 Registry Protocols

540

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

2.1 Requests and Responses

543

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

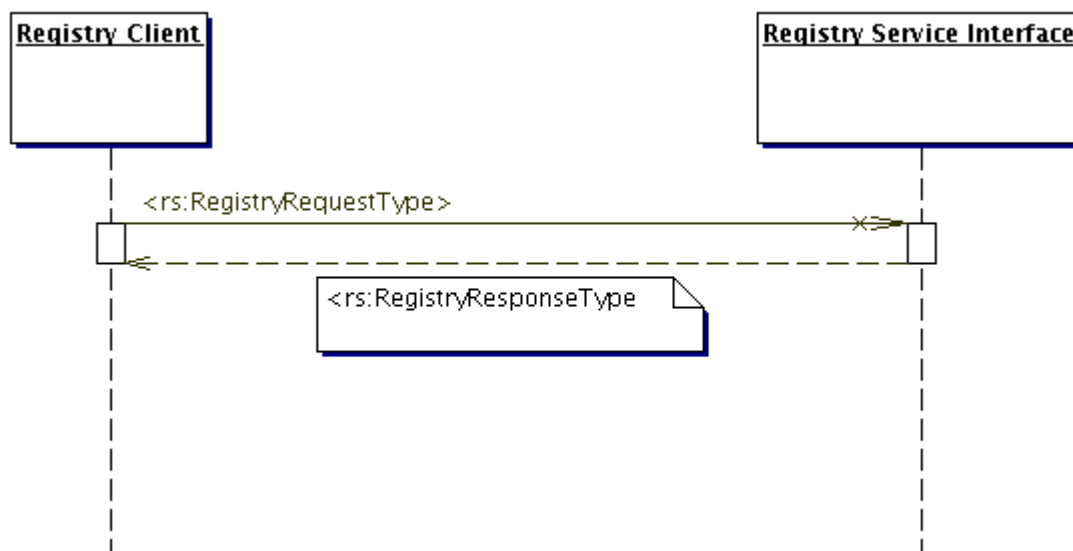


Figure 2: Registry Protocol Request-Response Pattern

548

549 Throughout this section, text mentions of elements and types are indicated with a namespace prefix. The
550 namespace prefix conventions are defined in the “Introduction” chapter.

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

2.1.1 RegistryRequestType

557

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

2.1.1.1 Syntax:

559

```
560 <complexType name="RegistryRequestType">
561   <sequence>
562     <!-- every request may be extended using Slots. -->
563     <element maxOccurs="1" minOccurs="0" name="RequestSlotList"
564 type="rim:SlotListType"/>
565   </sequence>
566   <attribute name="id" type="anyURI" use="required"/>
567   <!--Comment may be used by requestor to describe the request. Used in
568 VersionInfo.comment-->
```

569
570
571

```
<attribute name="comment" type="string" use="optional"/>
</complexType>
<element name="RegistryRequest" type="tns:RegistryRequestType"/>
```

572 2.1.1.2 Parameters:

- 573 ▪ **comment:** This parameter allows the requestor to specify a string value that describes the
574 action being performed by the request. This parameter is used by the “Registry Managed
575 Version Control” feature of the registry.
- 576 ▪ **id:** This parameter specifies a request identifier that is used by the corresponding
577 response to correlate the response with its request. It MAY also be used to correlate a
578 request with another related request. The value of the id parameter MUST abide by the
579 same constraints as the value of the id attribute for the <rim:IdentifiableType> type.
- 580 ▪ **RequestSlotList:** This parameter specifies a collection of Slot instances. A
581 RegistryRequestType MAY include Slots as an extensibility mechanism that provides a
582 means of adding additional attributes to the request in form of Slots. The use of registry
583 implementation specific slots MUST be ignored silently by a registry that does not support
584 such Slots and MAY not be interoperable across registry implementations.
585

586 2.1.1.3 Returns:

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

588 2.1.1.4 Exceptions:

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

- 590 ▪ **AuthorizationException:** Indicates that the requestor attempted to perform an operation
591 for which he or she was not authorized.
- 592 ▪ **InvalidRequestException:** Indicates that the requestor attempted to perform an operation
593 that was semantically invalid.
- 594 ▪ **SignatureValidationException:** Indicates that a Signature specified for the request failed
595 to validate.
- 596 ▪ **TimeoutException:** Indicates that the processing time for the request exceeded a registry
597 specific limit.
- 598 ▪ **UnsupportedCapabilityException:** Indicates that this registry did not support the
599 capability required to service the request.

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

603 2.1.2 RegistryRequest

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

607 2.1.3 RegistryResponseType

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

609 2.1.3.1 Syntax:

610
611
612

```
<complexType name="RegistryResponseType">
  <sequence>
    <!-- every response may be extended using Slots. -->
```

```

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

```

623 2.1.3.2 Parameters:

- 624 ▪ **status:** The status attribute is used to indicate the status of the request. The value of the
625 status attribute MUST be a reference to a ClassificationNode within the canonical
626 ResponseStatusType ClassificationScheme as described in [ebRIM]. A Registry MUST
627 support the status types as defined by the canonical ResponseStatusType
628 ClassificationScheme. The canonical ResponseStatusType ClassificationScheme may be
629 extended by adding additional ClassificationNodes to it.
- 630 The following canonical values are defined for the ResponseStatusType
631 ClassificationScheme:
- 632 • **Success** - This status specifies that the request was successful.
 - 633 • **Failure** - This status specifies that the request encountered a failure. One or more
634 errors MUST be included in the RegistryErrorList in this case or returned as a SOAP
635 Fault.
 - 636 • **Unavailable** – This status specifies that the response is not yet available. This may be
637 the case if this RegistryResponseType represents an immediate response to an
638 asynchronous request where the actual response is not yet available.
 - 639 ▪ **requestId:** This parameter specifies the id of the request for which this is a response. It
640 matches value of the id attribute of the corresponding RegistryRequestType.
 - 641 ▪ **ResponseSlotList:** This parameter specifies a collection of Slot instances. A
642 RegistryResponseType MAY include Slots as an extensibility mechanism that provides a
643 means of adding dynamic attributes in form of Slots. The use of registry implementation
644 specific slots MUST be ignored silently by a Registry Client that does not support such
645 Slots and MAY not be interoperable across registry implementations.
 - 646 ▪ **RegistryErrorList:** This parameter specifies an optional collection of RegistryError
647 elements in the event that there are one or more errors that were encountered while the
648 registry processed the request for this response. This is described in more detail in 6.9.4.

649 2.1.4 RegistryResponse

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

653 2.1.5 RegistryErrorList

654 A RegistryErrorList specifies an optional collection of RegistryError elements in the event that there are one
655 or more errors that were encountered while the registry processed a request.

656 2.1.5.1 Syntax:

```

657 <element name="RegistryErrorList">
658 <complexType>

```

```

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

```

669 2.1.5.2 Parameters:

- 670 ▪ *highestSeverity*: This parameter specifies the *ErrorType* for the highest severity
- 671 *RegistryError* in the *RegistryErrorList*. Values for *highestSeverity* are defined by *ErrorType*
- 672 in .
- 673 ▪ *RegistryError*: A *RegistryErrorList* has one or more *RegistryErrors*. A *RegistryError* specifies
- 674 an error or warning message that is encountered while the registry processes a request.
- 675 *RegistryError* is defined in 2.1.6.
- 676

677 2.1.6 RegistryError

678 A *RegistryError* specifies an error or warning message that is encountered while the registry processes a
679 request.

680 2.1.6.1 Syntax:

```

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

```

695 2.1.6.2 Parameters:

- 696 ▪ *codeContext*: This attribute specifies a string that indicates contextual text that provides
- 697 additional detail to the *errorCode*. For example, if the *errorCode* is *InvalidRequestException*
- 698 the *codeContext* MAY provide the reason why the request was invalid.
- 699 ▪ *errorCode*: This attribute specifies a string that indicates the error that was encountered.
- 700 Implementations MUST set this attribute to the *Exception* or *Error* as defined by this
- 701 specification (e.g. *InvalidRequestException*).
- 702 ▪ *severity*: This attribute indicates the severity of error that was encountered. The value of the
- 703 *severity* attribute MUST be a reference to a *ClassificationNode* within the canonical
- 704 *ErrorSeverityType ClassificationScheme* as described in [ebRIM]. A *Registry* MUST support
- 705 the error severity types as defined by the canonical *ErrorSeverityType*
- 706 *ClassificationScheme*. The canonical *ErrorSeverityType ClassificationScheme* may be

707
708
709
710
711
712
713
714
715
716
717
718

extended by adding additional ClassificationNodes to it.

The following canonical values are defined for the ErrorSeverityType ClassificationScheme:

- **Error** – An Error is a fatal error encountered by the registry while processing a request. A registry **MUST** return a status of Failure in the RegistryResponse for a request that encountered Errors during its processing.
- **Warning** – A Warning is a non-fatal error encountered by the registry while processing a request. A registry **MUST** return a status of Success in the RegistryResponse for a request that only encountered Warnings during its processing and encountered no Errors.
- **location**: This attribute specifies a string that indicated where in the code the error occurred. Implementations **SHOULD** show the stack trace and/or, code module and line number information where the error was encountered in code.

3 SOAP Binding

719

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

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

3.1 ebXML Registry Service Interfaces: Abstract Definition

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

- 727 • A portType is defined for each Service Interface:

728

729

```
730 <portType name="QueryManagerPortType">  
731 ...  
732 </portType>  
733 <portType name="LifecycleManagerPortType">  
734 ...  
735 </portType>
```

736

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

738

```
739 <portType name="QueryManagerPortType">  
740 <operation name="submitAdhocQuery">  
741 ...  
742 </operation>  
743 </portType>
```

744

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

747

```
748 <portType name="QueryManagerPortType">  
749 <operation name="submitAdhocQuery">  
750 <input message="tns:msgAdhocQueryRequest"/>  
751 <output message="tns:msgAdhocQueryResponse"/>  
752 </operation>  
753 </portType>
```

754

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

758

```
759 <message name="msgAdhocQueryRequest">  
760 <part element="query:AdhocQueryRequest"  
761 name="partAdhocQueryRequest"/>  
762 </message>  
763 <message name="msgAdhocQueryResponse">  
764 <part element="query:AdhocQueryResponse"  
765 name="partAdhocQueryResponse"/>  
766 </message>
```

766 3.2 ebXML Registry Service Interfaces SOAP Binding

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

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

776

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

793

794 3.3 ebXML Registry Service Interfaces SOAP Service Template

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

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

805

```
806     <service name="ebXMLRegistrySOAPService">  
807       <port binding="bindings:QueryManagerBinding" name="QueryManagerPort">  
808         <soap:address location="http://your.server.com/soap"/>  
809       </port>  
810       <port binding="bindings:LifeCycleManagerBinding"  
811       name="LifeCycleManagerPort">  
812         <soap:address location="http://your.server.com/soap"/>  
813       </port>  
814     </service>
```

815

816 **3.4 Mapping of Exception to SOAP Fault**

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

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

823

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/omars/registry</i>

Table 1: Mapping a Registry Exception to SOAP Fault

4 HTTP Binding

824

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

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

829

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

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

837

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

838

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

4.2 RPC Encoding URL

843

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

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

4.2.1 Standard URL Parameters

850

851 The following table specifies the URL parameters supported by RPC Encoding URLs. A Registry MAY
852 implement additional URL parameters in addition to these parameters. Note that the URL Parameter names

853 MUST be processed by the registry in a case-insensitive manner while the parameter values MUST be
 854 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= urn:freebxml:registry:demoDB :ExtrinsicObject:zeusDescription

Table 3: Standard URL Parameters

855 4.2.2 QueryManager Binding

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

859

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

860

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

863 4.2.2.1 Sample getRegistryObject Request

864 The following example shows a getRegistryObject request.

865

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

868

869 4.2.2.2 Sample getRegistryObject Response

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

872

```
873 HTTP/1.1 200 OK  
874 Content-Type: text/xml  
875 Content-Length: 555
```

876

```
877 <?xml version="1.0"?>
878 <ExtrinsicObject
879   id =
880   "urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription"
881   objectType="{OBJECT_TYPE}">
882   ...
883 </ExtrinsicObject>
```

884

885 **4.2.2.3 Sample getRepositoryItem Request**

886 The following example shows a getRepositoryItem request.

887

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

890

891 **4.2.2.4 Sample getRepositoryItem Response**

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

894

```
895 HTTP/1.1 200 OK
896 Content-Type: text/xml
897 Content-Length: 555
898
899 <?xml version="1.0"?>
900 <CollaborationProtocolProfile>
901   ...
902 </CollaborationProtocolProfile>
```

903

904 **4.2.3 LifeCycleManager HTTP Interface**

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

908 **4.3 Submitter Defined URL**

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

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

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

920

921 `http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&metho`
922 `d=getRegistryObject¶m-`
923 `id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription`
924
925
926

927 The same RegistryObject (an ExtrinsicObject) may be accessed via the following Submitter defined URL:
928

929 `http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.xml`
930
931
932

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

935 `http://localhost:8080/ebxmlrr/registry/http/?interface=QueryManager&metho`
936 `d=getRepositoryItem¶m-`
937 `id=urn:freebxml:registry:demoDB:ExtrinsicObject:zeusDescription`
938
939
940
941

942 The same repository item may be accessed via the following Submitter defined URL:
943

944 `http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus.jpg`
945
946
947

948 **4.3.1 Submitter defined URL Syntax**

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

952 The use of relative URLs is illustrated as follows:

- 953 • **Base URL for Registry:** <http://localhost:8080/ebxml/registry>
- 954 • **Implied Prefix URL for HTTP interface:** <http://localhost:8080/ebxml/registry/http>
- 955 • **Submitter Defined URL suffix:** /pictures/nikola/zeus
- 956 • **Complete URL:** <http://localhost:8080/ebxmlrr/registry/http/pictures/nikola/zeus>

957 **4.3.2 Assigning URL to a RegistryObject**

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

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

961 `urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:locator`
962
963
964

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

969 **4.3.3 Assigning URL to a Repository Item**

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

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

973

974

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

976

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

981 **4.4 File Path Based URL**

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

985 **4.4.1 File Folder Metaphor**

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

991 In this file-folder metaphor, a Submitter creates a RegistryPackage to create the functional equivalent of a
992 folder and creates a RegistryObject to create the functional equivalent of a file. The Submitter adds a
993 RegistryObjects as a member of a RegistryPackage to create the functional equivalent of adding a file to a
994 folder.

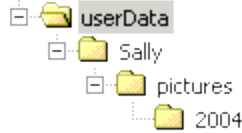
995 **4.4.2 File Path of a RegistryObject**

996 Each RegistryObject has an implicit *file path*. The file path of a RegistryObject is a path structure similar to
997 the Unix file path structure. The file path is composed of file path segments. Analogous to the Unix file path,
998 the last segment within the file path represents the RegistryObject, while preceding segments represent the
999 RegistryPackage(s) within the membership hierarchy of the RegistryObject. Each segment consists of the
1000 *name* of the RegistryPackage or the RegistryObject. Because the name attribute is of type
1001 InternationalString the path segment matches the name of an object within a specific locale.

1002 **4.4.2.1 File Path Example**

1003 Consider the example where a registry has a RegistryPackage hierarchy as illustrated below using the
1004 name of the objects in locale "en_US":

1005



1006

Figure 3: Example Registry Package Hierarchy

1007 Now let us assume that the RegistryPackage named “2004” has an ExtrinsicObject named “baby.gif” for a
 1008 repository item that is a photograph in the GIF format. In this example the file paths for various objects in
 1009 locale “en_US” are shown in table below:

1010

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

1011 Note that above example assumes that the RegistryPackage named userData is a root level package (not
 1012 contained within another RegistryPackage).

1013 4.4.3 Matching URL To Objects

1014 A registry client MAY access RegistryObjects and RepositoryItems over the HTTP GET request using URL
 1015 patterns that are based upon the File Path for the target objects. This section describes how a registry
 1016 resolves File Path URLs specified by an HTTP client.

1017 The registry MUST process each path segment from the beginning of the path to the end and for each path
 1018 segment match the segment to the value attribute of a LocalizedString in the name attribute of a
 1019 RegistryObject. For all but the last path segment, the matched RegistryObject MUST be a RegistryPackage.
 1020 The last path segment MAY match any RegistryObject including a RegistryPackage. If any path segment
 1021 fails to be matched then the URL is not resolvable by the File Path based URL method. When matching any
 1022 segment other than the first segment the registry MUST also ensure that the matched RegistryObject is a
 1023 member of the RegistryPackage that matches the previous segment.

1024 4.4.4 URL Matches a Single Object

1025 When a File Path based URL matches a single object there are two possible responses.

1026

- 1027 • If the URL pattern does not end in a '/' character or the last segment does not match a
 1028 RegistryPackage then the Registry MUST send as response an XML document that is the
 1029 XML representation of the RegistryObject that matches the last segment. If the last
 1030 segment matches an ExtrinsicObject then if the URL specifies the HTTP GET parameter
 1031 with name 'getRepositoryItem' and value of 'true' then the registry MUST return as
 1032 response the repository item associated with the ExtrinsicObject.
- 1033 • If the URL pattern ends in a '/' character and the last segment matches a RegistryPackage
 1034 then the Registry MUST send as response an HTML document that is the directory listing
 1035 (section 4.4.6) of all RegistryObjects that are members of the RegistryPackage that
 1036 matches the last segment.

1037

1038 4.4.5 URL Matches Multiple Object

1039 A registry MUST show a partial Directory Listing of a Registry Package when a File Path

1040 based URL matches multiple objects.
1041 A File Path based URL may match multiple objects if:

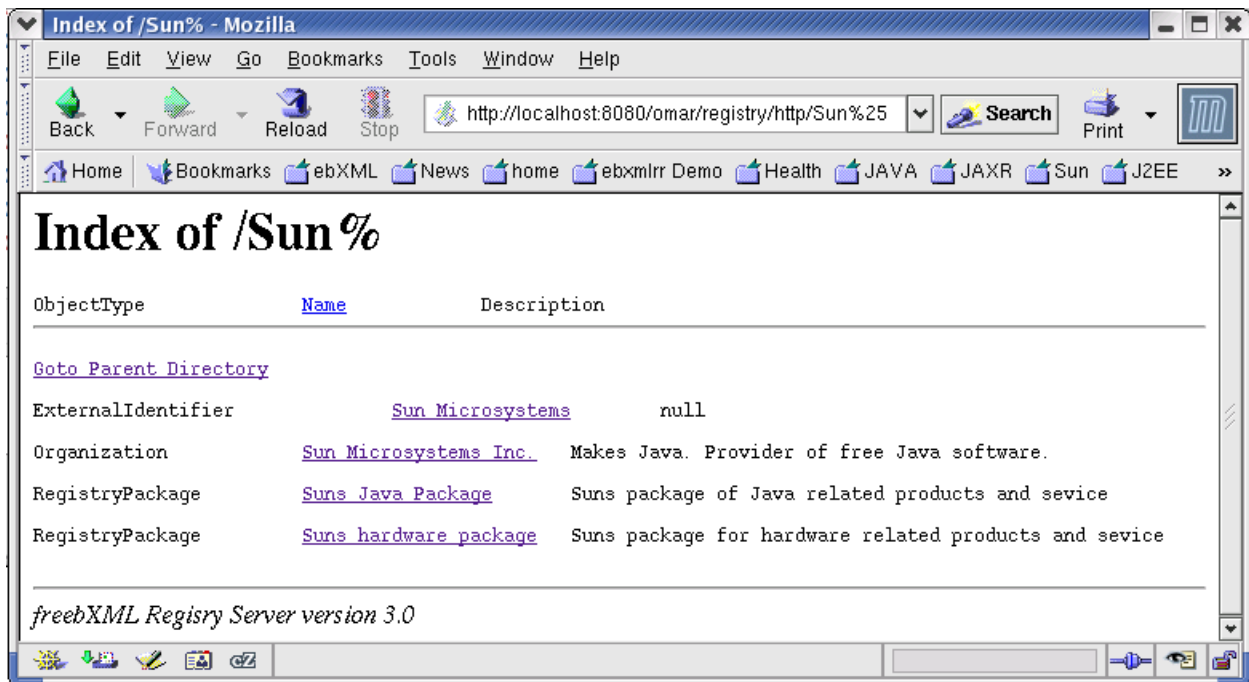
- 1042
- 1043 • Multiple objects with the same name exist in the same RegistryPackage
 - 1044 • The segment contains wildcard characters such as '%' or '?' to match the names of multiple objects
1045 within the same RegistryPackage. Note that wildcard characters must be URL encoded as defined
1046 by the HTTP protocol. For example the '%' character is encoded as '%25'.
1047

1048 4.4.6 Directory Listing

1049 A registry MUST return a directory listing as a response under certain circumstances as describes earlier.
1050 The directory listing MUST show a list of objects within a specific RegistryPackage.

1051 A registry SHOULD structure a directory listing such that each item in the listing provides information about
1052 a RegistryObject within the RegistryPackage. A registry MAY format its directory listing page in a registry
1053 specific manner. However, it is suggested that a registry SHOULD format it as an HTML page that minimally
1054 includes the objectType, name and description attributes for each RegistryObject in the directory listing.

1055 Figure 4 shows a non-normative example of a directory listing that matches all root level objects that have a
1056 name that begins with 'Sun' (path /Sun%25).
1057



1058
Figure 4: Example of a Directory Listing

1059 4.4.7 Access Control In RegistryPackage Hierarchy

1060 The ability to control who can add files and sub-folders to a folder is important in a file system. The same is
1061 true for the File Path Based URL mechanism.

1062 A Submitter MAY assign a custom Access Control Policy to a Registry Package to create the functional
1063 equivalent of assigning access control to a folder in the file-folder metaphor. The custom Access Control
1064 Policy SHOULD use the "reference" action to control who can add RegistryObjects as members of the folder
1065 as described in [ebRIM].

1066 **4.5 URL Resolution Algorithm**

1067 Since the HTTP Binding supports multiple mechanisms to resolve an HTTP URL a registry SHOULD
1068 implement an algorithm to determine the correct HTTP Binding mechanism to resolve a URL.

1069 This section gives a non-normative URL resolution algorithm that a registry SHOULD use to determine
1070 which of the various HTTP Binding mechanisms to use to resolve an HTTP URL.

1071 Upon receiving an HTTP GET request a registry SHOULD first check if the URL is an RPC Encoded URL.
1072 This MAY be done by checking if the *interface* URL parameter is specified in the URL. If specified the
1073 registry SHOULD resolve the URL using the RPC Encoded URL method as defined by section 4.2. If the
1074 *interface* URL parameter is not specified then the registry SHOULD use the Submitter specified URL
1075 method to check if the URL is resolvable. If the URL is still unresolvable then the registry SHOULD check if
1076 the URL is resolvable using the File Path based URL method. If the URL is still unresolvable then the
1077 registry should return an HTTP 404 (NotFound) error as defined by the HTTP protocol.

1078 **4.6 Security Consideration**

1079 A registry MUST enforce all Access Control Policies including restriction on the READ action when
1080 processing a request to the HTTP binding of a service interface. This implies that a Registry MUST not
1081 resolve a URL to a RegistryObject or RepositoryItem if the client is not authorized to read that object.

1082 **4.7 Exception Handling**

1083 If a service interface method generates an Exception it MUST be reported in a `RegistryErrorList`, and
1084 sent back to the client within the HTTP response for the HTTP request.

1085 When errors occur, the HTTP status code and message SHOULD correspond to the error(s) being reported
1086 in the `RegistryErrorList`. For example, if the `RegistryErrorList` reports that an object wasn't
1087 found, therefore cannot be returned, an appropriate error code SHOULD be 404, with a message of
1088 "ObjectNotFoundException". A detailed list of HTTP status codes can be found in [RFC2616]. The mapping
1089 between registry exceptions and HTTP status codes is currently unspecified.

5 Lifecycle Management Protocols

1090

1091 This section defines the protocols supported by Lifecycle Management service interface of the Registry. The
1092 Lifecycle Management protocols provide the functionality required by RegistryClients to manage the
1093 lifecycle of RegistryObjects and RepositoryItems within the registry.

1094 The XML schema for the Lifecycle Management protocols is described in [RR-LCM-XSD].

5.1 Submit Objects Protocol

1095

1096 This SubmitObjects allows a RegistryClient to submit one or more RegistryObjects and/or repository items.

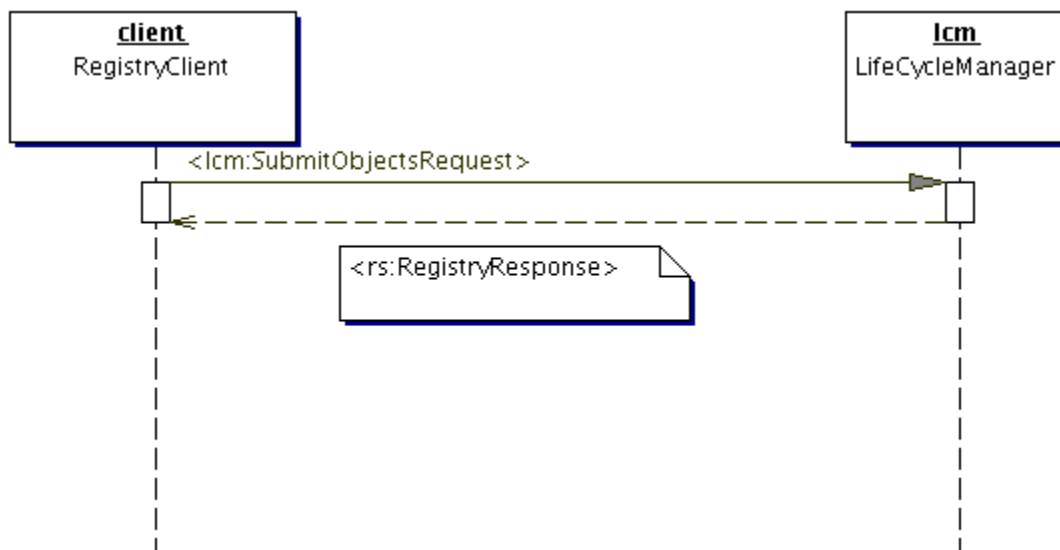


Figure 5: Submit Objects Protocol

1098

5.1.1 SubmitObjectsRequest

1099

1100 The SubmitObjectsRequest is used by a client to submit RegistryObjects and/or repository items to the
1101 registry.

5.1.1.1 Syntax:

1102

```
1103 <element name="SubmitObjectsRequest">  
1104   <complexType>  
1105     <complexContent>  
1106       <extension base="rs:RegistryRequestType">  
1107         <sequence>  
1108           <element ref="rim:RegistryObjectList"/>  
1109         </sequence>  
1110       </extension>  
1111     </complexContent>  
1112   </complexType>  
1113 </element>
```

5.1.1.2 Parameters:

1114

- 1115 **RegistryObjectList:** This parameter specifies a collection of RegistryObject instances that

1116 are being submitted to the registry. The RegistryObjects in the list may be brand new
1117 objects being submitted to the registry or they may be current objects already existing in
1118 the registry. In case of existing objects the registry MUST treat them in the same manner as
1119 UpdateObjectsRequest and simply update the existing objects.

1120 **5.1.1.3 Returns:**

1121 This request returns a RegistryResponse. See section 2.1.4 for details.

1122 **5.1.1.4 Exceptions:**

1123 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be
1124 returned:

- 1125 ▪ *UnresolvedReferenceException*: Indicates that the requestor referenced an object within
1126 the request that was not resolved during the processing of the request.
- 1127 ▪ *UnsignedRepositoryItemException*: Indicates that the requestor attempted to submit a
1128 RepositoryItem that was not signed.
- 1129 ▪ *QuotaExceededException*: Indicates that the requestor attempted to submit more content
1130 than the quota allowed for them by the registry.

1131 **5.1.2 Unique ID Generation**

1132 A Submitter MUST supply the id attribute for submitted objects. If the id is not specified then the registry
1133 MUST return an InvalidRequestException.

1134 If the id and lid match the id and lid of an existing RegistryObject within the home registry, then the registry
1135 MUST treat it as an Update action upon the existing RegistryObject.

1136 If the id matches the id of an existing RegistryObject within the home registry but the lid does not match that
1137 existing object's lid, then the registry MUST return an InvalidRequestException.

1138 If the lid matches the lid of an existing RegistryObject within the home registry but the id does not match that
1139 existing object's id, then the registry MUST create the newly submitted object as a new version of the
1140 existing object.

1141 If the Submitter supplies the id and it is a valid URN then the registry MUST honor the Submitter-supplied id
1142 value and use it as the value of the id attribute of the object in the registry. If the id is not a valid URN then
1143 the registry MUST treat it as a temporary id and replace it, and all references to it within the request, with a
1144 registry generated universally unique id. A registry generated universally unique id value MUST conform to
1145 the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID]:

1146 (e.g. *urn:uuid:a2345678-1234-1234-123456789012*)

1147

1148 **5.1.3 ID Attribute And Object References**

1149 The id attribute of an object MAY be used by other objects to reference that object. Within a
1150 SubmitObjectsRequest, the id attribute MAY be used to refer to an object within the same
1151 SubmitObjectsRequest as well as to refer to an object within the registry. An object in the
1152 SubmitObjectsRequest that needs to be referred to within the request document MAY be assigned an id by
1153 the submitter so that it can be referenced within the request. The submitter MAY give the object a valid
1154 URN, in which case the id is permanently assigned to the object within the registry. Alternatively, the
1155 submitter MAY assign an arbitrary id that is not a valid URN as long as the id is a unique anyURI value
1156 within the request document. In this case the id serves as a linkage mechanism within the request document
1157 but MUST be replaced with a registry generated id upon submission.

1158 When an object in a SubmitObjectsRequest needs to reference an object that is already in the registry, the
1159 request MAY contain an ObjectRef whose id attribute is the id of the object in the registry. This id is by
1160 definition a valid URN. An ObjectRef MAY be viewed as a proxy within the request for an object that is in the
1161 registry.

1162 **5.1.4 Audit Trail**

1163 The registry MUST create a single AuditableEvent object with eventType *Created* for all the RegistryObjects
1164 created by a SubmitObjectsRequest.

1165 **5.1.5 Sample SubmitObjectsRequest**

1166 The following example shows a simple SubmitObjectsRequest that submits a single Organization object to
1167 the registry. It does not show the complete SOAP Message with the message header and additional
1168 payloads in the message for the repository items.

```
1169  
1170 <lcm:SubmitObjectsRequest>  
1171   <rim:RegistryObjectList>  
1172     <rim:Organization lid="{LOGICAL_ID}"  
1173       id="{ID}"  
1174       primaryContact="{CONTACT_USER_ID}">  
1175       <rim:Name>  
1176         <rim:LocalizedString value="Sun Microsystems Inc." xml:lang="en-  
1177         US"/>  
1178       </rim:Name>  
1179       <rim:Address city="Burlington" country="USA" postalCode="01867"  
1180       stateOrProvince="MA" street="Network Dr." streetNumber="1"/>  
1181       <rim:TelephoneNumber areaCode="781" countryCode="1" number="123-  
1182       456" phoneType="office"/>  
1183     </rim:Organization>  
1184   </rim:RegistryObjectList>  
1185 </SubmitObjectsRequest>
```

1186 **5.2 The Update Objects Protocol**

1187 The UpdateObjectsRequest protocol allows a Registry Client to update one or more existing
1188 RegistryObjects and/or repository items in the registry.

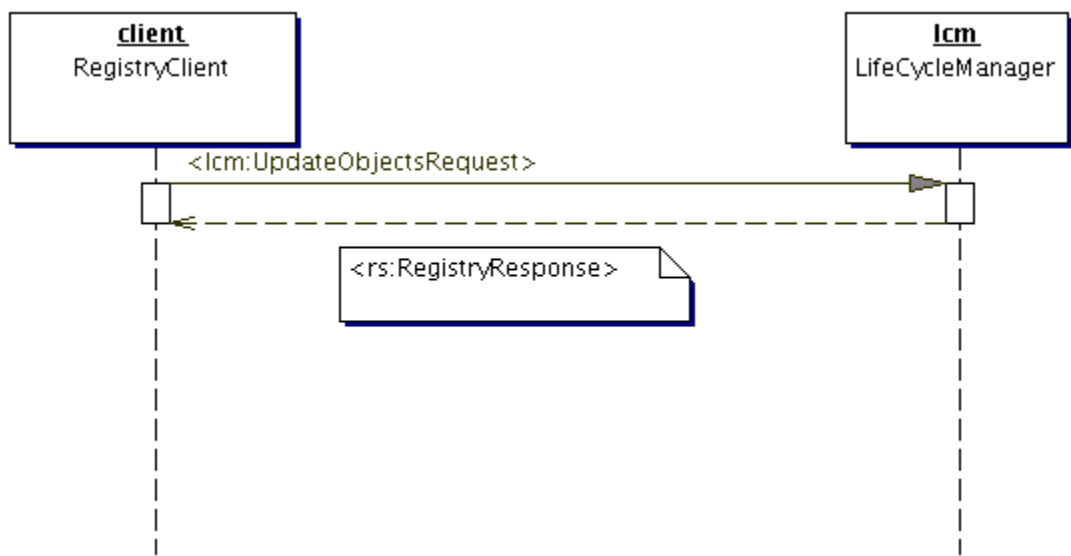


Figure 6: Update Objects Protocol

1190

1191 5.2.1 UpdateObjectsRequest

1192 The UpdateObjectsRequest is used by a client to update RegistryObjects and/or repository items that
1193 already exist within the registry.

1194 5.2.1.1 Syntax:

```
1195 <element name="UpdateObjectsRequest">  
1196   <complexType>  
1197     <complexContent>  
1198       <extension base="rs:RegistryRequestType">  
1199         <sequence>  
1200           <element ref="rim:RegistryObjectList"/>  
1201         </sequence>  
1202       </extension>  
1203     </complexContent>  
1204   </complexType>  
1205 </element>
```

1206 5.2.1.2 Parameters:

1207

- 1208 *RegistryObjectList*: This parameter specifies a collection of RegistryObject instances that
1209 are being updated within the registry. All immediate RegistryObject children of the
1210 RegistryObjectList MUST be current RegistryObjects already in the registry.
1211 RegistryObjects MUST include all required attributes, even those the user does not intend
1212 to change. A missing attribute MUST be interpreted as a request to set that attribute to
1213 NULL or in case it has a default value, the default value will be assumed. If this collection
1214 contains an immediate child RegistryObject that does not already exist in the registry, then
1215 the registry MUST return an InvalidRequestException. If the user wishes to submit a mix of
1216 new and updated objects then he or she SHOULD use a SubmitObjectsRequest.
1217 If an ExtrinsicObject is being updated and no RepositoryItem is provided in the
1218 UpdateObjectsRequest then the registry MUST maintain any previously existing
1219 RepositoryItem associated with the original ExtrinsicObject with the updated
1220 ExtrinsicObject. If the client wishes to remove the RepositoryItem from an existing
1221 ExtrinsicObject they MUST use a RemoveObjectsRequest with
1222 deletionScope=DeleteRepositoryItemOnly.

1223 5.2.1.3 Returns:

1224 This request returns a RegistryResponse. See section 2.1.4 for details.

1225 5.2.1.4 Exceptions:

1226 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be
1227 returned:

1228

- 1229 *UnresolvedReferenceException*: Indicates that the requestor referenced an object within
1230 the request that was not resolved during the processing of the request.
- 1231 *UnsignedRepositoryItemException*: Indicates that the requestor attempted to submit a
1232 RepositoryItem that was not signed.
- 1233 *QuotaExceededException*: Indicates that the requestor attempted to submit more content
than the quota allowed for them by the registry.

1234 5.2.2 Audit Trail

1235 The registry MUST create a single AuditableEvent object with eventType *Updated* for all RegistryObjects
1236 updated via an UpdateObjectsRequest.

1237 5.3 The Approve Objects Protocol

1238 The Approve Objects protocol allows a client to approve one or more previously submitted RegistryObject
1239 objects using the LifeCycleManager service interface.

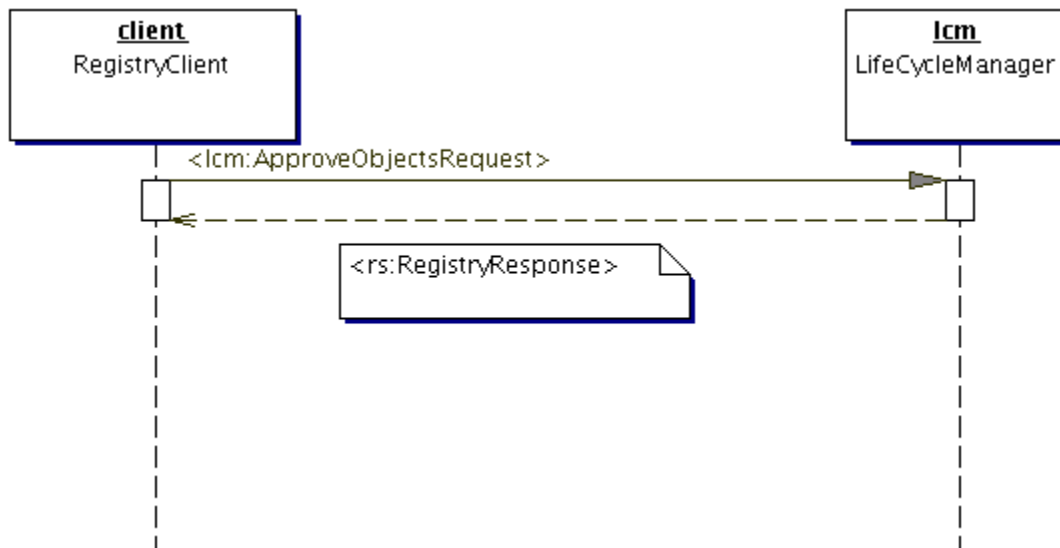


Figure 7: Approve Objects Protocol

1241 5.3.1 ApproveObjectsRequest

1242 The ApproveObjectsRequest is used by a client to approve one or more existing RegistryObject instances in
1243 the registry.

1244 5.3.1.1 Syntax:

```
1245 <element name="ApproveObjectsRequest">  
1246   <complexType>  
1247     <complexContent>  
1248       <extension base="rs:RegistryRequestType">  
1249         <sequence>  
1250           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />  
1251           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />  
1252         />  
1253       </sequence>  
1254     </extension>  
1255   </complexContent>  
1256 </complexType>  
1257 </element>
```

1258 5.3.1.2 Parameters:

- 1259
- 1260
- 1261 ▪ **AdhocQuery:** This parameter specifies a query. A registry MUST approve all objects that
1262 match the specified query in addition to any other objects identified by other parameters.
 - 1263 ▪ **ObjectRefList:** This parameter specifies a collection of references to existing
RegistryObject instances in the registry. A registry MUST approve all objects that are
referenced by this parameter in addition to any other objects identified by other parameters.

1264 **5.3.1.3 Returns:**

1265 This request returns a RegistryResponse. See section 2.1.4 for details.

1266 **5.3.1.4 Exceptions:**

1267 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be
1268 returned:

- 1269 *ObjectNotFoundException*: Indicates that the requestor requested an object within the
1270 request that was not found.

1271

1272 **5.3.2 Audit Trail**

1273 The registry MUST create a single AuditableEvent object with eventType *Approved* for all RegistryObject
1274 instance approved via an ApproveObjectsRequest.

1275 **5.4 The Deprecate Objects Protocol**

1276 The Deprecate Object protocol allows a client to deprecate one or more previously submitted RegistryObject
1277 instances using the LifeCycleManager service interface. Once a RegistryObject is deprecated, no new
1278 references (e.g. new Associations, Classifications and ExternalLinks) to that object can be submitted.
1279 However, existing references to a deprecated object continue to function normally.

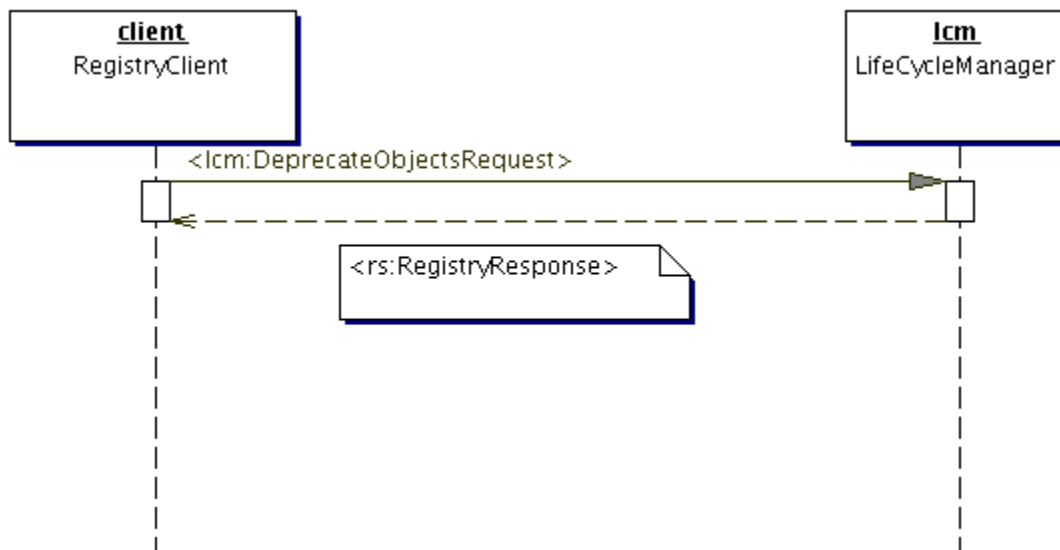


Figure 8: Deprecate Objects Protocol

1281 **5.4.1 DeprecateObjectsRequest**

1282 The DeprecateObjectsRequest is used by a client to deprecate one or more existing RegistryObject
1283 instances in the registry.

1284 **5.4.1.1 Syntax:**

```
1285 <element name="DeprecateObjectsRequest">  
1286   <complexType>  
1287     <complexContent>  
1288       <extension base="rs:RegistryRequestType">  
1289         <sequence>
```

```
1290         <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />
1291         <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1"
1292 />
1293     </sequence>
1294 </extension>
1295 </complexContent>
1296 </complexType>
1297 </element>
```

1298 **5.4.1.2 Parameters:**

- 1299 ▪ **AdhocQuery:** This parameter specifies a query. A registry **MUST** deprecate all objects that
1300 match the specified query in addition to any other objects identified by other parameters.
- 1301 ▪ **ObjectRefList:** This parameter specifies a collection of references to existing
1302 RegistryObject instances in the registry. A registry **MUST** deprecate all objects that are
1303 referenced by this parameter in addition to any other objects identified by other parameters.

1304 **5.4.1.3 Returns:**

1305 This request returns a RegistryResponse. See section 2.1.4 for details.

1306 **5.4.1.4 Exceptions:**

1307 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions **MAY** be
1308 returned:

- 1309 ▪ **UnresolvedReferenceException:** Indicates that the requestor referenced an object within
1310 the request that was not resolved during the processing of the request.

1311 **5.4.2 Audit Trail**

1312 The registry **MUST** create a single AuditableEvent object with eventType *Deprecated* for all RegistryObject
1313 deprecated via a DeprecateObjectsRequest.

1314 **5.5 The Undeprecate Objects Protocol**

1315 The Undeprecate Objects protocol of the LifecycleManager service interface allows a client to undo the
1316 deprecation of one or more previously deprecated RegistryObject instances. When a RegistryObject is
1317 undeprecated, it goes back to the Submitted status and new references (e.g. new Associations,
1318 Classifications and ExternalLinks) to that object can now again be submitted.

1319

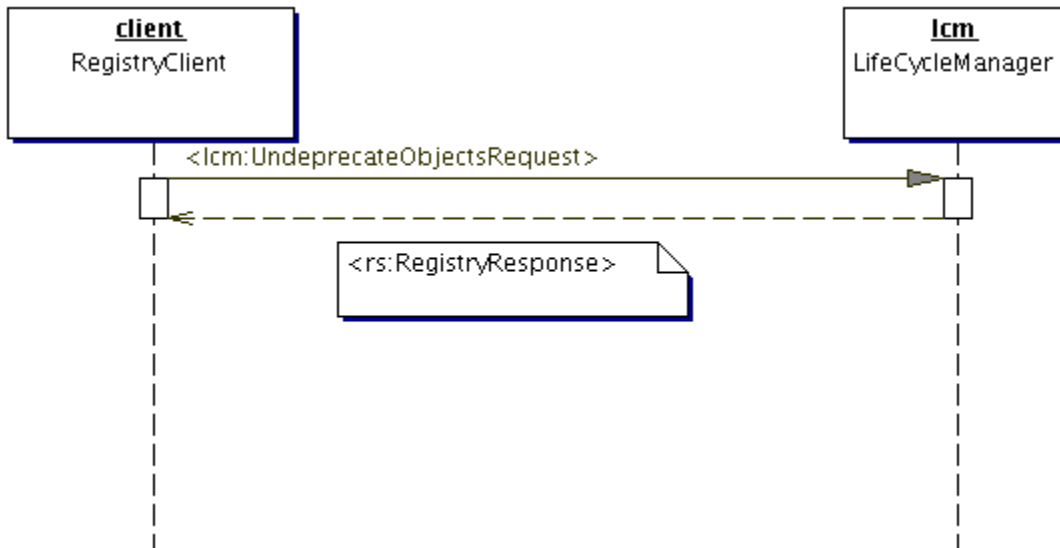


Figure 9: Undeprecate Objects Protocol

1320 5.5.1 UndeprecateObjectsRequest

1321 The UndeprecateObjectsRequest is used by a client to undeprecate one or more existing RegistryObject
1322 instances in the registry. The registry MUST silently ignore any attempts to undeprecate a RegistryObject
1323 that is not deprecated.

1324 5.5.1.1 Syntax:

```

1325 <element name="UndeprecateObjectsRequest">
1326   <complexType>
1327     <complexContent>
1328       <extension base="rs:RegistryRequestType">
1329         <sequence>
1330           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />
1331           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
1332         />
1333       </sequence>
1334     </extension>
1335   </complexContent>
1336 </complexType>
1337 </element>
1338 </element>
  
```

1339 5.5.1.2 Parameters:

- 1340 ▪ **AdhocQuery:** This parameter specifies a query. A registry MUST undeprecate all objects
1341 that match the specified query in addition to any other objects identified by other
1342 parameters.
- 1343 ▪ **ObjectRefList:** This parameter specifies a collection of references to existing
1344 RegistryObject instances in the registry. A registry MUST undeprecate all objects that are
1345 referenced by this parameter in addition to any other objects identified by other parameters.

1346 5.5.1.3 Returns:

1347 This request returns a RegistryResponse. See section 2.1.4 for details.

1348 **5.5.1.4 Exceptions:**

1349 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be
1350 returned:

- 1351 *UnresolvedReferenceException*: Indicates that the requestor referenced an object within
1352 the request that was not resolved during the processing of the request.

1353 **5.5.2 Audit Trail**

1354 The Registry Service MUST create a single AuditableEvent object with eventType *Undeprecated* for all
1355 RegistryObjects undeprecated via an UndeprecateObjectsRequest.

1356 **5.6 The Remove Objects Protocol**

1357 The Remove Objects protocol allows a client to remove one or more RegistryObject instances and/or
1358 repository items using the LifeCycleManager service interface.

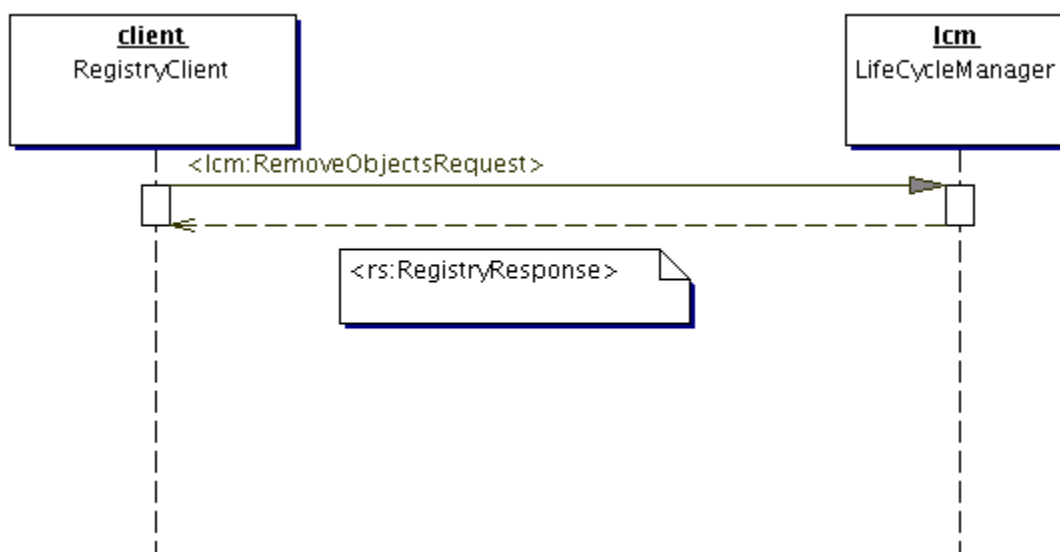


Figure 10: Remove Objects Protocol

1360 For details on the schema for the business documents shown in this process refer to .

1361 **5.6.1 RemoveObjectsRequest**

1362 The RemoveObjectsRequest is used by a client to remove one or more existing RegistryObject and/or
1363 repository items from the registry.

1364 **5.6.1.1 Syntax:**

```
1365 <element name="RemoveObjectsRequest">  
1366   <complexType>  
1367     <complexContent>  
1368       <extension base="rs:RegistryRequestType">  
1369         <sequence>  
1370           <element ref="rim:AdhocQuery" minOccurs="0" maxOccurs="1" />  
1371           <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />  
1372         />  
1373       </sequence>
```

```
1374     <attribute name="deletionScope"
1375 default="urn:oasis:names:tc:ebxml-regrep:DeletionScopeType:DeleteAll"
1376 type="rim:referenceURI" use="optional"/>
1377     </extension>
1378   </complexContent>
1379 </complexType>
1380 </element>
```

1381 5.6.1.2 Parameters:

- 1382
- 1383 ▪ **deletionScope:** This parameter indicates the scope of impact of the
1384 RemoveObjectsRequest. The value of the deletionScope attribute MUST be a reference to
1385 a ClassificationNode within the canonical DeletionScopeType ClassificationScheme as
1386 described in appendix A of [ebRIM]. A Registry MUST support the deletionScope types as
1387 defined by the canonical DeletionScopeType ClassificationScheme. The canonical
1388 DeletionScopeType ClassificationScheme may easily be extended by adding additional
ClassificationNodes to it.

1389 The following canonical ClassificationNodes are defined for the DeletionScopeType
1390 ClassificationScheme:

1391 **DeleteRepositoryItemOnly:** This deletionScope specifies that the registry MUST
1392 delete the RepositoryItem for the specified ExtrinsicObjects but MUST NOT delete
1393 the specified ExtrinsicObjects. This is useful in keeping references to the
1394 ExtrinsicObjects valid. A registry MUST set the status of the ExtrinsicObject
1395 instance to *Withdrawn* in this case.

1396 **DeleteAll:** This deletionScope specifies that the request MUST delete both the
1397 RegistryObject and the RepositoryItem (if any) for the specified objects. A
1398 RegistryObject can be removed using a RemoveObjectsRequest with
1399 deletionScope DeleteAll only if all references (e.g. Associations, Classifications,
1400 ExternalLinks) to that RegistryObject have been removed.

- 1401 ▪ **AdhocQuery:** This parameter specifies a query. A registry MUST remove all objects that
1402 match the specified query in addition to any other objects identified by other parameters.
- 1403 ▪ **ObjectRefList:** This parameter specifies a collection of references to existing
1404 RegistryObject instances in the registry. A registry MUST remove all objects that are
1405 referenced by this parameter in addition to any other objects identified by other parameters.

1406 5.6.1.3 Returns:

1407 This request returns a RegistryResponse. See section 2.1.4 for details.

1408 5.6.1.4 Exceptions:

1409 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions MAY be
1410 returned:

- 1411 ▪ **UnresolvedReferenceException:** Indicates that the requestor referenced an object within
1412 the request that was not resolved during the processing of the request.
- 1413 ▪ **ReferencesExistException:** Indicates that the requestor attempted to remove a
1414 RegistryObject while references to it still exist. Note that it is valid to remove a
1415 RegistryObject and all RegistryObjects that refer to it within the same request. In such
1416 cases the ReferencesExistException MUST not be thrown.

1417 5.7 Registry Managed Version Control

1418 This section describes the version control features of the ebXML Registry. This feature is based upon
1419 [DeltaV]. The ebXML Registry provides a simplified façade that provides a small subset of [DeltaV]
1420 functionality.

1421 **5.7.1 Version Controlled Resources**

1422 All repository items in an ebXML Registry are implicitly version-controlled resources as defined by section
1423 2.2.1 of [DeltaV]. No explicit action is required to make them a version-controlled resource.

1424 In addition RegistryObject instances are also implicitly version-controlled resources. However, a registry
1425 may limit version-controlled resources to a sub-set of RegistryObject classes based upon registry specific
1426 policies.

1427 Minimally, a registry implementing the version control feature SHOULD make the following types as version-
1428 controlled resources:

- 1429 ▪ ClassificationNode
- 1430 ▪ ClassificationScheme
- 1431 ▪ Organization
- 1432 ▪ ExtrinsicObject
- 1433 ▪ RegistryPackage
- 1434 ▪ Service

1435 The above list is chosen to exclude all composed types and include most of remaining RegistryObject types
1436 for which there are known use cases requiring versioning.

1437 **5.7.2 Versioning and Object Identification**

1438 Each version of a RegistryObject is a unique object and as such has its own unique value for its id attribute
1439 as defined by [ebRIM].

1440 **5.7.3 Logical ID**

1441 All versions of a RegistryObject are logically the same object and are referred to as the `logical`
1442 RegistryObject. A logical RegistryObject is a tree structure where nodes are specific versions of the
1443 RegistryObject.

1444 A specific version of a logical RegistryObject is referred to as a `RegistryObject instance`.

1445 A RegistryObject instance MUST have a *Logical ID (LID)* to identify its membership in a particular logical
1446 RegistryObject. Note that this is in contrast with the `id` attribute that MUST be unique for each version of
1447 the same logical RegistryObject. A client may refer to the logical RegistryObject in a version independent
1448 manner using its LID.

1449 A RegistryObject is assigned a LID using the `lid` attribute of the RegistryObject class. If the submitter
1450 assigns the `lid` attribute, she must guarantee that it is a globally unique URN. A registry MUST honor a valid
1451 submitter-supplied LID. If the submitter does not specify a LID then the registry MUST assign a LID and the
1452 value of the LID attribute MUST be identical to the value of the `id` attribute of the first (originally created)
1453 version of the logical RegistryObject.

1454 **5.7.4 Version Identification**

1455 An ebXML Registry supports independent versioning of both RegistryObject metadata as well as repository
1456 item content. It is therefore necessary to keep distinct version information for a RegistryObject instance and
1457 its repository item if it happens to be an ExtrinsicObject instance.

1458 **5.7.4.1 Version Identification for a RegistryObject**

1459 A RegistryObject MUST have a `versionInfo` attribute whose type is the VersionInfo class defined by ebRIM.
1460 The `versionInfo` attributes identifies the version information for that RegistryObject instance. A registry
1461 MUST not allow two versions of the same RegistryObject to have the same `versionInfo.versionName`
1462 attribute value.

1463 **5.7.4.2 Version Identification for a RepositoryItem**

1464 When a RegistryObject is an ExtrinsicObject with an associated repository item, the version identification for
 1465 the repository item is distinct from the version identification for the ExtrinsicObject.

1465 An ExtrinsicObject that has an associated repository item MUST have a contentVersionInfo attribute whose
 1466 type is the VersionInfo class defined by ebRIM. The contentVersionInfo attributes identifies the version
 1467 information for that repository item instance.

1466 An ExtrinsicObject that does not have an associated repository item MUST NOT have a contentVersionInfo
 1467 attribute defined.

1467 A registry MUST allow two versions of the same ExtrinsicObject to have the same
 1468 contentVersionInfo.versionName attribute value because multiple ExtrinsicObject versions MAY share the
 1469 same RepositoryItem version.

1468 **5.7.5 Versioning of ExtrinsicObject and Repository Items**

1469 An ExtrinsicObject and its associated repository item may be updated independently and therefore
 1470 versioned independently.

1470 A registry MUST maintain separate version trees for an ExtrinsicObject and its associated repository item as
 1471 described earlier.

1471 Table 6 shows all the combinations for versioning an ExtrinsicObject and its repository item. After eliminating
 1472 invalid or impossible combinations as well as those combinations where no action is needed, the only
 1473 combinations that require versioning are showed in gray background rows. Of these there are only two
 1474 unique cases (referred to as case A and B). Note that it is not possible to version a repository item without
 1475 versioning its ExtrinsicObject.
 1472

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

1473

1474 **5.7.5.1 ExtrinsicObject and Shared RepositoryItem**

1475 Because an ExtrinsicObject and its repository item are versioned independently (case B) it is possible for
1476 multiple versions of the ExtrinsicObject to share the same version of the repository item. In such cases the
1477 contentVersionInfo attributes MUST be the same across multiple version of the ExtrinsicObject.

1478 **5.7.6 Versioning and Composed Objects**

1479 When a registry creates a new version of a RegistryObject it MUST create copies of all composed¹ objects
1480 as new objects that are composed within the new version. This is because each version is a unique object
1481 and composed objects by definition are not shareable across multiple objects. Specifically, each new copy
1482 of a composed object MUST have a new id since it is a different object than the original composed object in
1483 the previous version.

1484 A registry MUST not version composed objects.

1485 **5.7.7 Versioning and References**

1486 An object reference from a RegistryObject references a specific version of the referenced RegistryObject.
1487 When a registry creates a new version of a referenced RegistryObject it MUST NOT move references from
1488 other objects from the previous version to the new version of the referenced object. Clients that wish to
1489 always reference the latest versions of an object MAY use the Event Notification feature to update
1490 references when new versions are created and thus always reference the latest version.

1491 A special case is when a SubmitObjectsRequest or an UpdateObjectRequest contains an object that is
1492 being versioned by the registry and the request contains other objects that reference the object being
1493 versioned. In such case, the registry MUST update all references within the submitted objects to the object
1494 being versioned such that those objects now reference the new version of the object being created by the
1495 request.

1496 **5.7.8 Versioning and Audit Trail**

1497 The canonical EventType ClassificationScheme used by the Audit Trail feature defines an Updated event
1498 type and then defines a Versioned event type as a child of the Updated event type ClassificationNode. The
1499 semantic are that a Versioned event type is specialization of the Updated event type.

1500 A registry MUST use the Updated event type in the AuditableEvent when it updates a RegistryObject without
1501 creating a new version.

1502 A registry MUST use the Versioned event type in the AuditableEvent when it creates a new version of a
1503 logical RegistryObject.

1504 A registry MUST NOT use the Created event type in the AuditableEvent when it creates a new version of a
1505 logical RegistryObject.

1506 **5.7.9 Inter-versions Association**

1507 Within any single branch within the version tree for an object any given version implicitly supersedes the
1508 version immediately prior to it. Sometimes it may be necessary to explicitly indicate which version
1509 supersedes another version for the same object. This is especially true when two versions are siblings
1510 branch roots of the version tree for the same object.

1511 A client MAY specify an Association between any two versions of an object within the objects version tree
1512 using the canonical associationType "Supersedes" to indicate that the sourceObject supersedes the target
1513 targetObject within the Association.

1514 A client MUST NOT specify an Association between two version of an object using the canonical
1515 associationType "Supersedes" if the sourceObject is an earlier version within the same branch in the version
1516 tree than the targetObject as this violates the implicit "Supersedes" association between the two version.

1517 Note that this section is functionally equivalent to the predecessor-set successor-set elements of the Version

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

1518 Properties as defined by [DeltaV].

1519 **5.7.10 Client Initiated Version Removal**

1520 An ebXML Registry MAY allow clients to remove specified versions of a RegistryObject. A client MAY delete
1521 older version of an object using the RemoveObjectsRequest by specifying the version by its unique id.

1522 Removing an ExtrinsicObject instance MUST remove its repository item if no other version references that
1523 repository item.

1524 **5.7.11 Registry Initiated Version Removal**

1525 The registry MAY prune older versions based upon registry specific administrative policies in order to
1526 manage storage resources.

1527 **5.7.12 Locking and Concurrent Modifications**

1528 This specification does not define a workspace feature with explicit checkin and checkout capabilities as
1529 defined by [DeltaV]. An ebXML Registry MAY support such features in an implementation specific manner.

1530 This specification does not prescribe a locking or branching model. An implementation may choose to
1531 support an optimistic (non-locking) model. Alternatively or in addition, an implementation may support a
1532 locking model that supports explicit checkout and checkin capability. A future technical note or specification
1533 may address some of these capabilities.

1534 **5.7.13 Version Creation**

1535 The registry manages creation of new version of a RegistryObject or a repository item automatically. A
1536 registry that supports versioning MUST implicitly create a new version for a repository item if the repository
1537 item is updated via a SubmitObjectsRequest or UpdateObjectsRequest. In such cases it MUST also create
1538 a new version of its ExtrinsicObject.

1539 If the client only wishes to update and version the ExtrinsicObject it may do so using an
1540 UpdateObjectsRequest without providing a repository item. In such cases the registry MUST assign the
1541 repository item version associated with the previous version of the ExtrinsicObject.

1542 **5.7.14 Versioning Override**

1543 A client MAY specify a *dontVersion* hint on a per RegistryObject basis when doing a submit or update of a
1544 RegistryObject. A registry SHOULD not create a new version for that RegistryObject when the dontVersion
1545 hint has value of "true". The dontVersion hint MAY be specified as a canonical Slot with the following name:

1546
1547 `urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersion`

1549 The value of the dontVersion Slot, if specified, MUST be either "true" or "false".

1550 A client MAY specify a *dontVersionContent* hint on a per ExtrinsicObject basis when doing a submit or
1551 update of an ExtrinsicObject with a repository item. A registry SHOULD not create a new version for that
1552 repository item when the dontVersionContent hint has value of "true". The dontVersionContent hint MAY be
1553 specified as a canonical Slot with the following name:

1554
1555 `urn:oasis:names:tc:ebxml-regrep:rim:RegistryObject:dontVersionContent`

1557 The value of the dontVersionContent Slot, if specified, MUST be either "true" or "false".

1558 A client MAY also specify the dontVersion and dontVersionContent Slots on the RegistryRequest using the
1559 `<rs:RequestSlotList>` element. A registry MUST treat these Slots when specified on the request as equivalent
1560 to being specified on every RegistryObject within the request. The value of these Slots as specified on the
1561 request take precedence over value of these Slots as specified on RegistryObjects within the request.

6 Query Management Protocols

1562

1563 This section defines the protocols supported by QueryManager service interface of the Registry. The Query
1564 Management protocols provide the functionality required by RegistryClients to query the registry and
1565 discover RegistryObjects and RepositoryItems.

1566 The XML schema for the Query Management protocols is described in [RR-QUERY-XSD].

6.1 Ad Hoc Query Protocol

1567

1568 The Ad hoc Query protocol of the QueryManager service interface allows a client to query the registry and
1569 retrieve RegistryObjects and/or RepositoryItems that match the specified query.

1570 A client submits an ad hoc query to the QueryManager by sending an AdhocQueryRequest. The
1571 AdhocQueryRequest contains a sub-element that specifies a query in one of the query syntaxes supported
1572 by the registry.

1573 The QueryManager sends an AdhocQueryResponse back to the client as response. The
1574 AdhocQueryResponse returns a collection of objects that match the query. The collection is potentially
1575 heterogeneous depending upon the query expression and request options.

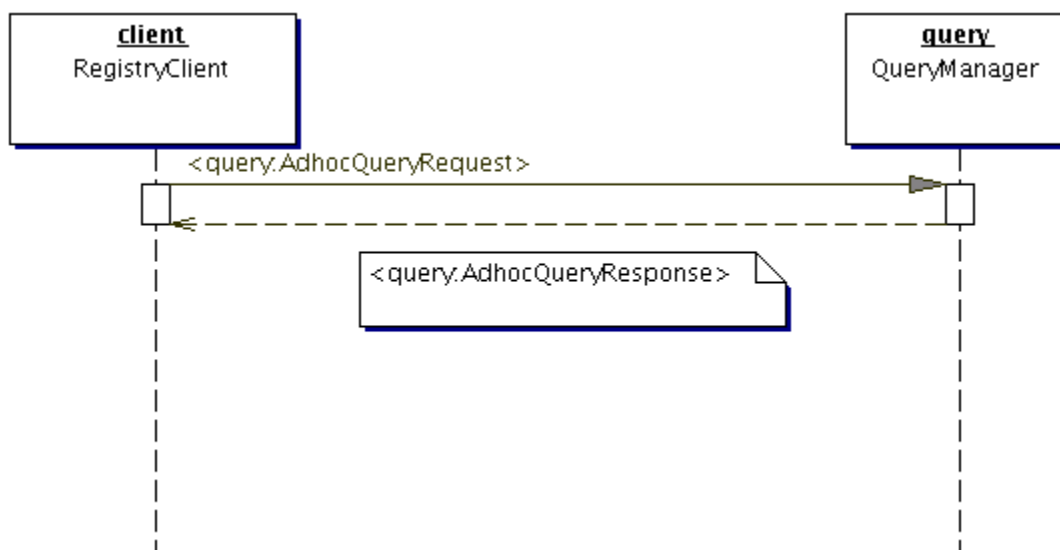


Figure 11: Ad Hoc Query Protocol

6.1.1 AdhocQueryRequest

1576

1577 The AdhocQueryRequest is used to submit a query to the registry.

6.1.1.1 Syntax:

1578

```
1579 <element name="AdhocQueryRequest">  
1580   <complexType>  
1581     <complexContent>  
1582       <extension base="rs:RegistryRequestType">  
1583         <sequence>  
1584           <element maxOccurs="1" minOccurs="1"  
1585             ref="tns:ResponseOption"/>  
1586           <element ref="rim:AdhocQuery" />  
1587         </sequence>  
1588         <attribute default="false" name="federated"  
1589           type="boolean" use="optional"/>  
1590       </extension>  
1591     </complexContent>  
1592   </complexType>  
</element>
```



```

1590     <attribute name="federation" type="anyURI" use="optional"/>
1591     <attribute default="0" name="startIndex" type="integer"/>
1592     <attribute default="-1" name="maxResults" type="integer"/>
1593     </extension>
1594   </complexContent>
1595 </complexType>
1596 </element>

```

1597 6.1.1.2 Parameters:

- 1598 ▪ **AdhocQuery:** This parameter specifies the actual query. It is described in detail in section
1599 6.1.3.
- 1600 ▪ **federated:** This optional parameter specifies that the registry must process this query as a
1601 federated query. By default its value is *false*. This value **MUST** be false when a registry
1602 routes a federated query to another registry in order to avoid an infinite loop in federated
1603 query processing.
- 1604 ▪ **federation:** This optional parameter specifies the id of the target Federation for a federated
1605 query in case the registry is a member of multiple federations. In the absence of this
1606 parameter a registry must route the federated query to all federations of which it is a
1607 member. This value **MUST** be unspecified when a registry routes a federated query to
1608 another registry in order to avoid an infinite loop in federated query processing.
- 1609 ▪ **maxResults:** This optional parameter specifies a limit on the maximum number of results
1610 the client wishes the query to return. If unspecified, the registry **SHOULD** return either all
1611 the results, or in case the result set size exceeds a registry specific limit, the registry
1612 **SHOULD** return a sub-set of results that are within the bounds of the registry specific limit.
1613 See section 6.2.1 for an illustrative example.
- 1614 ▪ **ResponseOption:** This required parameter allows the client to control the format and
1615 content of the AdhocQueryResponse generated by the registry in response to this request.
1616 See section 6.1.4 for details.
- 1617 ▪ **startIndex:** This optional integer value is used to indicate which result *must* be returned as
1618 the first result when iterating over a large result set. The default value is 0, which returns
1619 the result set starting with index 0 (first result). See section 6.2.1 for an illustrative example.

1620 6.1.1.3 Returns:

1621 This request returns an AdhocQueryResponse. See section 6.1.2 for details.

1622 6.1.1.4 Exceptions:

1623 In addition to the exceptions common to all requests defined in 2.1.1.4, the following exceptions **MAY** be
1624 returned:

- 1625 ▪ **InvalidQueryException:** signifies that the query syntax or semantics was invalid. Client must
1626 fix the query syntax or semantic error and re-submit the query.

1627 6.1.2 AdhocQueryResponse

1628 The AdhocQueryResponse is sent by the registry as a response to an AdhocQueryRequest.

1629 6.1.2.1 Syntax:

```

1630 <element name="AdhocQueryResponse">
1631   <complexType>
1632     <complexContent>
1633       <extension base="rs:RegistryResponseType">
1634         <sequence>
1635           <element ref="rim:RegistryObjectList" />

```

```

1636         </sequence>
1637         <attribute default="0" name="startIndex" type="integer"/>
1638         <attribute name="totalResultCount" type="integer"
1639 use="optional"/>
1640     </extension>
1641 </complexContent>
1642 </complexType>
1643 </element>

```

1644 6.1.2.2 Parameters:

- 1645 ▪ **RegistryObjectList:** This is the element that contains the RegistryObject instances that
1646 matched the specified query.
- 1647 ▪ **startIndex:** This optional integer value is used to indicate the index for the first result in the
1648 result set returned by the query, within the complete result set matching the query. By
1649 default, this value is 0. See section 6.2.1 for an illustrative example.
- 1650 ▪ **totalResultCount:** This optional parameter specifies the size of the complete result set
1651 matching the query within the registry. When this value is unspecified, the client should
1652 assume it is the size of the result set contained within the result. See section 6.2.1 for an
1653 illustrative example.

1654 6.1.3 AdhocQuery

1655 A client specifies a <rim:AdhocQuery> element within an AdhocQueryRequest to specify the actual query
1656 being submitted.

1657 6.1.3.1 Syntax:

```

1658 <complexType abstract="true" name="AdhocQueryType">
1659   <complexContent>
1660     <extension base="tns:RegistryObjectType">
1661       <sequence>
1662         <element ref="tns:QueryExpression"
1663           minOccurs="0" maxOccurs="1" />
1664       </sequence>
1665     </extension>
1666   </complexContent>
1667 </complexType>
1668 <element name="AdhocQuery" type="tns:AdhocQueryType"
1669   substitutionGroup="tns:RegistryObject" />

```

1670

1671 6.1.3.2 Parameters:

- 1672 ▪ **queryExpression:** This element contains the actual query expression. The schema for
1673 queryExpression is extensible and can support any query syntax supported by the registry.

1674 6.1.4 ReponseOption

1675 A client specifies a ResponseOption structure within an AdhocQueryRequest to indicate the format of the
1676 results within the corresponding AdhocQueryResponse.

1677

1678 6.1.4.1 Syntax:

```

1679 <complexType name="ResponseOptionType">
1680   <attribute default="RegistryObject" name="returnType">

```

```

1681     <simpleType>
1682         <restriction base="NCName">
1683             <enumeration value="ObjectRef"/>
1684             <enumeration value="RegistryObject"/>
1685             <enumeration value="LeafClass"/>
1686             <enumeration value="LeafClassWithRepositoryItem"/>
1687         </restriction>
1688     </simpleType>
1689 </attribute>
1690     <attribute default="false" name="returnComposedObjects"
1691 type="boolean"/>
1692 </complexType>
1693 <element name="ResponseOption" type="tns:ResponseOptionType"/>
1694

```

1695 **6.1.4.2 Parameters:**

- 1696 ▪ **returnComposedObjects:** This optional parameter specifies whether the RegistryObjects
1697 returned should include composed objects as defined by Figure 1 in [ebRIM]. The default is
1698 to return all composed objects.
- 1699 ▪ **returnType:** This optional enumeration parameter specifies the type of RegistryObject to
1700 return within the response. Values for returnType are as follows:
 - 1701 • **ObjectRef** - This option specifies that the AdhocQueryResponse MUST contain
1702 a collection of <rim:ObjectRef> elements. The purpose of this option is to return
1703 references to registry objects rather than the actual objects.
 - 1704 • **RegistryObject** - This option specifies that the AdhocQueryResponse MUST
1705 contain a collection of <rim:RegistryObject> elements.
 - 1706 • **LeafClass** - This option specifies that the AdhocQueryResponse MUST contain
1707 a collection of elements that correspond to leaf classes as defined in [RR-RIM-
1708 XSD].
 - 1709 • **LeafClassWithRepositoryItem** - This option is same as LeafClass option with
1710 the additional requirement that the response include the RepositoryItems, if any,
1711 for every <rim:ExtrinsicObject> element in the response.

1712 If “returnType” specified does not match a result returned by the query, then the registry
1713 *must* use the closest matching semantically valid returnType that matches the result.

1714 To illustrate, consider a case where OrganizationQuery is asked to return
1715 LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume
1716 LeafClass option instead.

1718 **6.2 Iterative Query Support**

1719 The AdhocQueryRequest and AdhocQueryResponse support the ability to iterate over a large result set
1720 matching a logical query by allowing multiple AdhocQueryRequest requests to be submitted such that each
1721 query requests a different subset of results within the result set. This feature enables the registry to handle
1722 queries that match a very large result set, in a scalable manner. The iterative query feature is accessed via
1723 the startIndex and maxResults parameters of the AdhocQueryRequest and the startIndex and
1724 totalResultCount parameters of the AdhocQueryResponse as described earlier.

1725 The iterative queries feature is not a true Cursor capability as found in databases. The registry is not
1726 required to maintain transactional consistency or state between iterations of a query. Thus it is possible for
1727 new objects to be added or existing objects to be removed from the complete result set in between
1728 iterations. As a consequence it is possible to have a result set element be skipped or duplicated between
1729 iterations.

1730 Note that while it is not required, an implementations MAY implement a transactionally consistent iterative
1731 query feature.

1732 **6.2.1 Query Iteration Example**

1733 Consider the case where there are 1007 Organizations in a registry. The user wishes to submit a query that
1734 matches all 1007 Organizations. The user wishes to do the query iteratively such that Organizations are
1735 retrieved in chunks of 100. The following table illustrates the parameters of the AdhocQueryRequest and
1736 those of the AdhocQueryResponses for each iterative query in this example.
1737

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

1738

1739 **6.3 Stored Query Support**

1740 The AdhocQuery protocol allow clients to submit queries that may be as general or as specific as the use
1741 case demands. As the queries get more specific they also get more complex. In these situations it is
1742 desirable to hide the complexity of the query from the client using parameterized queries stored in the
1743 registry. When using parameterized stored queries the client is only required to specify the identity of the
1744 query and the parameters for the query rather than the query expression itself.

1745 Parameterized stored queries are useful to Registry Administrators because they provide a system wide
1746 mechanism for the users of the registry to share a set of commonly used queries.

1747 Parameterized stored queries are useful to vertical standards because the standard can define domain
1748 specific parameterized queries and require that they be stored within the registry.

1749 An ebXML Registry MUST support parameterized stored queries as defined by this section.

1750 **6.3.1 Submitting a Stored Query**

1751 A stored query is submitted using the standard SubmitObjectsRequest protocol where the object submitted
1752 is an AdhocQueryType instance.

1753 **6.3.1.1 Declaring Query Parameters**

1754 When submitting a stored query, the submitter MAY declare zero or more parameters for that query. A
1755 parameter MUST be declared using a parameter name that begins with the '\$' character followed
1756 immediately by a letter and then followed by any combination of letters and numbers. The following BNF
1757 defines how a parameter name MUST be declared.

1758

1759 `QueryParameter := '$' [a-zA-Z] ([a-zA-Z] | [0-9])*`

1760

1761 A query parameter MAY be used as a placeholder for any part of the stored query.

1762 The following example illustrates how a parameterized stored query may be submitted:

1763

```

1764 <SubmitObjectsRequest>
1765   <rim:RegistryObjectList>
1766     <rim:AdhocQuery id="{QUERY_ID}">
1767       <rim:QueryExpression queryLanguage="{SQL_QUERY_LANG_ID}">
1768         SELECT * from $tableName ro, Name_ nm, Description d
1769         WHERE
1770         objectType = '$objectType'
1771         AND (nm.parent = ro.id AND UPPER ( nm.value ) LIKE UPPER
1772         ( '$name' ) )
1773         AND (d.parent = ro.id AND UPPER ( d.value ) LIKE UPPER
1774         ( '$description' ) )
1775         AND (ro.id IN ( SELECT classifiedObject FROM Classification WHERE
1776         classificationNode IN ( SELECT id
1777         FROM ClassificationNode WHERE path LIKE '$classificationPath1%'
1778         ) ) )
1779       </rim:QueryExpression>
1780     </rim:AdhocQuery>
1781   </rim:RegistryObjectList>
1782 </SubmitObjectsRequest>

```

Listing 1: Example of Stored Query Submission

1783
1784

1785 The above query takes parameters *\$objectType*, *\$name*, *\$description* and *\$classificationPath1* and find all
1786 objects for that match specified objectType, name, description and classification.

1787 6.3.1.2 Canonical Context Parameters

1788 A query MAY contain one or more context parameters as defined in this section. Context parameters are
1789 special query parameters whose value does not need to be supplied by the client. Instead the value for a
1790 context parameter is supplied by the registry based upon the context within which the client request is being
1791 processed.

1792 When processing a query, a registry MUST replace all context parameters present in the query with the
1793 context sensitive value for the parameter. A registry MUST ignore any context parameter values supplied by
1794 the client.

1795

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.

1796

1797 6.3.2 Invoking a Stored Query

1798 A stored query is invoked using the AdhocQueryRequest with the following constraints:

- 1799 • The <rim:AdhocQuery> element MUST not contain a <rim:queryExpression> element.
- 1800 • The <rim:AdhocQuery> element's id attribute value MUST match the id attribute value of the stored
1801 query.
- 1802 • The <rim:AdhocQuery> element MAY have a Slot for each non-context parameter defined for the stored
1803 query being invoked. These Slots provide the value for the query parameters.

1804 6.3.2.1 Specifying Query Invocation Parameters

1805 A stored query MAY be defined with zero or more parameters. A client may specify zero or more of the
1806 parameters defined for the stored query when submitting the AdhocQueryRequest for the stored query. It is
1807 important to note that the client MAY specify fewer parameters than those declared for the stored query. A

1808 registry MUST prune any predicates of the stored query that contain parameters that were not supplied by
1809 the client during invocation of the stored query.

1810 In essence, the client may narrow or widen the specificity of the search by supplying more or less
1811 parameters.

1812 A client specifies a query invocation parameter by using a Slot whose name matches the parameter name
1813 and whose value MUST be a single value that matches the specified value for the parameter.

1814 A registry MUST ignore any parameters specified by the client for a stored query that do not match the
1815 parameters defined by the stored query.

1816 The following listing shows an example of how the stored query shown earlier is invoked. It shows:

- 1817 • ~~The stored query being identified by the value of the canonical slot with name-~~
1818 "urn:oasis:names:tc:ebxml-regrep:rs:AdhocQueryRequest:queryId"~~The stored query being identified by~~
1819 the value of the id attribute of the <rim:AdhocQuery> element.
- 1820 • The value for the \$name parameter being supplied
- 1821 • The value of other parameters defined by the query not being supplied. This indicates that the client does
1822 not wish to use those parameters as search criterea:criteria.

1823

```
1824 <AdhocQueryRequest xmlns="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"  
1825 xmlns:lem="urn:oasis:names:tc:ebxml-regrep:xsd:lem:3.0"  
1826 xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"  
1827 xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"  
1828 xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0"  
1829 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
1830 xsi:schemaLocation="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0  
1831 http://oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd"  
1832 <rs:RequestSlotList>  
1833 <rim:Slot name="urn:oasis:names:tc:ebxml-  
1834 regrep:rs:AdhocQueryRequest:queryId">  
1835 <rim:ValueList>  
1836 <rim:Value>urn:freebxml:registry:query:BusinessQuery</rim:Value>  
1837 </rim:ValueList>  
1838 </rim:Slot>  
1839 <rim:Slot name="$name">  
1840 <rim:ValueList>  
1841 <rim:Value>%ebXML% </rim:Value>  
1842 </rim:ValueList>  
1843 </rim:Slot>  
1844 </rs:RequestSlotList>  
1845 <query:ResponseOption returnComposedObjects="true"  
1846 returnType="LeafClassWithRepositoryItem" />  
1847 <rim:AdhocQuery id="temporaryId">  
1848 <rim:QueryExpression queryLanguage="urn:oasis:names:tc:ebxml-  
1849 regrep:QueryLanguage:SQL-92">  
1850 <!-- No need for an actual query since it is fetched from registry  
1851 using the queryId -->  
1852 </rim:QueryExpression>  
1853 </rim:AdhocQuery>  
1854 </AdhocQueryRequest>  
1855 <AdhocQueryRequest>  
1856 <query:ResponseOption returnComposedObjects="true"  
1857 returnType="LeafClassWithRepositoryItem" />  
1858  
1859 <rim:AdhocQuery id="{STORED_QUERY_ID}">  
1860 <rim:Slot name="$name">  
1861 <rim:ValueList>  
1862 <rim:Value>%ebXML%</rim:Value>  
1863 </rim:ValueList>  
1864 </rim:Slot>  
1865 </rim:AdhocQuery>
```

*Listing 2: Example of Stored Query Invocation***1867 6.3.3 Response to Stored Query Invocation**

1868 A registry **MUST** send a standard AdhocQueryResponse when a client invokes a stored query using an
1869 AdhocQueryRequest.

1870 6.3.4 Access Control on a Stored Query

1871 A stored query is a RegistryObject. Like all RegistryObjects, access to the stored query is governed by the
1872 Access Control Policy defined the stored query. By default a stored query is assigned the default Access
1873 Control Policy that allows any client to read and invoke that query and only the owner of the query and the
1874 Registry Administrator role to update or delete the query. The owner of the query may define a custom
1875 Access Control Policy for the query that restricts the visibility of the query, and ability to invoke it, to specific
1876 users, roles or groups. Thus the owner of the query or the Registry Administrator may control *who* gets to
1877 invoke *which* stored queries.

1878 6.3.5 Canonical Query: Get Client's User Object

1879 A registry **MUST** support a canonical stored query with

1880 `id="urn:oasis:names:tc:ebxml-regrep:query:GetCallersUser".`

1881 This query **MUST** return the User object associated with the client invoking the stored query. The client
1882 **MUST** not provide any parameters for this query. The stored query **SHOULD** use the canonical context
1883 parameter `$currentUser`.

1884 The following is a non-normative example of a stored SQL query that **MAY** be used by a registry for this
1885 canonical stored query:

1886

```
1887 <rim:AdhocQuery id="urn:oasis:names:tc:ebxml-regrep:query:GetCallersUser">
1888   <rim:QueryExpression
1889     queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:SQL-92">
1890     SELECT u.* FROM User u WHERE u.id = $currentUser;
1891   </rim:QueryExpression>
1892 </rim:AdhocQuery>
```

1893 Note that a registry **MAY** use an equivalent stored filter query instead of a stored SQL query.

1894 6.4 SQL Query Syntax

1895 An ebXML Registry **MAY** support SQL as a supported query syntax within the `<rim:queryExpression>`
1896 element of AdhocQueryRequest. This section normatively defines the SQL syntax that an ebXML Registry
1897 **MAY** support. Note that the support for SQL syntax within a registry does not imply a requirement that the
1898 registry must use a relational database in its implementation.

1899 The registry SQL syntax is a proper subset of the "SELECT" statement of Intermediate level SQL as defined
1900 by ISO/IEC 9075:1992, Database Language SQL [SQL].

1901 The terms below enclosed in angle brackets are defined in [SQL] or in [SQL/PSM]. The SQL query syntax
1902 conforms to the `<query specification>` with the following additional restrictions:

- 1903 1. A **<derived column>** **MAY NOT** have an **<as clause>**.
- 1904 2. A **<table expression>** does not contain the optional **<group by clause>** and **<having clause>**
1905 clauses.
- 1906 3. A **<table reference>** can only consist of **<table name>** and **<correlation name>**.
- 1907 4. A **<table reference>** does not have the optional AS between **<table name>** and **<correlation name>**.
- 1908 5. Restricted use of sub-queries is allowed by the syntax as follows. The **<in predicate>** allows for the

1909 right hand side of the **<in predicate>** to be limited to a restricted **<query specification>** as defined
1910 above.

1911 As defined by [SQL], a registry MUST process table names and attribute names in a case insensitive
1912 manner.

1913 **6.4.1 Relational Schema for SQL Queries**

1914 The normative Relational Schema definition that is the target of registry SQL queries can be found at the
1915 following location on the web:

1916 <http://www.oasis-open.org/committees/regrep/documents/3.0/sql/database.sql>

1917 **6.4.2 SQL Query Results**

1918 The result of an SQL query resolves to a collection of objects within the registry. It never resolves to partial
1919 attributes. The objects related to the result set may be returned as an ObjectRef, RegistryObject or leaf
1920 class depending upon the returnType attribute of the responseOption parameter specified by the client on
1921 the AdHocQueryRequest. The entire result set is returned as an **<rim:RegistryObjectList>**.

1922 **6.5 Filter Query Syntax**

1923 This section normatively defines an XML syntax for querying an ebXML Registry called *Filter Query* syntax.
1924 An ebXML Registry MUST support the Filter Query syntax as a supported query syntax within the
1925 **<rim:queryExpression>** element of AdhocQueryRequest.

1926 The Filter Query syntax is defined in [RR-QUERY-XSD] and is derived from a mapping from [ebRIM] to XML
1927 Schema following certain mapping patterns.

1928 The Filter Query operational model views the network of RegistryObjects in the registry as a virtual XML
1929 document and a query traverses a specified part of the tree and prunes or filters objects from the virtual
1930 document using filter expressions and ultimately returns a collection of objects that are left after filtering out
1931 all objects that do not match the filters specified in the query.

1932 Unlike SQL query syntax, the filter query syntax does not support joins across classes. This constrains the
1933 expressive capabilities of the query and may also be somewhat less efficient in processing.

1934 **6.5.1 Filter Query Structure**

1935 The **<rim:queryExpression>** element of AdhocQueryRequest MUST contain a *Query* element derived from
1936 the **<query:RegistryObjectQueryType>** type.

1937 A Query element MAY contain a **<query:PrimaryFilter>** element and MAY contain additional Filter, Branch
1938 and Query elements within it as shown in the asbtract example below. The normative schema is defined by
1939 [RR-QUERY-XSD].

```
1940 <${QueryElement}>  
1941 <PrimaryFilter ... />  
1942 <${OtherFilterElement} ... />  
1943 <${BranchElement} ... />  
1944 <${QueryElement} ... />  
1945 </${QueryElement}>
```

1947

1948 The role of Query, Filter and Branch elements will be defined next.

1949 **6.5.2 Query Elements**

1950 A Query element is the top level element in the Filter Query syntax to query the registry. The [RR-QUERY-
1951 XSD] XML Schema defines a Query element for the RegistryObject class and all its descendant classes as
1952 defined by [ebRIM] using the following pattern:

- 1953 • For each class in model descendant from RegistryObject class define a complexType with name

- 1954 <class>QueryType. For example there is an OrganizationQueryType complexType defined for the
 1955 Organization class in [ebRIM].
- 1956 • The QueryType of a descendant of RegistryObject class MUST extend the QueryType for its super class.
 1957 For example the OrganizationQueryType extends the RegistryObjectQueryType.
 - 1958 • For RegistryObject class and each of its descendants define an element with name <class>Query and
 1959 with type <class>QueryType. For example the OrganizationQuery element is defined with type
 1960 OrganizationQueryType.

1961 The class associated with a Query element is referred to as the *Query domain class*.

1962 The following example shows the Query syntax where the Query domain class is the Organization class
 1963 defined by [ebRIM]:

1964

```

1965 <complexType name="OrganizationQueryType">
1966   <complexContent>
1967     <extension base="tns:RegistryObjectQueryType">
1968       ...Relevant Filters, Queries and Branches are defined here...
1969     </extension>
1970   </complexContent>
1971 </complexType>
1972 <element name="OrganizationQuery" type="tns:OrganizationQueryType"/>
```

1973

1974 A Query element MAY have Filter, Branch or nested Query Elements. These are described in subsequent
 1975 sections.

1976 6.5.3 Filter Elements

1977 A Query element MAY contain one or more Filter sub-elements. A Filter element is used to *filter* or select a
 1978 subset of instances of a specific [ebRIM] class. The class that a Filter filters is referred to as the *Filter*
 1979 *domain class*. A Filter element specifies a restricted predicate clause over the attributes of the Filter domain
 1980 class.

1981 [RR-QUERY-XSD] XML Schema defines zero or more Filter elements within a Query element definition
 1982 using the following pattern:

- 1983 • **PrimaryFilter.** A Filter element is defined within the RegistryObjectQueryType with name *PrimaryFilter*.
 1984 This Filter is used to filter the instances of the Query domain class based upon the value of its primitive
 1985 attributes. The cardinality of the Filter element is zero or one. The *PrimaryFilter* element is inherited by all
 1986 descendant QueryTypes of RegistryObjectQueryType.
- 1987 • **Additional Filters:** Additional Filters in a Query element used to filter the instances of the Query domain
 1988 class based upon whether the candidate domain class instance has a referenced object that satisfies the
 1989 additional filter.
 1990 Additional filter elements are defined for those attributes of the Query domain class that satisfy all of the
 1991 following criteria:
 - 1992 • The attribute's domain is not a primitive type (e.g. string, float, dateTime, int etc.).
 - 1993 • The attribute's domain class is not RegistryObject or its descendant.
 - 1994 • The attribute's domain class does not have any reference attributes (use Branch or sub-Query if
 1995 attribute's domain class has reference attributes).

1996 The attribute for which the Filter is defined is referred to as the Filter domain attribute. The domain
 1997 class of the Filter domain attribute is the Filter domain class for such Filters. This type of Filter is used
 1998 to filter the instances of the Query domain class based upon the attribute values within the Filter
 1999 domain class.

 - 2000 • The name of the Filter element is <Filter Domain Attribute Name>Filter.
 - 2001 • The type of the Filter element is the FilterType complex type that is described in 6.5.3.1.
 - 2002 • The cardinality of the Filter element matches the cardinality of the Filter domain attribute in the Query
 2003 domain class.

2004

2005 The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to define
2006 Filters for the OrganizationQueryType for the Organization class defined by [ebRIM].

2007

```

2008 <complexType name="OrganizationQueryType">
2009   <complexContent>
2010     <extension base="tns:RegistryObjectQueryType">
2011       <sequence>
2012         <element maxOccurs="unbounded" minOccurs="0"
2013           name="AddressFilter" type="tns:FilterType"/>
2014         <element maxOccurs="unbounded" minOccurs="0"
2015           name="TelephoneNumberFilter" type="tns:FilterType"/>
2016         <element maxOccurs="unbounded" minOccurs="0"
2017           name="EmailAdresseFilter" type="tns:FilterType"/>
2018         ...Branches and sub-Queries go here...
2019       </sequence>
2020     </extension>
2021   </complexContent>
2022 </complexType>

```

2023

2024 The following UML class diagram describing the Filter class structure as defined in [RR-QUERY-XSD] XML
2025 Schema. Note that the classes whose name ends in "Type" map to complexTypes and other Filter classes
2026 map to elements in the [RR-QUERY-XSD] XML Schema.

2027

2028

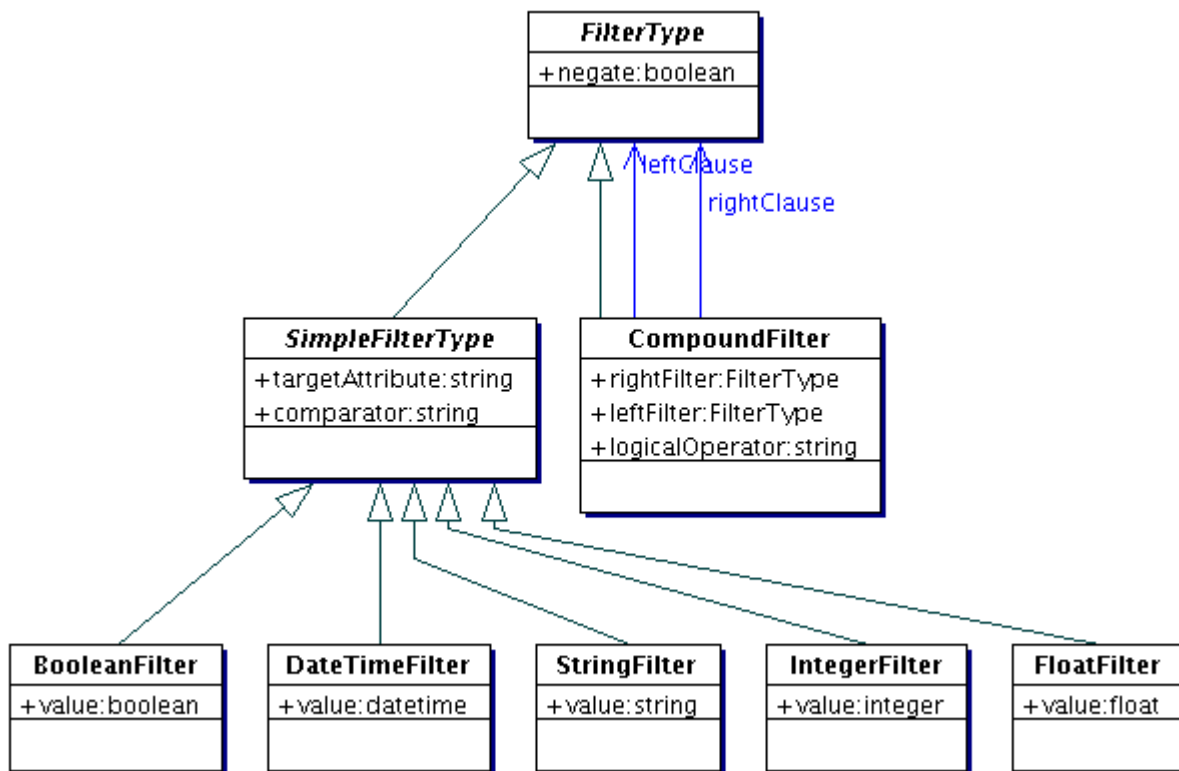


Figure 12: Filter Type Hierarchy

2029

2030

2031 6.5.3.1 FilterType

2032 The FilterType is an abstract complexType that is the root type in the inheritance hierarchy for all Filter

2033 types.

2034 **6.5.3.1.1 Parameters:**

- 2035 ▪ **negate:** This parameter specifies that the boolean value that the Filter evaluates to MUST
2036 be negated to complete the evaluation of the filter. It is functionally equivalent to the NOT
2037 operator in SQL syntax.

2038 **6.5.3.2 SimpleFilterType**

2039 The SimpleFilter is the abstract base type for several concrete Filter types defined for primitive type such as
2040 boolean, float, integer and string.

2041 **6.5.3.2.1 Parameters:**

- 2042 ▪ **domainAttribute:** This parameter specifies the attribute name of a primitive attribute within
2043 the Filter domain class. A registry MUST return an InvalidQueryException if this
2044 parameter's value does not match the name of primitive attribute within the Filter domain
2045 class. A registry MUST perform the attribute name match in a case insensitive manner.
- 2046 ▪ **comparator:** This parameter specifies the comparison operator for comparing the value of
2047 the attribute with the value supplied by the filter. The following comparators are defined:
 - 2048 • LE: abbreviation for LessThanOrEqual
 - 2049 • LT: abbreviation for LessThan
 - 2050 • GE: abbreviation for GreaterThanOrEqual
 - 2051 • GT: abbreviation for GreaterThan
 - 2052 • EQ: abbreviation for Equal
 - 2053 • NE: abbreviation for NotEqual
 - 2054 • Like: Same as LIKE operator in SQL-92. MUST only be used in StringFilter.
 - 2055 • NotLike: Same as NOT LIKE operator in SQL-92. MUST only be used in
2056 StringFilter.

2058 **6.5.3.3 BooleanFilter**

2059 The BooleanFilter MUST only be used for matching primitive attributes whose domain is of type boolean.

2060 **6.5.3.3.1 Parameters:**

- 2061 ▪ **value:** This parameter specifies the value that MUST be compared with the attribute value
2062 being tested by the Filter. It MUST be a boolean value.

2063 The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the
2064 ClassificationScheme class defined by [ebRIM]:

```
2065 <BooleanFilter  
2066     domainAttribute="isInternal" comparator="EQ" value="true"/>
```

2067

2068 **6.5.3.4 FloatFilter**

2069 The FloatFilter MUST only be used for matching primitive attributes whose domain is of type float.

2070 **6.5.3.4.1 Parameters:**

- 2071 ▪ **value:** This parameter specifies the value that MUST be compared with the attribute value

2072 being tested by the Filter. It MUST be a float value.

2073 The following example shows the use of a FloatFilter to match fictitious *amount* float attribute since [ebRIM]
2074 currently has no float attributes defined:

```
2075 <FloatFilter  
2076     domainAttribute="amount" comparator="GT" value="9.99"/>
```

2077

2078 6.5.3.5 IntegerFilter

2079 The IntegerFilter MUST only be used for matching primitive attributes whose domain is of type integer.

2080 6.5.3.5.1 Parameters:

- 2081 ▪ **value:** This parameter specifies the value that MUST be compared with the attribute value
2082 being tested by the Filter. It MUST be an integer value.

2083 The following example shows the use of a BooleanFilter to match a fictitious *count* integer attribute since
2084 [ebRIM] currently has no integer attributes defined:

```
2085 <IntegerFilter  
2086     domainAttribute="amount" comparator="LT" value="100"/>
```

2087

2088 6.5.3.6 DateTimeFilter

2089 The DateTimeFilter MUST only be used for matching primitive attributes whose domain is of type datetime.

2090 6.5.3.6.1 Parameters:

- 2091 ▪ **value:** This parameter specifies the value that MUST be compared with the attribute value
2092 being tested by the Filter. It MUST be a datetime value.

2093 The following example shows the use of a DateTimeFilter to match a the *timestamp* attribute of the
2094 Auditable class defined by [ebRIM] where the timestamp value is greater than (later than) the specified
2095 datetime value:

```
2096 <DateTimeFilter  
2097     domainAttribute="timestamp"  
2098     comparator="GT" value="1997-07-16T19:20+01:00"/>
```

2099

2100 6.5.3.7 StringFilter

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

2102 6.5.3.7.1 Parameters:

- 2103 ▪ **value:** This parameter specifies the value that MUST be compared with the attribute value
2104 being tested by the Filter. It MUST be a string value.

2105 The following example shows the use of a StringFilter to match a the *firstName* attribute of the Person class
2106 defined by [ebRIM] where the firstName value matches the pattern specified by the value:

```
2107 <StringFilter  
2108     domainAttribute="firstName"  
2109     comparator="Like" value="Farid%"/>
```

2110

2111 6.5.3.8 CompoundFilter

2112 The CompoundFilter MAY be used to specify a boolean conjunction (AND) or disjunction (OR) between two
2113 Filters. It allows a query to express a combination of predicate clauses within a Filter Query.

2114 6.5.3.8.1 Parameters:

- 2115 ▪ **LeftFilter:** This parameter specifies the first of two Filters for the CompoundFilter.
- 2116 ▪ **RightFilter:** This parameter specifies the second of two Filters for the CompoundFilter.
- 2117 ▪ **logicalOperator:** This parameter specifies the logical operator. The value of this
2118 parameter MUST be "AND" or "OR"

2119 The following example shows the use of a BooleanFilter to match the *isInternal* attribute of the
2120 ClassificationScheme class defined by [ebRIM]:

```
2121 <CompoundFilter logicalOperator="AND">  
2122   <LeftFilter domainAttribute="targetObject" comparator="EQ"  
2123     value="{REGISTRY_OBJECT_ID}" type="StringFilter"/>  
2124   <RightFilter domainAttribute="associationType" comparator="EQ"  
2125     value="{HAS_MEMBER_ASSOC_TYPE_NODE_ID}" type="StringFilter"/>  
2126 </CompoundFilter>
```

2127 6.5.4 Nested Query Elements

2128 A Query element MAY contain one or more nested Query sub-elements. The purpose of the nested Query
2129 element is to allow traversal of the branches within the network of relationships defined by the information
2130 model and prune or filter those branches that do not meet the predicates specified in the corresponding
2131 Branch element.

2132 The [RR-QUERY-XSD] XML Schema defines zero or more nested Query elements within a Query element
2133 definition using the following pattern:

- 2134 • A nested Query element is defined for each attribute of the Query domain class that satisfy all of the
2135 following criteria:
 - 2136 • The attribute's domain class is a descendant type of the RegistryObjectType.
 - 2137 • The attribute's domain class contains reference attributes that link the domain class to some third
2138 class via the reference.
- 2139 The attribute for which the nested Query is defined is referred to as the Nested Query domain
2140 attribute. The domain class of the nested Query domain attribute is the Query domain class for the
2141 nested Query element.
- 2142 • The name of the nested Query element is <Nested Query Domain Attribute Name>Query.
- 2143 • The type of the nested Query element matches the QueryType for the domain class for the Query
2144 domain attribute.
- 2145 • The cardinality of the nested Query element matches the cardinality of the nested Query domain attribute
2146 in the Query domain class.

2147 The following example shows the how [RR-QUERY-XSD] XML Schema uses the above pattern to define
2148 nested Query elements for the OrganizationQueryType for the Organization class defined by [ebRIM].

```
2149  
2150 <complexType name="OrganizationQueryType">  
2151   <complexContent>  
2152     <extension base="tns:RegistryObjectQueryType">  
2153       <sequence>  
2154         ...Filters and Branches go here ...  
2155         <element maxOccurs="1" minOccurs="0"  
2156           name="ParentQuery" type="tns:OrganizationQueryType"/>  
2157         <element maxOccurs="unbounded" minOccurs="0"  
2158           name="ChildOrganizationQuery" type="tns:OrganizationQueryType"/>  
2159         <element maxOccurs="1" minOccurs="0"  
2160           name="PrimaryContactQuery" type="tns:PersonQueryType"/>  
2161       </sequence>  
2162     </extension>  
2163   </complexContent>  
2164 </complexType>
```

2161
2162
2163
2164

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

2165 6.5.5 Branch Elements

2166 A Query element MAY contain one or more Branch sub-elements. A Branch element is similar to the nested
2167 Query element as it too can have sub-elements that are Filter, Branch and subQuery elements. However, it
2168 is different from Query elements because its type is not a descendant type of RegistryObjectQueryType.
2169 The purpose of the branch element is to allow traversal of the branches within the network of relationships
2170 defined by the information model and prune or filter those branches that do not meet the predicates
2171 specified in the corresponding Branch element.

2172 The [RR-QUERY-XSD] XML Schema defines zero or more Branch elements within a Query element
2173 definition using the following pattern:

2174 • A Branch element is defined for each attribute of the Query domain class that satisfies all of the following
2175 criteria:

- 2176 • The attribute's domain is not a primitive type (e.g. String, float, dateTime, int etc.).
- 2177 • The attribute's domain class contains reference attributes that link the domain class to some third
2178 class via the reference.

2179 The attribute for which the Branch is defined is referred to as the Branch domain attribute. The
2180 domain class of the Branch domain attribute is the Branch domain class for the Branch element.

- 2181 • The name of the Branch element is <Branch Domain Attribute Name>Branch.
- 2182 • The cardinality of the Branch element matches the cardinality of the Branch domain attribute in the
2183 Query domain class.

2184 The following example shows how the [RR-QUERY-XSD] XML Schema uses the above pattern to define
2185 Branches for the RegistryObjectQueryType for the RegistryObject class defined by [ebRIM].

2186

```
2187 <complexType name="RegistryObjectQueryType">  
2188 <complexContent>  
2189 <extension base="tns:FilterQueryType">  
2190 <sequence>  
2191 <element maxOccurs="unbounded" minOccurs="0"  
2192 name="SlotBranch" type="tns:SlotBranchType"/>  
2193 <element maxOccurs="1" minOccurs="0" name="NameBranch"  
2194 type="tns:InternationalStringBranchType"/>  
2195 <element maxOccurs="1" minOccurs="0" name="DescriptionBranch"  
2196 type="tns:InternationalStringBranchType"/>  
2197 ... Relevant Filters, queries go here...  
2198 </sequence>  
2199 </extension>  
2200 </complexContent>  
2201 </complexType>
```

2202

2203 6.6 Query Examples

2204 This section provides examples in both SQL and Filter Query syntax for some common query use cases.
2205 Each example gives the SQL syntax for the query followed by blank line followed by the equivalent Filter
2206 Query syntax for it.

2207 6.6.1 Name and Description Queries

2208 The following queries matches all RegistryObject instances whose name contains the word 'Acme' and
2209 whose description contains the word "bicycle".

2210

```

2211 SELECT ro.* from RegistryObject ro, Name nm, Description d WHERE
2212 nm.value LIKE '%Acme%' AND
2213 d.value LIKE '%bicycle%' AND
2214 (ro.id = nm.parent AND ro.id = d.parent);
2215
2216 <RegistryObjectQuery>
2217 <NameBranch>
2218 <LocalizedStringFilter comparator="Like" domainAttribute="value"
2219 value="%Acme%" xsi:type="StringFilterType"/>
2220 </NameBranch>
2221 <DescriptionBranch>
2222 <LocalizedStringFilter comparator="Like" domainAttribute="value"
2223 value="%bicycle%" xsi:type="StringFilterType"/>
2224 </DescriptionBranch>
2225 </RegistryObjectQuery>
2226

```

2227

2228 6.6.2 Classification Queries

2229 This section describes various classification related queries.

2230 6.6.2.1 Retrieving ClassificationSchemes

2231 The following query retrieves the collection of all ClassificationSchemes. Note that the above query may
 2232 also specify additional Filters, Querys and Branches as search criteria if desired.

2233

```

2234 SELECT scheme.* FROM ClassificationScheme scheme;
2235
2236 <ClassificationSchemeQuery/>

```

2237

2238 6.6.2.2 Retrieving Children of Specified ClassificationNode

2239 The following query retrieves the children of a ClassificationNode given the "id" attribute of the parent
 2240 ClassificationNode:

2241

```

2242 SELECT cn.* FROM ClassificationNode cn WHERE parent = ${PARENT_ID};
2243
2244 <ClassificationNodeQuery>
2245 <PrimaryFilter comparator="Like" domainAttribute="parent"
2246 value="${PARENT_ID}" xsi:type="StringFilterType"/>
2247 </ClassificationNodeQuery>

```

2248

2249 6.6.2.3 Retrieving Objects Classified By a ClassificationNode

2250 The following query retrieves the collection of ExtrinsicObjects that are classified by the Automotive Industry
 2251 and the Japan Geography. Note that the query does not match ExtrinsicObjects classified by descendant
 2252 ClassificationNodes of the Automotive Industry and the Japan Geography. That would require a slightly
 2253 more complex query.

2254

```

2255 SELECT eo.* FROM ExtrinsicObject eo WHERE
2256 id IN (SELECT classifiedObject FROM Classification
2257 WHERE
2258 classificationNode IN (SELECT id FROM ClassificationNode
2259 WHERE path = '/${GEOGRAPHY_SCHEME_ID}/Asia/Japan'))
2260 AND

```

```

2261 id IN (SELECT classifiedObject FROM Classification
2262 WHERE
2263     classificationNode IN (SELECT id FROM ClassificationNode
2264     WHERE path = '\${INDUSTRY_SCHEME_ID}/Automotive'))
2265
2266 <ExtrinsicObjectQuery>
2267   <ClassificationQuery>
2268     <ClassificationNodeQuery>
2269       <PrimaryFilter comparator="EQ" domainAttribute="path"
2270       value="/${GEOGRAPHY_SCHEME_ID}/Asia/Japan"
2271       xsi:type="StringFilterType"/>
2272     </ClassificationNodeQuery>
2273   </ClassificationQuery>
2274   <ClassificationQuery>
2275     <ClassificationNodeQuery>
2276       <PrimaryFilter comparator="EQ" domainAttribute="path"
2277       value="/${INDUSTRY_SCHEME_ID}/Automotive"
2278       xsi:type="StringFilterType"/>
2279     </ClassificationNodeQuery>
2280   </ClassificationQuery>
2281 </ExtrinsicObjectQuery>

```

2282

2283 6.6.2.4 Retrieving Classifications that Classify an Object

2284 The following query retrieves the collection of Classifications that classify an object with id matching \${ID}:

2285

```

2286 SELECT c.* FROM Classification c
2287 WHERE c.classifiedObject = ${ID};
2288
2289 <ClassificationQuery>
2290   <PrimaryFilter comparator="EQ" domainAttribute="classifiedObject"
2291   value="${ID}" xsi:type="StringFilterType"/>
2292 </ClassificationQuery>

```

2293

2294 6.6.3 Association Queries

2295 This section describes various Association related queries.

2296 6.6.3.1 Retrieving All Associations With Specified Object As Source

2297 The following query retrieves the collection of Associations that have the object with id matching
 2298 \${SOURCE_ID} as their source:

2299

```

2300 SELECT a.* FROM Association a WHERE sourceObject = ${SOURCE_ID}
2301
2302 <AssociationQuery>
2303   <PrimaryFilter comparator="EQ" domainAttribute="sourceObject"
2304   value="${SOURCE_ID}" xsi:type="StringFilterType"/>
2305 </AssociationQuery>

```

2306

2307 6.6.3.2 Retrieving All Associations With Specified Object As Target

2308 The following query retrieves the collection of Associations that have the object with id matching
 2309 \${TARGET_ID} as their target:

2310


```

2311 SELECT a.* FROM Association a WHERE targetObject = ${TARGET_ID}
2312
2313 <AssociationQuery>
2314   <PrimaryFilter comparator="EQ" domainAttribute="targetObject"
2315     value="${TARGET_ID}" xsi:type="StringFilterType"/>
2316 </AssociationQuery>

```

2317

2318 6.6.3.3 Retrieving Associated Objects Based On Association Type

2319

2320 Select Associations whose associationType attribute value matches the value specified by the
 2321 \${ASSOC_TYPE_ID}. The \${ASSOC_TYPE_ID} value MUST reference a ClassificationNode that is a
 2322 descendant of the canonical AssociationType ClassificationScheme.

2323

```

2324 SELECT a.* FROM Association a WHERE
2325     associationType = ${ASSOC_TYPE_ID}
2326
2327 <AssociationQuery>
2328   <PrimaryFilter comparator="EQ" domainAttribute="associationType"
2329     value="${ASSOC_TYPE_ID}" xsi:type="StringFilterType"/>
2330 </AssociationQuery>

```

2331

2332

2333 6.6.3.4 Complex Association Query

2334 The various forms of Association queries may be combined into complex predicates. The following query
 2335 selects Associations that match specified specific sourceObject, targetObject and associationType:

2336

```

2337 SELECT a.* FROM Association a WHERE
2338     sourceObject = ${SOURCE_ID} AND
2339     targetObject = ${TARGET_ID} AND
2340     associationType = ${ASSOC_TYPE_ID};
2341
2342 <AssociationQuery>
2343   <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2344     <LeftFilter comparator="EQ" domainAttribute="sourceObject"
2345       xsi:type="StringFilterType" value="${SOURCE_ID}"/>
2346     <RightFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2347       <LeftFilter comparator="EQ" domainAttribute="targetObject"
2348         xsi:type="StringFilterType" value="${TARGET_ID}"/>
2349       <RightFilter comparator="EQ" domainAttribute="associationType"
2350         xsi:type="StringFilterType" value="${ASSOC_TYPE_ID}"/>
2351     </RightFilter>
2352   </PrimaryFilter>
2353 </AssociationQuery>

```

2354

2355 6.6.4 Package Queries

2356 The following query retrieves all Packages that have as member the RegistryObject specified by
 2357 \${REGISTRY_OBJECT_ID}:

2358

```

2359 SELECT p.* FROM Package p, Association a WHERE
2360     a.sourceObject = p.id AND
2361     a.targetObject = ${REGISTRY_OBJECT_ID} AND

```

```

2362         a.associationType = ${HAS_MEMBER_ASSOC_TYPE_NODE_ID};
2363
2364 <RegistryPackageQuery>
2365   <SourceAssociationQuery>
2366     <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2367       <LeftFilter comparator="EQ" domainAttribute="targetObject"
2368         value="${REGISTRY_OBJECT_ID}"
2369         xsi:type="StringFilterType"/>
2370       <RightFilter comparator="EQ" domainAttribute="associationType"
2371         value="${HAS_MEMBER_ASSOC_TYPE_NODE_ID}"
2372         xsi:type="StringFilterType"/>
2373     </PrimaryFilter>
2374   </SourceAssociationQuery>
2375 </RegistryPackageQuery>

```

2376

2377 Note that the `${HAS_MEMBER_ASSOC_TYPE_NODE_ID}` is a placeholder for the value of the id attribute
2378 of the canonical HasMember AssociationType ClassificationNode.

2379 6.6.5 ExternalLink Queries

2380 The following query retrieves all ExternalLinks that serve as ExternalLink for the RegistryObject specified by
2381 `${REGISTRY_OBJECT_ID}`:

2382

```

2383 SELECT el.* From ExternalLink el, Association a WHERE
2384     a.sourceObject = el.id AND
2385     a.targetObject = ${REGISTRY_OBJECT_ID} AND
2386     a.associationType = ${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};
2387
2388 <ExternalLinkQuery>
2389   <SourceAssociationQuery>
2390     <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2391       <LeftFilter comparator="EQ" domainAttribute="targetObject"
2392         value="${REGISTRY_OBJECT_ID}"
2393         xsi:type="StringFilterType"/>
2394       <RightFilter comparator="EQ" domainAttribute="associationType"
2395         value="${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
2396         xsi:type="StringFilterType"/>
2397     </PrimaryFilter>
2398   </SourceAssociationQuery>
2399 </ExternalLinkQuery>

```

2400

2401 Note that the `${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}` is a placeholder for the value of the id
2402 attribute of the canonical ExternallyLinks AssociationType ClassificationNode.

2403 The following query retrieves all ExtrinsicObjects that are linked to an ExternalLink specified by
2404 `${EXTERNAL_LINK_ID}`:

2405

```

2406 SELECT eo.* From ExtrinsicObject eo, Association a WHERE
2407     a.sourceObject = ${EXTERNAL_LINK_ID} AND
2408     a.targetObject = eo.id AND
2409     a.associationType = ${EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID};
2410
2411 <ExtrinsicObjectQuery>
2412   <TargetAssociationQuery>
2413     <PrimaryFilter logicalOperator="AND" xsi:type="CompoundFilterType">
2414       <LeftFilter comparator="EQ" domainAttribute="sourceObject"
2415         value="${EXTERNAL_LINK_ID}"
2416         xsi:type="StringFilterType"/>
2417       <RightFilter comparator="EQ" domainAttribute="associationType"

```

```
2418         value="{EXTERNALLY_LINKS_ASSOC_TYPE_NODE_ID}"
2419         xsi:type="StringFilterType"/>
2420     </PrimaryFilter>
2421 </TargetAssociationQuery>
2422 </ExtrinsicObjectQuery>
```

2423

2424 **6.6.6 Audit Trail Queries**

2425 The following query retrieves all the AuditableEvents for the RegistryObject specified by
2426 \${REGISTRY_OBJECT_ID}:

2427

```
2428 SELECT ae.* FROM AuditableEvent ae, AffectedObject ao WHERE
2429         ao.eventId = ae.id AND
2430         ao.id = ${REGISTRY_OBJECT_ID}
2431
2432 <AuditableEventQuery>
2433     <AffectedObjectQuery>
2434         <PrimaryFilter comparator="EQ" domainAttribute="id"
2435             value="{REGISTRY_OBJECT_ID}" xsi:type="StringFilterType"/>
2436     </AffectedObjectQuery>
2437 </AuditableEventQuery>
```

2438

2439 7 Event Notification Protocols

2440 This chapter defines the Event Notification feature of the OASIS ebXML Registry.

2441 Event Notification feature allows OASIS ebXML Registries to notify its users and / or other registries about
2442 events of interest. It allows users to stay informed about registry events without being forced to periodically
2443 poll the registry. It also allows a registry to propagate internal changes to other registries whose content
2444 might be affected by those changes.

2445 ebXML registries support content-based Notification where interested parties express their interest in form of
2446 a query. This is different from subject-based (sometimes referred to as topic-based) notification, where
2447 information is categorized by subjects and interested parties express their interests in those predefined
2448 subjects.

2449 7.1 Use Cases

2450 The following use cases illustrate different ways in which ebXML registries notify users or other registries.

2451 7.1.1 CPP Has Changed

2452 A user wishes to know when the CPP [ebCPP] of a partner is updated or superseded by another CPP. When
2453 that happens he may wish to create a CPA [ebCPP] based upon the new CPP.

2454 7.1.2 New Service is Offered

2455 A user wishes to know when a new plumbing service is offered in her town and be notified every 10 days.
2456 When that happens, she might try to learn more about that service and compare it with her current plumbing
2457 service provider's offering.

2458 7.1.3 Monitor Download of Content

2459 User wishes to know whenever his CPP [ebCPP] is downloaded in order to evaluate on an ongoing basis
2460 the success of his recent advertising campaign. He might also want to analyze who the interested parties
2461 are.

2462 7.1.4 Monitor Price Changes

2463 User wishes to know when the price of a product that she is interested in buying drops below a certain
2464 amount. If she buys it she would also like to be notified when the product has been shipped to her.

2465 7.1.5 Keep Replicas Consistent With Source Object

2466 In order to improve performance and availability of accessing some registry objects, a local registry MAY
2467 make replicas of certain objects that are hosted by another registry. The registry would like to be notified
2468 when the source object for a replica is updated so that it can synchronize the replica with the latest state of
2469 the source object.

2470 7.2 Registry Events

2471 Activities within a registry result in meaningful events. Typically, registry events are generated when a
2472 registry processes client requests. In addition, certain registry events may be caused by administrative
2473 actions performed by a registry operator. [ebRIM] defines the AuditableEvent class, instances of which
2474 represent registry events. When such an event occurs, an AuditableEvent instance is generated by the
2475 registry.

2476 7.3 Subscribing to Events

2477 A user MAY create a subscription with a registry if he or she wishes to receive notification for a specific type
2478 of event. A user creates a subscription by submitting a Subscription instance to a registry using the

2479 SubmitObjectsRequest. If a Subscription is submitted to a registry that does not support event notification
2480 then the registry MUST return an UnsupportedCapabilityException.

2481 The listing below shows a sample Subscription using a pre-defined SQL query as its selector that will result
2482 in an email notification to the user whenever a Service is created that is classified as a "Plumbing" service
2483 and located in "A Little Town."
2484

2485 The SQL query within the selector in plain English says the following:

2486 *Find all Services that are Created AND classified by ClassificationNode*
2487 *where ClassificationNode's Path ends with string "Plumbing", AND classified by ClassificationNode where*
2488 *ClassificationNode's Code contains string "A Little Town."*

2489

```
2490 <rim:Subscription id="{SUBSCRIPTION_ID}" selector="{QUERY_ID}">  
2491   <!--  
2492     The selector is a reference to a query object that has the  
2493     following query defined  
2494     SELECT * FROM Service s, AuditableEvent e, AffectedObject ao,  
2495     Classification c1, Classification c2  
2496     ClassificationNode cn1, ClassificationNode cn2 WHERE  
2497     e.eventType = 'Created' AND ao.id = s.id AND ao.parent=e.id AND  
2498     c1.classifiedObject = s.id AND c1.classificationNode = cn1.id AND  
2499     cn1.path LIKE '%Plumbing' AND  
2500     c2.classifiedObject = s.id AND c2.classificationNode = cn2.id AND  
2501     cn2.path LIKE '%A Little Town%'  
2502   -->  
2503   <!-- Next endPoint is an email address -->  
2504   <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-  
2505   regrep:NotificationOptionType:Objects"  
2506   endPoint="mailto:farrukh.najmi@sun.com"/>  
2507   <!-- Next endPoint is a service via reference to its ServiceBinding  
2508   object -->  
2509   <rim:NotifyAction notificationOption="urn:oasis:names:tc:ebxml-  
2510   regrep:NotificationOptionType:ObjectRefs"  
2511   endPoint="urn:freebxml:registry:demoDB:serviceBinding:EpidemicAlertListen  
2512   erServiceBinding"/>  
2513 </rim:Subscription>  
2514
```

2515 **7.3.1 Event Selection**

2516 In order to only be notified of specific events of interest, the user MUST specify a reference to a stored
2517 AdHocQuery object via the selector attribute within the Subscription instance. The query determines
2518 whether an event qualifies for that Subscription or not. For details on query syntax see chapter 6.

2519 **7.3.2 Notification Action**

2520 When creating a Subscription, a user MAY also specify Actions within the subscription that specify what the
2521 registry must do when an event matching the Subscription (subscription event) transpires.

2522 A user MAY omit specifying an Action within a Subscription if he does not wish to be notified by the registry.
2523 A user MAY periodically poll the registry and pull the pending Notifications.

2524 [ebRIM] defines two standard ways that a NotifyAction may be used:

- 2525 • Email NotifyAction that allows delivery of event notifications via email to a human user or to an
2526 email end point for a software component or agent.
- 2527 • Service NotifyAction that allows delivery of event notifications via a programmatic interface by
2528 invoking a specified listener web service.

2529 If the registry supports event notification, at some time after the successful processing of each request, it
2530 MUST check all registered and active Subscriptions and see if any Subscriptions match the event. If a

2531 match is found then the registry performs the Notification Actions required for the Subscription. A registry
2532 MAY periodically perform such checks and corresponding notification actions in a batch mode based upon
2533 registry specific policies.

2534 **7.3.3 Subscription Authorization**

2535 A registry operator or content owner MAY use custom Access Control Policies to decide which users are
2536 authorized to create a subscription and to what events. A Registry MUST return an AuthorizationException in
2537 the event that an unauthorized user submits a Subscription to a registry. It is up to registry implementations
2538 whether to honour the existing subscription if an access control policy governing subscriptions becomes
2539 more restrictive after subscription have already been created based on the older policy.

2540 **7.3.4 Subscription Quotas**

2541 A registry MAY use registry specific policies to decide an upper limit on the number of Subscriptions a user
2542 is allowed to create. A Registry MUST return a QuotaExceededException in the event that an authorized
2543 user submits more Subscriptions than allowed by their registry specific quota.

2544 **7.3.5 Subscription Expiration**

2545 Each subscription defines a startTime and an endTime attribute which determines the period within which a
2546 Subscription is active. Outside the bounds of the active period, a Subscription MAY exist in an expired state
2547 within the registry. A registry MAY remove an expired Subscription at any time. In such cases the identity of
2548 a RegistryOperator user MUST be used for the request in order to have sufficient authorization to remove a
2549 user's Subscription.

2550 A Registry MUST NOT consider expired Subscriptions when delivering notifications for an event to its
2551 Subscriptions. An expired Subscription MAY be renewed by submitting a new Subscription.

2552 **7.3.6 Subscription Rejection**

2553 A Registry MAY reject a Subscription if it is too costly to support. For instance a Subscription that wishes to
2554 be notified of any change in any object may be too costly for most registries. A Registry MUST return a
2555 SubscriptionTooCostlyException in the event that an Authorized User submits a Subscription that is too
2556 costly for the registry to process.

2557 **7.4 Unsubscribing from Events**

2558 A user MAY terminate a Subscription with a registry if he or she no longer wishes to be notified of events
2559 related to that Subscription. A user terminates a Subscription by deleting the corresponding Subscription
2560 object using the RemoveObjectsRequest to the registry.

2561 Removal of a Subscription object follows the same rules as removal of any other object.

2562 **7.5 Notification of Events**

2563 A registry performs the *Actions* for a Subscription in order to actually deliver the events information to the
2564 subscriber. However, regardless of the specific delivery Action, the registry MUST communicate the
2565 Subscription events. The Subscription events are delivered within a Notification instance as described by
2566 [ebRIM]. In case of Service NotifyAction, the Notification is delivered to a handler service conformant to the
2567 RegistryClient interface. In case of an Email NotifyAction the notification is delivered an email address.

2568 The listing below shows a sample Notification matching the subscription example in section 7.3:

2569

```
2570 <rim:Notification subscription="{SUBSCRIPTION_ID}">  
2571   <rim:RegistryObjectList>  
2572     <rim:Service id="f3373a7b-4958-4e55-8820-d03a191fb76a">  
2573       <rim:Name>  
2574         <rim:LocalizedString value="A Little Town Plumbing"/>  
2575       </rim:Name>
```

```
2576     <rim:Classification id="a3373a7b-4958-4e55-8820-d03a191fb76a"  
2577     classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>  
2578     <rim:Classification id="b3373a7b-4958-4e55-8820-d03a191fb76a"  
2579     classifiedObject="f3373a7b-4958-4e55-8820-d03a191fb76a"/>  
2580     </rim:Service>  
2581     </rim:RegistryObjectList>  
2582 </rim:Notification>
```

2583

2584 A Notification MAY contain actual RegistryObjects or ObjectRefs to RegistryObjects within the
2585 <rim:RegistryObjectList>. A client MAY specify the whether they wish to receive RegistryObjects or
2586 ObjectRefs to RegistryObjects using the notificationOption attribute of the Action within the Subscription.
2587 The registry MAY override this notificationOption based upon registry specific operational policies.

2588 **7.6 Retrieval of Events**

2589 The registry provides asynchronous PUSH style delivery of Notifications via notify Actions as described
2590 earlier. However, a client MAY also use a PULL style to retrieve any pending events for their Subscriptions.
2591 Pulling of events is done using the AdHocQuery protocol and querying the Notification class. A registry
2592 SHOULD buffer undelivered notifications for some period to allow clients to PULL those notifications. The
2593 period that a registry SHOULD buffer undelivered notifications MAY be defined using registry specific
2594 policies.

2595 **7.7 Pruning of Events**

2596 A registry MAY periodically prune AuditableEvents in order to manage its resources. It is up to the registry
2597 when such pruning occurs. It is up to the registry to determine when undelivered events are purged. A
2598 registry SHOULD perform such pruning by removing the older information in its Audit Trail content. However,
2599 it MUST not remove the original Create Event at the beginning of the audit trail since the Create Event
2600 establishes the owner of the RegistryObject.

8 Content Management Services

2601

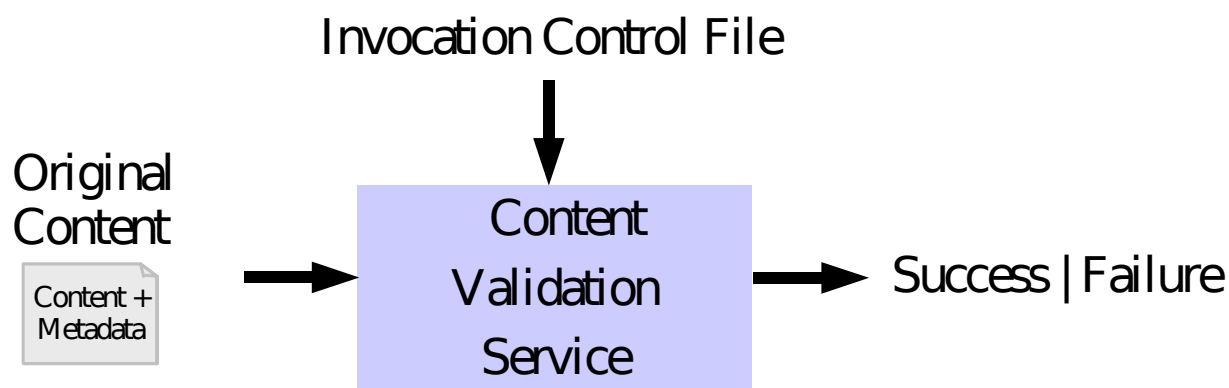
2602 This chapter describes the Content Management services of the ebXML Registry. Examples of Content
2603 Management Services include, but are not limited to, content validation and content cataloging. Content
2604 Management Services result in improved quality and integrity of registry content and metadata as well as
2605 improved ability for clients to discover that content and metadata.

2606 The Content Management Services facility of the registry is based upon a pluggable architecture that allows
2607 clients to publish and discover new Content Management Services as Service objects that conform to a
2608 normative web service interface specified in this chapter. Clients MAY configure a Content Management
2609 Service that is specialized for managing a specific type of content.

8.1 Content Validation

2610

2611 The Content Validation feature provides the ability to enforce domain specific validation rules upon
2612 submitted content and metadata in a content specific manner.



2613

2614

Figure 13: Content Validation Service

2615 A registry uses one or more Content Validation Services to automatically validate the RegistryObjects and
2616 repository items when they are submitted to the registry. A registry MUST reject a submission request in its
2617 entirety if it contains invalid data. In such cases a ValidationException MUST be returned to the client.

2618 Content Validation feature improves the quality of data in the registry.

8.1.1 Content Validation: Use Cases

2619

2620 The following use cases illustrate the Content Validation feature:

8.1.1.1 Validation of HL7 Conformance Profiles

2621

2622 The Healthcare Standards organization HL7 uses content validation to enforce consistency rules and
2623 semantic checks whenever an HL7 member submits an HL7 Conformance Profile. HL7 is also planning to
2624 use the feature to improve the quality of other types of HL7 artifacts.

8.1.1.2 Validation of Business Processes

2625

2626 Content validation may be used to enforce consistency rules and semantic checks whenever a Business
2627 Process is submitted to the registry. This feature may be used by organizations such as UN/CEFACT, OAGi,
2628 and RosettaNet.

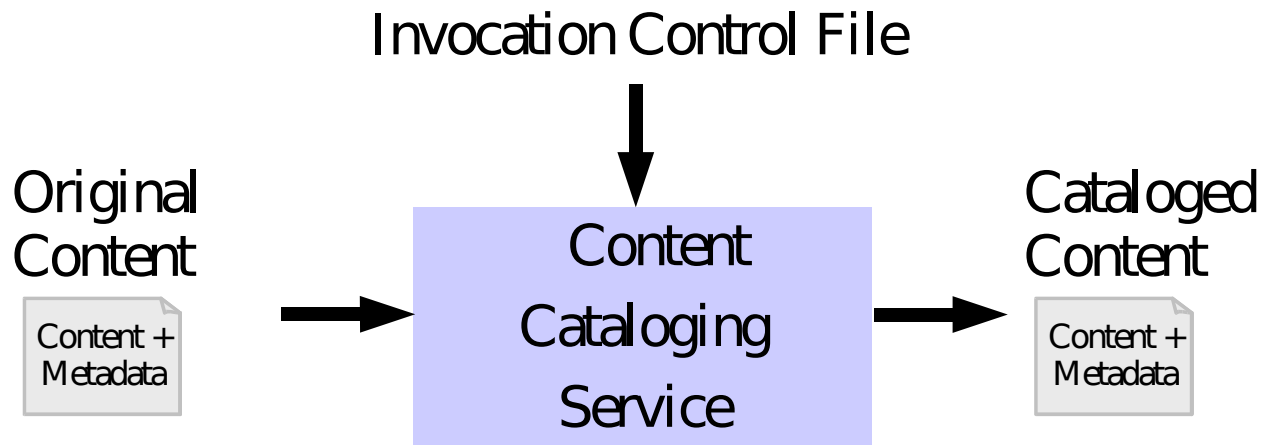
8.1.1.3 Validation of UBL Business Documents

2629

2630 Content validation may be used by the UBL technical committee to enforce consistency rules and semantic
2631 checks whenever a UBL business document is submitted to the registry.

2632 8.2 Content Cataloging

2633 The Content Cataloging feature provides the ability to selectively convert submitted RegistryObject and
2634 repository items into metadata defined by [ebRIM], in a content specific manner.



2635
2636

Figure 14: Content Cataloging Service

2637 A registry uses one or more Content Cataloging Services to automatically catalog RegistryObjects and
2638 repository items. Cataloging creates and/or updates RegistryObject metadata such as ExtrinsicObject or
2639 Classification instances. The cataloged metadata enables clients to discover the repository item based upon
2640 content from the repository item, using standard query capabilities of the registry. This is referred to as
2641 *Content-based Discovery*.

2642 The main benefit of the Content Cataloging feature is to enable Content-based Discovery.

2643 8.2.1 Content-based Discovery: Use Cases

2644 There are many scenarios where content-based discovery is necessary.

2645 8.2.1.1 Find All CPPs Where Role is “Buyer”

2646 A company that sells a product using the RosettaNet PIP3A4 Purchase Order process wants to find CPPs
2647 for other companies where the Role element of the CPP is that of “Buyer”.

2648 8.2.1.2 Find All XML Schema’s That Use Specified Namespace

2649 A client may wish to discover all XML Schema documents in the registry that use an XML namespace
2650 containing the word “oasis”.

2651 8.2.1.3 Find All WSDL Descriptions with a SOAP Binding

2652 An ebXML registry client is attempting to discover all repository items that are WSDL descriptions that have
2653 a SOAP binding defined. Note that SOAP binding related information is content within the WSDL document
2654 and not metadata.

2655 8.3 Abstract Content Management Service

2656 This section describes in abstract terms how the registry supports pluggable, user-defined Content
2657 Management Services. A Content Management Service is invoked in response to content being submitted to
2658 the registry via the standard Submit/UpdateObjectsRequest method. The Service invocation is on a per
2659 request basis where one request may result in many invocations, one for each RegistryObject for which a
2660 Content Management Service is configured within the registry.

2661 The registry may perform such invocation in one of two ways.

2662

2663 • **Inline Invocation Model:** Content Management Service may be invoked inline with the processing
2664 of the Submit/UpdateObjectsRequest and prior to committing the content. This is referred to as
2665 Inline Invocation Model.

2666 • **Decoupled Invocation Model:** Content Management Service may be invoked decoupled from the
2667 processing of the Submit/UpdateObjectsRequest and some time after committing the content. This
2668 is referred to as Decoupled Invocation Model.
2669

2670 8.3.1 Inline Invocation Model

2671 In an inline invocation model a registry MUST invoke a Content Management Service inline with
2672 Submit/UpdateObjectsRequest processing and prior to committing the Submit/UpdateObjectsRequest. All
2673 metadata and content from the original Submit/UpdateObjectsRequest request or from the Content
2674 Management Service invocation MUST be committed as an atomic transaction.

2675 Figure 15 shows an abstract Content Management Service and how it is used by an ebXML Registry using
2676 an inline invocation model. The steps are as follows:
2677

- 2678 1. A client submits a Content Management Service S1 to an ebXML Registry. The client
2679 typically belongs to an organization responsible for defining a specific type of content. For
2680 example the client may belong to RosettaNet.org and submit a Content Validation Service
2681 for validating RosettaNet PIPs. The client uses the standard Submit/UpdateObjectsRequest
2682 interface to submit the Service. This is a one-time step to configure this Content
2683 Management Service in the registry.
- 2684 2. Once the Content Management Service has been submitted, a potentially different client
2685 may submit content to the registry that is of the same object type for which the Content
2686 Management Service has been submitted. The client uses the standard
2687 Submit/UpdateObjectsRequest interface to submit the content.
- 2688 3. The registry determines there is a Content Management Service S1 configured for the
2689 object type for the content submitted. It invokes S1 using a
2690 ContentManagementServiceRequest and passes it the content.
- 2691 4. The Content Management Service S1 processes the content and sends back a
2692 ContentManagementServiceResponse.
- 2693 5. The registry then commits the content to the registry if there are no errors encountered.
- 2694 6. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest
2695 in step 2.
2696

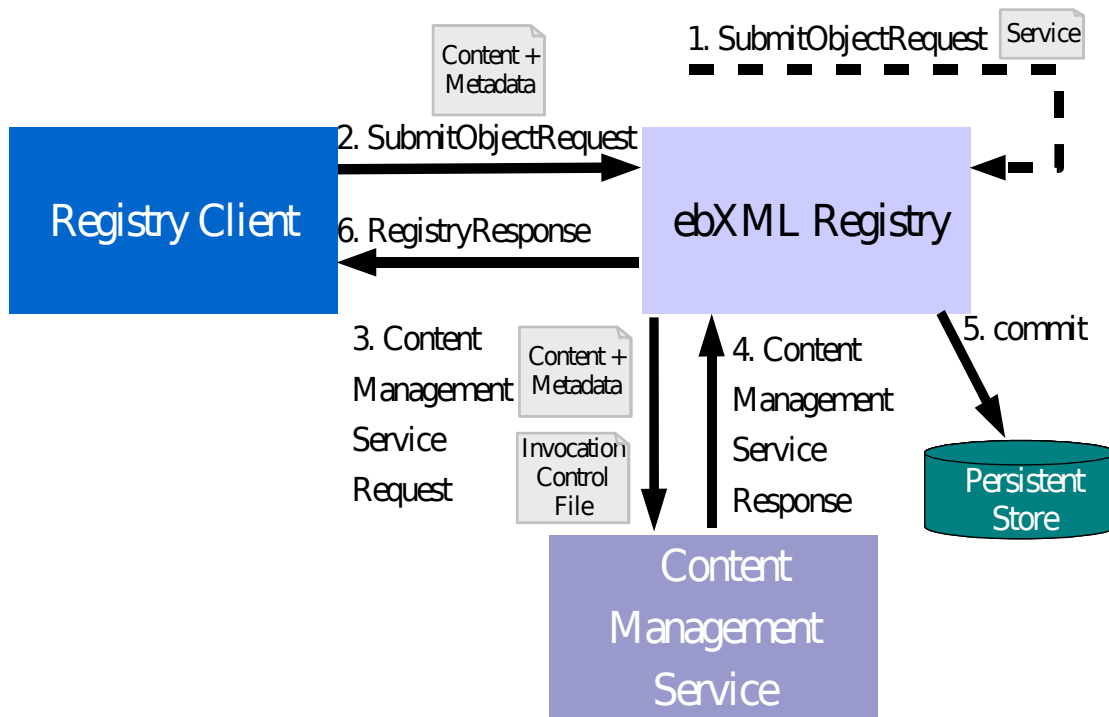


Figure 15: Content Management Service: Inline Invocation Model

2697
2698

2699 8.3.2 Decoupled Invocation Model

2700 In a decoupled invocation model a registry MUST invoke a Content Management Service independent of or
 2701 decoupled from the Submit/UpdateObjectsRequest processing. Any errors encountered during Content
 2702 Management Service invocation MUST NOT have any impact on the original Submit/UpdateObjectsRequest
 2703 processing.

2704 All metadata and content from the original Submit/UpdateObjectsRequest request MUST be committed as
 2705 an atomic transaction that is decoupled from the metadata and content that may be generated by the
 2706 Content Management Service invocation.

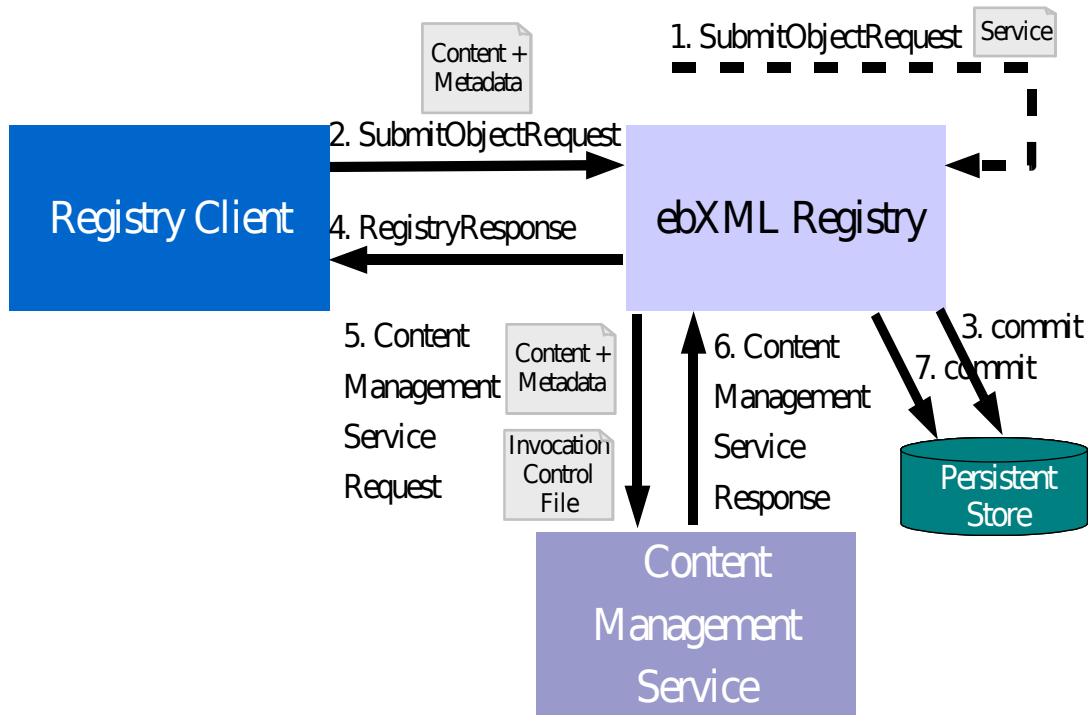
2707 Figure 16 shows an abstract Content Management Service and how it is used by an ebXML Registry using
 2708 a decoupled invocation model. The steps are as follows:

2709

- 2710 1. Same as in inline invocation model (Content Management Service is submitted).
- 2711 2. Same as in inline invocation model (client submits content using
2712 Submit/UpdateObjectsRequest).
- 2713 3. The registry processes the Submit/UpdateObjectsRequest and commits it to persistent
2714 store.
- 2715 4. The registry returns a RegistryResponse to the client for the Submit/UpdateObjectsRequest
2716 in step 2.
- 2717 5. The registry determines there is a Content Management Service S1 configured for the
2718 object type for the content submitted. It invokes S1 using a
2719 ContentManagementServiceRequest and passes it the content.
- 2720 6. The Content Management Service S1 processes the content and sends back a
2721 ContentManagementServiceResponse.

2722
2723
2724
2725
2726
2727

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



2728
2729

Figure 16: Content Management Service: Decoupled Invocation Model

2730

8.4 Content Management Service Protocol

2731 This section describe the abstract Content Management Service protocol that is the base- protocol for other
2732 concrete protocols such as Validate Content protocol and Catalog Content protocol. The concrete protocols
2733 will be defined later in this document.

2734

8.4.1 ContentManagementServiceRequestType

2735 The ContentManagementServiceRequestType MUST be the abstract base type for all requests sent from a
2736 registry to a Content Management Service.

2737

8.4.1.1 Syntax:

2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749

```
<complexType name="ContentManagementServiceRequestType">
  <complexContent>
    <extension base="rs:RegistryRequestType">
      <sequence>
        <element name="OriginalContent"
type="rim:RegistryObjectListType"/>
        <element name="InvocationControlFile"
type="rim:ExtrinsicObjectType" maxOccurs="unbounded" minOccurs="0"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

2750

2751 **8.4.1.2 Parameters:**

2752 The following parameters are parameters that are either newly defined for this type or are inherited and
2753 have additional semantics beyond those defined in the base type description.

- 2754 ▪ *InvocationControlFile*: This parameter specifies the ExtrinsicObject for a repository item
2755 that the caller wishes to specify as the Invocation Control File. This specification does not
2756 specify the format of this file. There MUST be a corresponding repository item as an
2757 attachment to this request. The corresponding repository item SHOULD follow the same
2758 rules as attachments in Submit/UpdateObjectsRequest.
 - 2759 ▪ *OriginalContent*: This parameter specifies the RegistryObjects that will be processed by
2760 the content management service. In case of ExtrinsicObject instances within the
2761 OriginalContent there MAY be repository items present as attachments to the
2762 ContentManagementServiceRequest. This specification does not specify the format of such
2763 repository items. The repository items SHOULD follow the same rules as attachments in
2764 Submit/UpdateObjectsRequest.
- 2765

2766 **8.4.1.3 Returns:**

2767 This request returns a ContentManagementServiceResponse. See section 8.4.2 for details.

2768 **8.4.1.4 Exceptions:**

2769 In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

- 2770 ▪ *MissingRepositoryItemException*: signifies that the caller did not provide a repository item
2771 as an attachment to this request when the Service requires it.
 - 2772 ▪ *InvocationControlFileException*: signifies that the InvocationControlFile(s) provided by the
2773 caller do not match the InvocationControlFile(s) expected by the Service.
 - 2774 ▪ *UnsupportedContentException*: signifies that this Service does not support the content
2775 provided by the caller.
- 2776

2777 **8.4.2 ContentManagementServiceResponseType**

2778 The ContentManagementServiceResponseType is sent by a Content Management Service as a response to
2779 a ContentManagementServiceRequestType. The ContentManagementServiceResponseType is the abstract
2780 base type for all responses sent to a registry from a Content Management Service. It extends the
2781 RegistryResponseType and does not define any new parameters.

2782

2783 **8.4.2.1 Syntax:**

```
2784 <complexType name="ContentManagementServiceResponseType">  
2785   <complexContent>  
2786     <extension base="rs:RegistryResponseType">  
2787       <sequence>  
2788         </sequence>  
2789     </extension>  
2790   </complexContent>  
2791 </complexType>
```

2792

2793 **8.4.2.2 Parameters:**

2794 No new parameters are defined other than those inherited from RegistryResponseType.
2795

2796 **8.5 Publishing / Configuration of a Content Management Service**

2797 Any Submitter MAY submit an arbitrary Content Management Service to an ebXML Registry. The Content
2798 Management Service MUST be published using the standard LifecycleManager interface.

2799 The Submitter MUST use the standard Submit/UpdateObjectsRequest to publish:

- 2800 ○ A Service instance for the Content Management Service. In Figure 17 this is exemplified by the
2801 defaultXMLCatalogingService in the upper-left corner. The Service instance MUST have an
2802 Association with a ClassificationNode in the canonical ObjectType ClassificationScheme as defined
2803 by [ebRIM]. The Service MUST be the sourceObject while a ClassificationNode MUST be the
2804 targetObject. This association binds the Service to that specific ObjectType. The associationType
2805 for this Association instance MUST be "ContentManagementServiceFor." The Service MUST be
2806 classified by the canonical ContentManagementService ClassificationScheme as defined by
2807 [ebRIM]. For example it may be classified as a "ContentValidationService" or a
2808 "ContentCatalogingService."
 - 2809 ○ The Service instance MAY be classified by a ClassificationNode under the canonical
2810 InvocationModel ClassificationScheme as defined by [ebRIM], to determine whether it uses the
2811 Inline Invocation model or the Decoupled Invocation model.
 - 2812 ○ The Service instance MAY be classified by a ClassificationNode under the canonical
2813 ErrorHandlingModel ClassificationScheme as defined by [ebRIM], to determine whether the Service
2814 should fail on first error or simply log the error as a warning and continue. See section 8.6.4 for
2815 details.
 - 2816 ○ A ServiceBinding instance contained within the Service instance that MUST provide the accessURI
2817 to the Cataloging Service.
 - 2818 ○ An optional ExternalLink instance on the ServiceBinding that is resolvable to a web page describing:
 - 2819 ■ The format of the supported content to be Cataloged
 - 2820 ■ The format of the supported Invocation Control File
- 2821 Note that no SpecificationLink is required since this specification [ebRS] is implicit for Content
2822 Cataloging Services.
- 2823 ○ One or more Invocation Control File(s) consisting of an ExtrinsicObject and a repository item pair.
2824 The ExtrinsicObject for the Invocation Control File MUST have a required Association with
2825 associationType value that references a descendant ClassificationNode of the canonical
2826 ClassificationNode "InvocationControlFileFor." This is exemplified by the
2827 cppCatalogingServiceXSLT and the oagBODCatalogingServiceXSLT objects in Figure 17 (left side
2828 of picture). The Invocation Control File MUST be the sourceObject while a ClassificationNode in the
2829 canonical ObjectType ClassificationScheme MUST be the targetObject.

2830

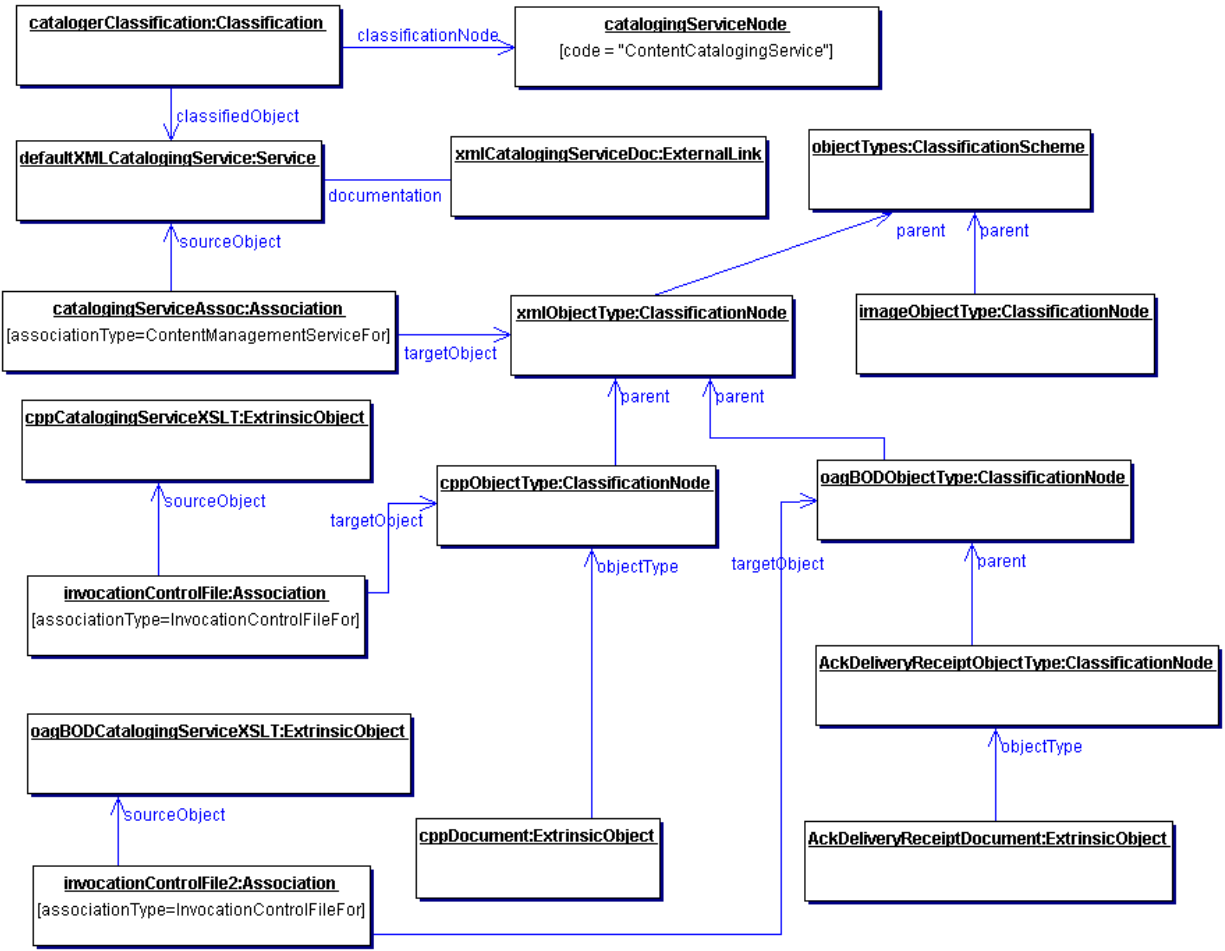


Figure 17: Cataloging Service Configuration

2831
2832

2833 Figure 17 shows an example of the configuration of the Canonical XML Cataloging Service associated with
 2834 the objectType for XML content. This Cataloging Service may be used with any XML content that has its
 2835 objectType attribute hold a reference to the xmlObjectType ClassificationNode or one of its descendants.

2836 The figure also shows two different Invocation Control Files, cppCatalogingServiceXSLT and
 2837 oagBODCatalogingServiceXSLT that may be used to catalog ebXML CPP and OAG Business Object
 2838 Documents (BOD) respectively.

2839 **8.5.1 Multiple Content Management Services and Invocation Control Files**

2840 This specification allows clients to submit multiple Content Management Services of the same type (e.g.
 2841 validation, cataloging) and multiple Invocation Control Files for the same objectType. Content Management
 2842 Services of the same type of service for the same ObjectType are referred to as peer Content Management
 2843 Services.

2844

2845 When there are multiple Content Management Services and Invocation Control Files for the same
 2846 ObjectType there MUST be an unambiguous association between a Content Management Service and its
 2847 Invocation Control File(s). This MUST be defined by an Association instance with associationType value that
 2848 references a ClassificationNode that is a descendant of the canonical ClassificationNode
 2849 "InvocationControlFileFor" where the ExtrinsicObject for each Invocation Control File is the sourceObject
 2850 and the Service is the targetObject.

2851 The order of invocation of peer Content Management Services is undefined and MAY be determined in a
 2852 registry specific manner.

2853 **8.6 Invocation of a Content Management Service**

2854 This section describes how a registry invokes a Content Management Service.

2855 **8.6.1 Resolution Algorithm For Service and Invocation Control File**

2856 When a registry receives a submission of a RegistryObject, it MUST use the following algorithm to
2857 determine or resolve the Content Management Services and Invocation Control Files to be used for dynamic
2858 content management for the RegistryObject:

2859

- 2860 1. Get the objectType attribute of the RegistryObject.
- 2861 2. Query to see if the ClassificationNode referenced by the objectType is the targetObject of an Association with
2862 associationType of *ContentManagementServiceFor*. If the desired Association is not found for this
2863 ClassificationNode then repeat this step with its parent ClassificationNode. Repeat until the desired
2864 Association is found or until the parent is the ClassificationScheme. If desired Association(s) is found then
2865 repeat following steps for each such Association instance.
- 2866 3. Check if the sourceObject of the desired Association is a Service instance. If not, log an
2867 InvalidConfigurationException. If it is a Service instance, then use this Service as the Content Management
2868 service for the RegistryObject.
- 2869 4. Query to see if the objectType ClassificationNode is the targetObject of one or more Associations whose
2870 associationType value references a ClassificationNode that is a descendant of the canonical
2871 ClassificationNode *InvocationControlFileFor*. If desired Association is not found for this ClassificationNode
2872 then repeat this step with its parent ClassificationNode. Repeat until the desired Association is found or until
2873 the parent is the ClassificationScheme.
- 2874 5. If desired Association(s) is found then check if the sourceObject of the desired Association is an
2875 ExtrinsicObject instance. If not, log an InvalidConfigurationException. If sourceObject is an ExtrinsicObject
2876 instance, then use its repository item as an Invocation Control File. If there are multiple
2877 InvocationControlFiles then all of them MUST be provided when invoking the Service.

2878 The above algorithm allows for objectType hierarchy to be used to configure Content Management Services
2879 and Invocation Control Files with varying degrees of specificity or specialization with respect to the type of
2880 content.

2881 **8.6.2 Audit Trail and Cataloged Content**

2882 The Cataloged Content generated as a result of the invocation of a Content Management Service has an
2883 audit trail consistent with RegistryObject instances that are submitted by Registry Clients. However, since a
2884 Registry Client does not submit Cataloged Content, the user attribute of the AuditableEvent instances for
2885 such Cataloged Content references the Service object for the Content Management Service that generated
2886 the Cataloged Content. This allows an efficient way to distinguish Cataloged Content from content submitted
2887 by Registry Clients.

2888 **8.6.3 Referential Integrity**

2889 A registry MUST maintain referential integrity between the RegistryObjects and repository items invocation
2890 of a Content Management Service.

2891 **8.6.4 Error Handling**

2892 If the Content Management Service is classified by the "FailOnError" ClassificationNode under canonical
2893 ErrorHandlerModel ClassificationScheme as defined by [ebRIM], then the registry MUST stop further
2894 processing of the Submit/UpdateObjectsRequest and return status of "Failure" upon first error returned by a
2895 Content Management Service Invocation.

2896 If the Content Management Service is classified by the "LogErrorAndContinue" ClassificationNode under
2897 ErrorHandlerModel then the registry MUST continue to process the Submit/UpdateObjectsRequest and not
2898 let any Content Management Service invocation error affect the storing of the RegistryObjects and

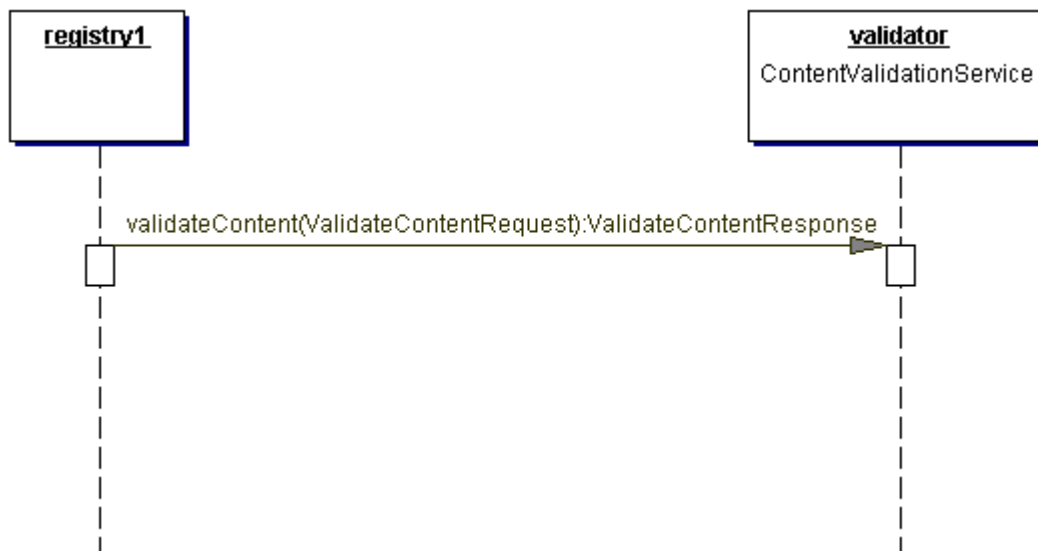
2899 repository items that were submitted. Such errors SHOULD be logged as Warnings within the
2900 RegistryResponse returned to the client. In this case a registry MUST return a normal response with status
2901 of "Success" if the submitted content and metadata is stored successfully even when there are errors
2902 encountered during dynamic invocation of one or more Content Management Services.

2903 8.7 Validate Content Protocol

2904 The interface of a Content Validation Service MUST implement a single method called validateContent. The
2905 validateContent method accepts a ValidateContentRequest as parameter and returns a
2906 ValidateContentResponse as its response if there are no errors.

2907 The OriginalContent element within a ValidateContentRequest MUST contain exactly one RegistryObject
2908 that needs to be cataloged. The resulting ValidateContentResponse contains the status attribute that
2909 communicates whether the RegistryObject (and its content) are valid or not.

2910 The Validate Content protocol does not specify the implementation details of any specific Content Validation
2911 Service.



2912
2913

Figure 18: Validate Content Protocol

2914 8.7.1 ValidateContentRequest

2915 The ValidateContentRequest is used to pass content to a Content Validation Service so that it can validate
2916 the specified RegistryObject and any associated content. The RegistryObject typically is an ExternalLink (in
2917 the case of external content) or an ExtrinsicObject. The ValidateContentRequest extends the base type
2918 ContentManagementServiceRequestType.

2919 8.7.1.1 Syntax:

```
2920 <element name="ValidateContentRequest">
2921   <complexType>
2922     <complexContent>
2923       <extension base="cms:ContentManagementServiceRequestType">
2924         <sequence>
2925           </sequence>
2926         </extension>
2927       </complexContent>
2928     </complexType>
2929   </element>
```

2930

2931 **8.7.1.2 Parameters:**

2932 The following parameters are parameters that are either newly defined for this type or are inherited and
2933 have additional semantics beyond those defined in the base type description.

- 2934 ▪ *InvocationControlFile*: Inherited from base type. This parameter may not be present. If
2935 present its format is defined by the Content Validation Service.
- 2936 ▪ *OriginalContent*: Inherited from base type. This parameter MUST contain exactly one
2937 RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content.
2938 This specification does not specify the format of the content. If it is an ExtrinsicObject then
2939 there MAY be a corresponding repository item as an attachment to this request that is the
2940 content. The corresponding repository item SHOULD follow the same rules as attachments
2941 in Submit/UpdateObjectsRequest.

2942

2943 **8.7.1.3 Returns:**

2944 This request returns a ValidateContentResponse. See section 8.7.2 for details.

2945 **8.7.1.4 Exceptions:**

2946 In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

- 2947 ▪ *InvalidContentException*: signifies that the specified content was found to be invalid. The
2948 exception SHOULD include enough detail for the client to be able to determine how to
2949 make the content valid.

2950

2951 **8.7.2 ValidateContentResponse**

2952 The ValidateContentResponse is sent by the Content Validation Service as a response to a
2953 ValidateContentRequest.

2954

2955 **8.7.2.1 Syntax:**

```
2956 <element name="ValidateContentResponse">  
2957   <complexType>  
2958     <complexContent>  
2959       <extension base="cms:ContentManagementServiceResponseType">  
2960         <sequence>  
2961           </sequence>  
2962         </extension>  
2963       </complexContent>  
2964     </complexType>  
2965   </element>
```

2966

2967 **8.7.2.2 Parameters:**

2968 The following parameters are parameters that are either newly defined for this type or are inherited and
2969 have additional semantics beyond those defined in the base type description.

- 2970 ▪ *status*: Inherited attribute. This enumerated value is used to indicate the status of the
2971 request. Values for status are as follows:

2972

- 2973 • Success - This status specifies that the content specified in the
2974 ValidateContentRequest was valid.

2975
2976
2977
2978
2979
2980

- Failure - This status specifies that the request failed. If the error returned is an InvalidContentException then the content specified in the ValidateContentRequest was invalid. If there was some other failure encountered during the processing of the request then a different error MAY be returned.

2981 8.8 Catalog Content Protocol

2982 The interface of the Content Cataloging Service MUST implement a single method called catalogContent.
2983 The catalogContent method accepts a CatalogContentRequest as parameter and returns a
2984 CatalogContentResponse as its response if there are no errors.

2985 The CatalogContentRequest MAY contain repository items that need to be cataloged. The resulting
2986 CatalogContentResponse contains the metadata and possibly content that gets generated or updated by
2987 the Content Cataloging Service as a result of cataloging the specified repository items.

2988 The Catalog Content protocol does not specify the implementation details of any specific Content
2989 Cataloging Service.



2990
2991

Figure 19: Catalog Content Protocol

2992 8.8.1 CatalogContentRequest

2993 The CatalogContentRequest is used to pass content to a Content Cataloging Service so that it can create
2994 catalog metadata for the specified RegistryObject and any associated content. The RegistryObject typically
2995 is an ExternalLink (in case of external content) or an ExtrinsicObject. The CatalogContentRequest extends
2996 the base type ContentManagementServiceRequestType.

2997 8.8.1.1 Syntax:

```
2998 <element name="CatalogContentRequest">
2999   <complexType>
3000     <complexContent>
3001       <extension base="cms:ContentManagementServiceRequestType">
3002         <sequence>
3003         </sequence>
3004       </extension>
3005     </complexContent>
3006   </complexType>
```

3006
3007
3008
3009

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

3010 **8.8.1.2 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 ▪ *InvocationControlFile*: Inherited from base type. If present its format is defined by the
3014 Content Cataloging Service.
- 3015 ▪ *OriginalContent*: Inherited from base type. This parameter MUST contain exactly one
3016 RegistryObject (e.g. ExternalLink, ExtrinsicObject) and potentially an associated content.
3017 This specification does not specify the format of the content. If it is an ExtrinsicObject then
3018 there MAY be a corresponding repository item as an attachment to this request that is the
3019 content. The corresponding repository item SHOULD follow the same rules as attachments
3020 in Submit/UpdateObjectsRequest.
3021

3022 **8.8.1.3 Returns:**

3023 This request returns a CatalogContentResponse. See section 8.8.2 for details.

3024 **8.8.1.4 Exceptions:**

3025 In addition to the exceptions returned by base request types, the following exceptions MAY be returned:

- 3026 ▪ *CatalogingException*: signifies that an exception was encountered in the Cataloging
3027 algorithm for the service.
3028

3029 **8.8.2 CatalogContentResponse**

3030 The CatalogContentResponse is sent by the Content Cataloging Service as a response to a
3031 CatalogContentRequest.

3032

3033 **8.8.2.1 Syntax:**

```
3034 <element name="CatalogContentResponse">  
3035   <complexType>  
3036     <complexContent>  
3037       <extension base="cms:ContentManagementServiceResponseType">  
3038         <sequence>  
3039           <element name="CatalogedContent"  
3040             type="rim:RegistryObjectListType"/>  
3041         </sequence>  
3042       </extension>  
3043     </complexContent>  
3044   </complexType>  
3045 </element>
```

3046

3047 **8.8.2.2 Parameters:**

3048 The following parameters are parameters that are either newly defined for this type or are inherited and
3049 have additional semantics beyond those defined in the base type description.

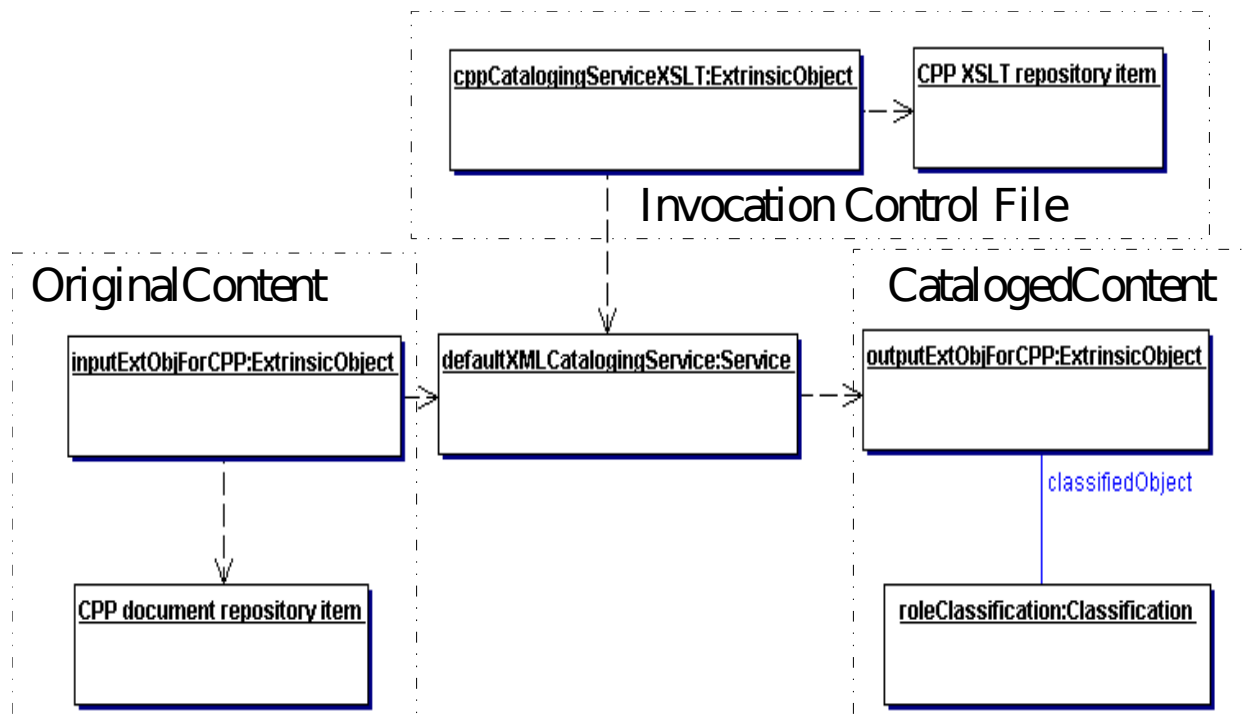
3050
3051
3052
3053
3054
3055
3056
3057

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

3058 8.9 Illustrative Example: Canonical XML Cataloging Service

3059 Figure 20 shows a UML instance diagram to illustrate how a Content Cataloging Service is used. This
3060 Content Cataloging Service is the normative Canonical XML Cataloging Service described in section 8.10.

- 3061 ○ In the center we see a Content Cataloging Service name defaultXMLCataloger Service.
- 3062 ○ On the left we see a CPP repository item and its ExtrinsicObject inputExtObjForCPP being input as
3063 Original Content to the defaultXMLCataloging Service.
- 3064 ○ On top we see an XSLT style sheet repository item and its ExtrinsicObject that is configured as an
3065 Invocation Control File for the defaultXMLCataloger Service.
- 3066 ○ On the right we see the outputExtObjForCPP, which is the modified ExtrinsicObject for the CPP. We
3067 also see a Classification roleClassification, which classifies the CPP by the Role element within the
3068 CPP. These are the Cataloged Content generated as a result of the Cataloging Service cataloging
3069 the CPP.



3070
3071
3072

Figure 20: Example of CPP cataloging using Canonical XML Cataloging Service

3073 8.10 Canonical XML Content Cataloging Service

3074 An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in
3075 service with the following constraints:

- 3076 • There is exactly one Service instance for the Canonical XML Content Cataloging Service

- 3077 • The Service is an XSLT engine
- 3078 • The Service may be invoked with exactly one Invocation Control File
- 3079 • The Original Content for the Service MUST be XML document(s)
- 3080 • The Cataloged Content for the Service MUST be XML document(s)
- 3081 • The Invocation Control File MUST be an XSLT style sheet
- 3082 • Each invocation of the Service MAY be with different Invocation Control File (XSLT style sheet)
- 3083 depending upon the objectType of the RegistryObject being cataloged. Each objectType SHOULD
- 3084 have its own unique XSLT style sheet. For example, ebXML CPP documents SHOULD have a
- 3085 specialized ebXML CPP Invocation Control XSLT style sheet.
- 3086 • The Service MUST have at least one input XML document that is a RegistryObject. Typically this is
- 3087 an ExtrinsicObject or an ExternalLink.
- 3088 • The Service MAY have at most one additional input XML document that is the content represented
- 3089 by the RegistryObject (e.g. a CPP document or an HL7 Conformance Profile). The optional second
- 3090 input MUST be referenced within the XSLT Style sheet by a using the “document” function with the
- 3091 document name specified by variable “repositoryItem” as in “document(\$repositoryItem).” A registry
- 3092 MUST define the variable “repositoryItem” when invoking the Canonical XML Cataloging Service.
- 3093 • The canonical XML Content Cataloging Service MUST apply the XSLT style sheet to the input XML
- 3094 instance document(s) in an XSLT transformation to generate the Cataloged Output.
- 3095 The Canonical XML Content Cataloging Service is a required normative feature of an ebXML Registry.

3096 **8.10.1 Publishing of Canonical XML Content Cataloging Service**

3097 An ebXML Registry MUST provide the canonical XML Content Cataloging Service natively as a built-in
3098 service. This built-in service MUST be published to the registry as part of the intrinsic bootstrapping of
3099 required canonical data within the registry.

9 Cooperating Registries Support

3100

3101 This chapter describes the capabilities and protocols that enable multiple ebXML registries to cooperate with
3102 each other to meet advanced use cases.

9.1 Cooperating Registries Use Cases

3103

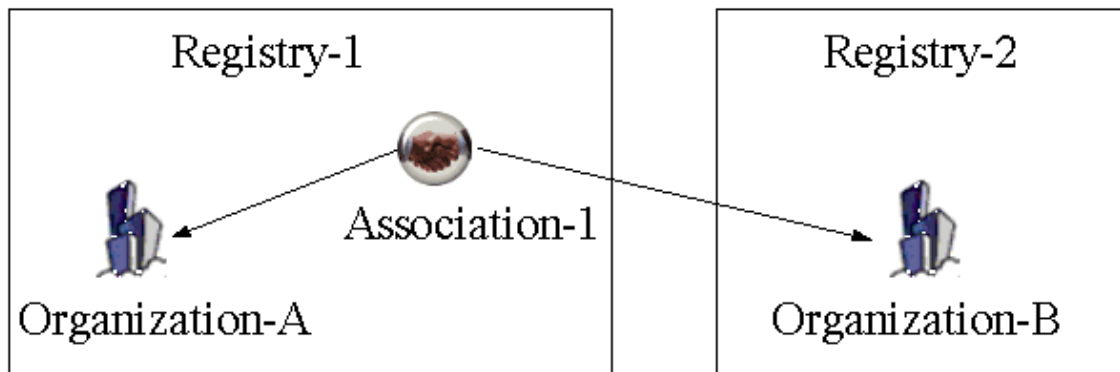
3104 The following is a list of use cases that illustrate different ways that ebXML registries cooperate with each
3105 other.

9.1.1 Inter-registry Object References

3106

3107 A Submitting Organization wishes to submit a RegistryObject to a registry such that the submitted object
3108 references a RegistryObject in another registry.

3109 An example might be where a RegistryObject in one registry is associated with a RegistryObject in another
3110 registry.



3111

3112

Figure 21: Inter-registry Object References

3113

9.1.2 Federated Queries

3114

3115 A client wishes to issue a single query against multiple registries and get back a single response that
3116 contains results based on all the data contained in all the registries. From the client's perspective it is
3117 issuing its query against a single logical registry that has the union of all data within all the physical
3118 registries.

9.1.3 Local Caching of Data from Another Registry

3119

3120 A destination registry wishes to cache some or all the data of another source registry that is willing to share
3121 its data. The shared dataset is copied from the source registry to the destination registry and is visible to
3122 queries on the destination registry even when the source registry is not available.

3123 Local caching of data may be desirable in order to improve performance and availability of accessing that
3124 object.

3125 An example might be where a RegistryObject in one registry is associated with a RegistryObject in another
3126 registry, and the first registry caches the second RegistryObject locally.

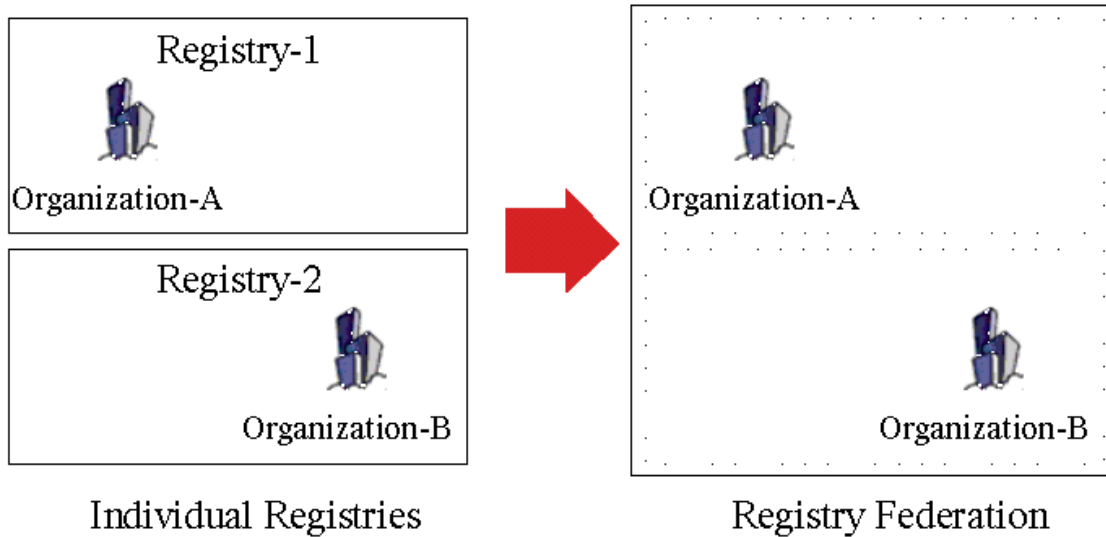
9.1.4 Object Relocation

3127

3128 A Submitting Organization wishes to relocate its RegistryObjects and/or repository items from the registry
3129 where it was submitted to another registry.

3130 **9.2 Registry Federations**

3131 A registry federation is a group of registries that have voluntarily agreed to form a loosely coupled union.
3132 Such a federation may be based on common business interests and specialties that the registries may
3133 share. Registry federations appear as a single logical registry to registry clients.



3134 **Figure 22: Registry Federations**

3135
3136 Registry federations are based on a peer-to-peer (P2P) model where all participating registries are equal.
3137 Each participating registry is called a *registry peer*. There is no distinction between the registry operator that
3138 created a federation and those registry operators that joined that Federation later.
3139 Any registry operator MAY form a registry federation at any time. When a federation is created it MUST have
3140 exactly one registry peer which is the registry operated by the registry operator that created the federation.
3141 Any registry MAY choose to voluntarily join or leave a federation at any time.

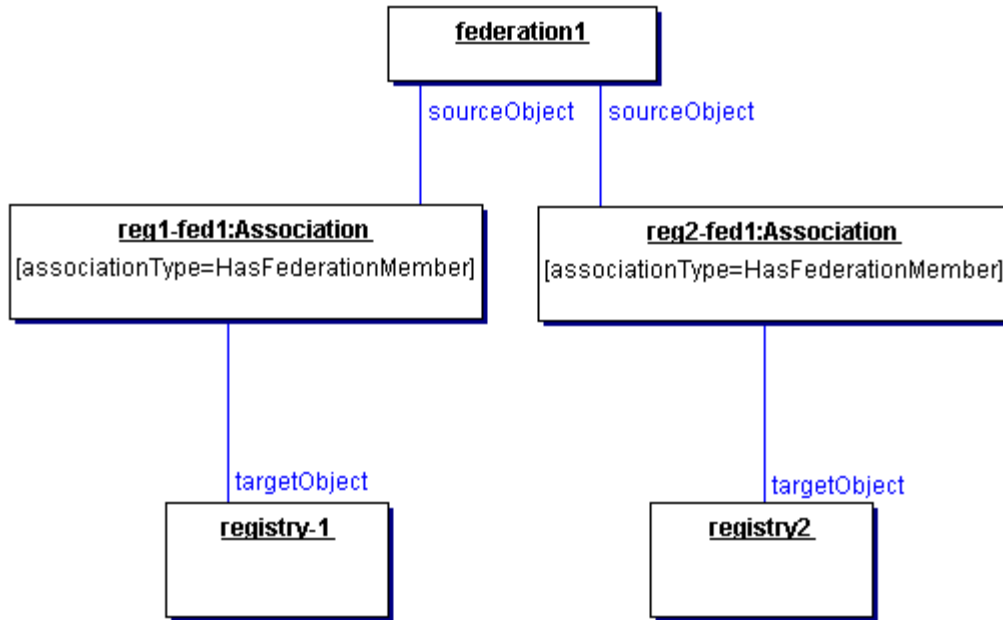
3142 **9.2.1 Federation Metadata**

3143 The Registry Information model defines the Registry and Federation classes. Instances of these classes
3144 and the associations between these instances describe a federation and its members. Such instance data is
3145 referred to as Federation Metadata. The Registry and Federation classes are described in detail in [ebRIM].

3146 The Federation information model is summarized here as follows:

- 3147 ○ A Federation instance represents a registry federation.
- 3148 ○ A Registry instance represents a registry that is a member of the Federation.
- 3149 ○ An Association instance with associationType of *HasFederationMember* represents membership of
3150 the registry in the federation. This Association links the Registry instance and the Federation
3151 instance.

3152



3153
3154

Figure 23: Federation Metadata Example

3155 9.2.2 Local Vs. Federated Queries

3156 A federation appears to registry clients as a single unified logical registry. An AdhocQueryRequest sent by a
3157 client to a federation member MAY be local or federated. A new boolean attribute named *federated* is added
3158 to AdhocQueryRequest to indicate whether the query is federated or not.

3159 9.2.2.1 Local Queries

3160 When the federated attribute of AdhocQueryRequest has the value of *false* then the query is a local query.
3161 In the absence of a *federated* attribute the default value of *federated* attribute is *false*.

3162 A local AdhocQueryRequest is only processed by the registry that receives the request. A local
3163 AdhocQueryRequest does not operate on data that belongs to other registries.

3164 9.2.2.2 Federated Queries

3165 When the *federated* attribute of AdhocQueryRequest has the value of *true* then the query is a federated
3166 query.

3167 A federation member MUST route a federated query received by it to all other federation member registries
3168 on a best attempt basis. If a member is not reachable for any reason then it MAY be skipped.

3169 When a registry routes a federated query to other federation members it MUST set the federated attribute
3170 value to *false* and the *federation* attribute value to null to avoid infinite loops.

3171 A federated query operates on data that belongs to all members of the federation.

3172 When a client submits a federated query to a registry such that the query specifies no federation and no
3173 federations exist in the registry, then the registry MUST treat it as a local query.

3174 When a client submits a federated query that invokes a parameterized stored query, the registry MUST
3175 resolve the parameterized stored query into its non-stored form and MUST replace all variables with user-
3176 supplied parameters on registry supplied contextual parameters before routing it to a federation member.

3177 When a client submits a federated iterative query, the registry MUST use the *startIndex* attribute value of the
3178 original request as the *startIndex* attribute value of the routed request sent to each federation member. The
3179 response to the original request MUST be the *union* of the results from each routed query. In such cases the
3180 registry MUST return a *totalResultCount* attribute value on the federated query response to be equal to the

3181 *maximum* of all *totalResultCount* attribute values returned by each federation member.

3182 **9.2.2.3 Membership in Multiple Federations**

3183 A registry *MAY* be a member of multiple federations. In such cases if the *federated* attribute of
3184 *AdhocQueryRequest* has the value of *true* then the registry *MUST* route the federated query to *all*
3185 federations that it is a member of.

3186 Alternatively, the client *MAY* specify the id of a specific federation that the registry is a member of, as the
3187 value of the *federation* parameter. The type of the federation parameter is anyURI and identifies the "id"
3188 attribute of the desired Federation.

3189 In such cases the registry *MUST* route the federated query to the specified federation only.

3190 **9.2.3 Federated Lifecycle Management Operations**

3191 Details on how to create and delete federations and how to join and leave a federation are described in
3192 9.2.8.

3193 All lifecycle operations *SHOULD* be performed on a *RegistryObject* within its home registry using the
3194 operations defined by the *LifeCycleManager* interface. Unlike query requests, lifecycle management
3195 requests do not support any federated capabilities.

3196 **9.2.4 Federations and Local Caching of Remote Data**

3197 A federation member is not required to maintain a local cache of replicas of *RegistryObjects* and repository
3198 items that belong to other members of the federation.

3199 A registry *MAY* choose to locally cache some or all data from any other registry whether that registry is a
3200 federation member or not. Data caching is orthogonal to registry federation and is described in section 9.3.

3201 Since by default there is minimal replication in the members of a federation, the federation architecture
3202 scales well with respect to memory and disk utilization at each registry.

3203 Data replication is often necessary for performance, scalability and fault-tolerance reasons.

3204 **9.2.5 Caching of Federation Metadata**

3205 A special case for local caching is the caching of the Federation and Registry instances and related
3206 Associations that define a federation and its members. Such data is referred to as federation metadata. A
3207 federation member is required to locally cache the federation metadata, from the federation home for each
3208 federation that it is a member of. The reason for this requirement is consistent with a Peer-to-Peer (P2P)
3209 model and ensures fault-tolerance in case the Federation home registry is unavailable.

3210 The federation member *MUST* keep the cached federation metadata synchronized with the master copy in
3211 the Federation home, within the time period specified by the *replicationSyncLatency* attribute of the
3212 Federation. Synchronization of cached Federation metadata may be done via synchronous polling or
3213 asynchronous event notification using the event notification feature of the registry.

3214 **9.2.6 Time Synchronization Between Registry Peers**

3215 Federation members are not required to synchronize their system clocks with each other. However, each
3216 Federation member *SHOULD* keep its clock synchronized with an atomic clock server within the latency
3217 described by the *replicationSyncLatency* attribute of the Federation.

3218 **9.2.7 Federations and Security**

3219 Federated operations abide by the same security rules as standard operations against a single registry.
3220 However, federation operations often require registry-to-registry communication. Such communication is
3221 governed by the same security rules as a Registry Client to registry communication. The only difference is
3222 that the requesting registry plays the role of Registry Client. Such registry-to-registry communication
3223 *SHOULD* be conducted over a secure channel such as HTTP/S. Federation members *SHOULD* be part of
3224 the same SAML Federation if member registries implement the Registry SAML Profile described in chapter

3225 11.

3226 **9.2.8 Federation Lifecycle Management Protocols**

3227 This section describes the various operations that manage the lifecycle of a federation and its membership.
3228 Federation lifecycle operations are done using standard LifeCycleManager interface of the registry in a
3229 stylized manner. Federation lifecycle operations are privileged operations. A registry SHOULD restrict
3230 Federation lifecycle operations to registry User's that have the RegistryAdministrator role.

3231 **9.2.8.1 Joining a Federation**

3232 The following rules govern how a registry joins a federation:

- 3233 • Each registry SHOULD have exactly one Registry instance within that registry for which it is a
3234 home. The Registry instance is owned by the RegistryOperator and may be placed in the registry
3235 using any operator specific means. The Registry instance SHOULD never change its home registry.
- 3236 • A registry MAY request to join an existing federation by submitting an instance of an Extramural
3237 Association that associates the Federation instance as sourceObject, to its Registry instance as
3238 targetObject, using an associationType of *HasFederationMember*. The home registry for the
3239 Association and the Federation objects MUST be the same.
3240

3241 **9.2.8.2 Creating a Federation**

3242 The following rules govern how a federation is created:

- 3243 • A Federation is created by submitting a Federation instance to a registry using
3244 SubmitObjectsRequest.
- 3245 • The registry where the Federation is submitted is referred to as the federation home.
- 3246 • The federation home may or may not be a member of that Federation.
- 3247 • A federation home MAY contain multiple Federation instances.

3248 **9.2.8.3 Leaving a Federation**

3249 The following rules govern how a registry leaves a federation:

3250 A registry MAY leave a federation at any time by removing its *HasFederationMember* Association instance
3251 that links it with the Federation instance. This is done using the standard RemoveObjectsRequest.

3252 **9.2.8.4 Dissolving a Federation**

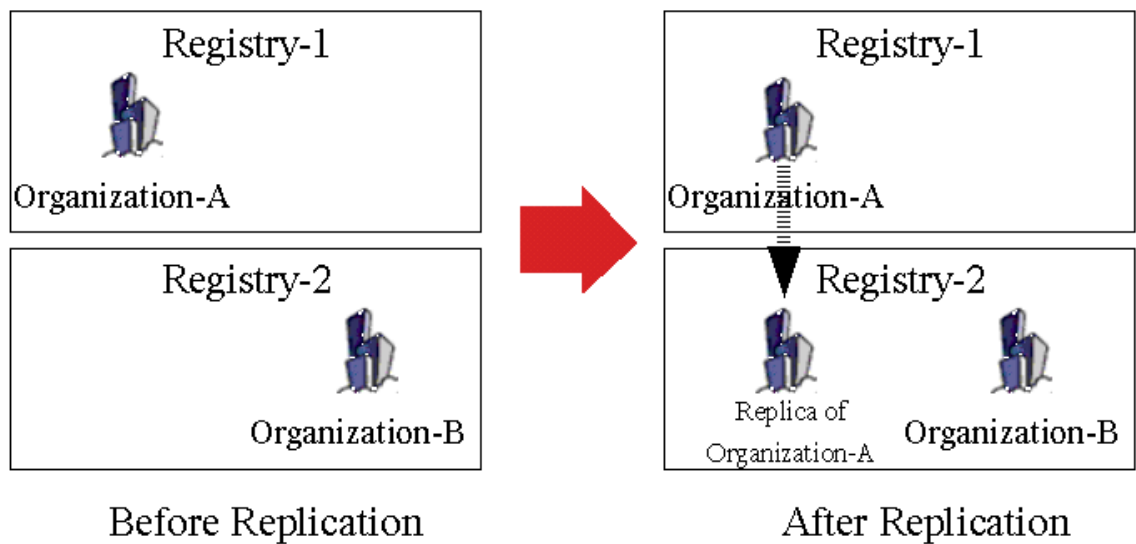
3253 The following rules govern how a federation is dissolved:

- 3254 • A federation is dissolved by sending a RemoveObjectsRequest to its home registry and removing its
3255 Federation instance.
- 3256 • The removal of a Federation instance is controlled by the same Access Control Policies that govern
3257 any RegistryObject.
- 3258 • The removal of a Federation instance is controlled by the same lifecycle management rules that
3259 govern any RegistryObject. Typically, this means that a federation MUST NOT be dissolved while it
3260 has federation members. It MAY however be deprecated at any time. Once a Federation is
3261 deprecated no new members can join it.
3262

3263 **9.3 Object Replication**

3264 RegistryObjects within a registry MAY be replicated in another registry. A replicated copy of a remote object
3265 is referred to as its replica. The remote object MAY be an original object or it MAY be a replica. A replica

3266 from an original is referred to as a first-generation replica. A replica of a replica is referred to as a second-
 3267 generation replica (and so on).
 3268 The registry that replicates a remote object locally is referred to as the destination registry for the replication.
 3269 The registry that contains the remote object being replicated is referred to as the source registry for the
 3270 replication.
 3271



3272
 3273 **Figure 24: Object Replication**
 3274

3275 **9.3.1 Use Cases for Object Replication**

3276 A registry MAY create a local replica of a remote object for a variety of reasons. A few sample use cases
 3277 follow:

- 3278 ○ Improve access time and fault tolerance by locally caching remote objects. For example, a registry
 3279 MAY automatically create a local replica when a remote ObjectRef is submitted to the registry.
- 3280 ○ Improve scalability by distributing access to hotly contested objects, such as NAICS scheme,
 3281 across multiple replicas.
- 3282 ○ Enable cooperating registry features such as hierarchical registry topology and local caching of
 3283 federation metadata.

3284 **9.3.2 Queries And Replicas**

3285 A registry MUST support client queries to consider a local replica of remote object as if it were a local object.
 3286 Local replicas are considered within the extent of the data set of a registry as far as local queries are
 3287 concerned.

3288 When a client submits a local query that retrieves a remote object by its id attribute, if the registry contains a
 3289 local replica of that object then the registry SHOULD return the state defined by the local replica.

3290 **9.3.3 Lifecycle Operations And Replicas**

3291 LifeCycle operations on an original object MUST be performed at the home registry for that object. LifeCycle
 3292 operations on replicas of an original object should result in an InvalidRequestException.

3293 **9.3.4 Object Replication and Federated Registries**

3294 Object replication capability is orthogonal to the registry federation capability. Objects MAY be replicated

3295 from any registry to any other registry without any requirement that the registries belong to the same
3296 federation.

3297 **9.3.5 Creating a Local Replica**

3298 Any Submitting Organization can create a replica by using the standard SubmitObjectsRequest. If a registry
3299 receives a SubmitObjectsRequest that has a RegistryObjectList containing a remote ObjectRef, then it
3300 MUST create a replica for that remote ObjectRef. In such cases the User that submitted the ObjectRef (via
3301 a SubmitObjectsRequest) owns the replica while the original RegistryObject is owned by its original owner.

3302 In addition to Submitting Organizations, a registry itself MAY create a replica under specific situations in a
3303 registry specific manner.

3304 Creating a local replica requires the destination registry to read the state of the remote object from the
3305 source registry and then create a local replica of the remote object.

3306 A registry SHOULD use standard QueryManager interface to read the state of a remote object (whether it is
3307 an original or a replica). No new APIs are needed to read the state of a remote object. Since query
3308 functionality does not need prior registration, no prior registration or contract is needed for a registry to read
3309 the state of a remote object.

3310 Once the state of the remote object has been read, a registry MAY use registry specific means to create a
3311 local replica of the remote object. Such registry specific means MAY include the use of the
3312 LifeCycleManager interface.

3313 A replica of a RegistryObject may be distinguished from an original since a replica MUST have its home
3314 attribute point to the remote registry where the original for the replica resides.

3315 **9.3.6 Transactional Replication**

3316 Transactional replication enables a registry to replicate events in another registry in a transactionally
3317 consistent manner. This is typically the case when entire registries are replicated to another registry.

3318 This specification defines a more loosely coupled replication model as an alternative to transactional
3319 replication for the following reasons:

- 3320 • Transactional replication requires a tight coupling between registries participating in the
3321 replication
- 3322 • Transactional replication is not a typical use case for registries
- 3323 • Loosely coupled replication as defined by this specification typically suffices for most use cases
- 3324 • Transaction replication is very complex and error prone

3325

3326 Registry implementations are not required to implement transactional replication.

3327 **9.3.7 Keeping Replicas Current**

3328 A registry MUST keep its replicas current within the latency specified by the value of the
3329 *replicationSyncLatency* attribute defined by the registry. This includes removal of the replica when its original
3330 is removed from its home registry.

3331 Replicas MAY be kept current using the event notification feature of the registry or via periodic polling.

3332 **9.3.8 Lifecycle Management of Local Replicas**

3333 Local Replicas are read-only objects. Lifecycle management actions are not permitted on local replicas with
3334 the exception of the Delete action which is used to remove the replica. All other lifecycle management
3335 actions MUST be performed on the original RegistryObject in the home registry for the object.

3336 **9.3.9 Tracking Location of a Replica**

3337 A local replica of a remote RegistryObject instance MUST have exactly one ObjectRef instance within the
3338 local registry. The home attribute of the ObjectRef associated with the replica tracks its home location. A
3339 RegistryObject MUST have exactly one home. The home for a RegistryObject MAY change via Object
3340 Relocation as described in section 9.4. It is optional for a registry to track location changes for replicas
3341 within it.

3342 **9.3.10 Remote Object References to a Replica**

3343 It is possible to have a remote ObjectRef to a RegistryObject that is a replica of another RegistryObject. In
3344 such cases the home attribute of the ObjectRef contains the base URI to the home registry for the replica.

3345 **9.3.11 Removing a Local Replica**

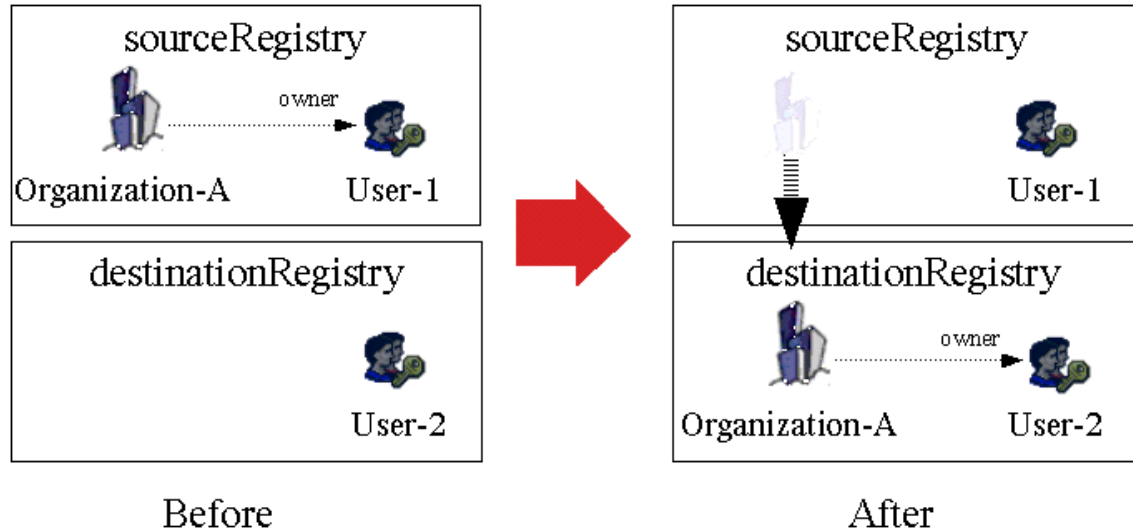
3346 A client can remove a replica by using the RemoveObjectsRequest. If a registry receives a
3347 RemoveObjectsRequest that has an ObjectRefList containing a remote ObjectRef, then it MUST remove the
3348 local replica for that remote ObjectRef assuming that the client was authorized to remove the replica.

3349 **9.4 Object Relocation Protocol**

3350 Every RegistryObject has a home registry and a User within the home registry that is the Submitter or owner
3351 of that object. Initially, the home registry is the where the object is originally submitted. Initially, the owner is
3352 the User that submitted the object.

3353 A RegistryObject MAY be relocated from one home registry to another home registry using the Object
3354 Relocation protocol.

3355 Within the Object Relocation protocol, the new home registry is referred to as the *destination* registry while
3356 the previous home registry is called the *source* registry.



3357
3358

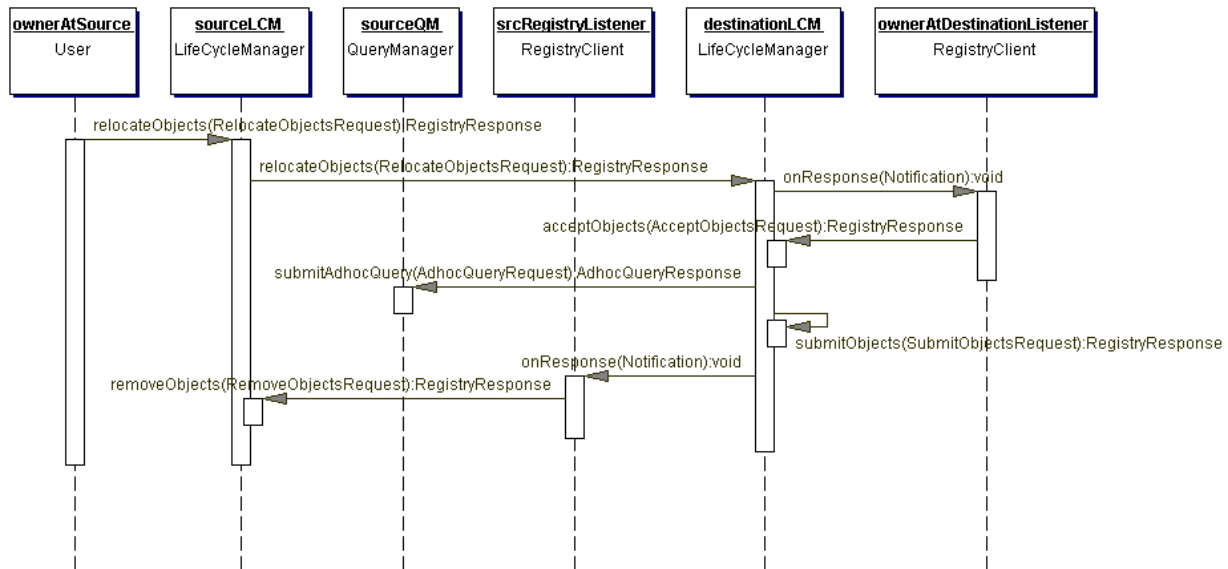
Figure 25: Object Relocation

3359 The User at the source registry who owns the objects being relocated is referred to as the *ownerAtSource*.
3360 The User at the destination registry, who is the new owner of the objects, is referred to as the
3361 *ownerAtDestination*. While the ownerAtSource and the ownerAtDestination may often be the same, the
3362 Object Relocation protocol treats them as two distinct identities.

3363 A special case usage of the Object Relocation protocol is to transfer ownership of RegistryObjects from one
3364 User to another within the same registry. In such cases the protocol is the same except for the fact that the
3365 source and destination registries are the same.

3366 Following are some notable points regarding object relocation:

- 3367 • Object relocation does not require that the source and destination registries be in the same
3368 federation or that either registry have a prior contract with the other.
- 3369 • Object relocation MUST preserve object id. While the home registry for a RegistryObject MAY
3370 change due to object relocation, its id never changes.
- 3371 • ObjectRelocation MUST preserve referential integrity of RegistryObjects. Relocated objects that
3372 have references to an object that did not get relocated MUST preserve their reference. Similarly
3373 objects that have references to a relocated object MUST also preserve their reference. Thus,
3374 relocating an object may result in making the value of a reference attribute go from being a local
3375 reference to being a remote reference or vice versa.
- 3376 • AcceptObjectsRequest does not include ObjectRefList. It only includes an opaque transactionId
3377 identifying the relocateObjects transaction.
- 3378 • The requests defined by the Relocate Objects protocol MUST be sent to the source or destination
3379 registry only.
- 3380 • When an object is relocated an AuditableEvent of type “Relocated” MUST be recorded by the
3381 sourceRegistry. Relocated events MUST have the source and destination registry’s base URIs
3382 recorded as two Slots on the Relocated event. The names of these Slots are:
 - 3383 ○ urn:oasis:names:tc:ebxml-regrep:rs:events:sourceRegistry
 - 3384 ○ urn:oasis:names:tc:ebxml-regrep:rs:events:destinationRegistry
- 3385



3386 **Figure 26: Relocate Objects Protocol**
3387

3388 Figure 26 illustrates the Relocate Objects Protocol. The participants in the protocol are the ownerAtSource
3389 and ownerAtDestination User instances as well as the LifeCycleManager interfaces of the sourceRegistry
3390 and destinationRegistry.

3391 The steps in the protocol are described next:

- 3392 1. The protocol is initiated by the ownerAtSource sending a RelocateObjectsRequest message to the
3393 LifeCycleManager interface of the sourceRegistry. The sourceRegistry MUST make sure that the
3394 ownerAtSource is authorized to perform this request. The id of this RelocateObjectsRequest is used
3395 as the transaction identifier for this instance of the protocol. This RelocateObjectsRequest message
3396 MUST contain an ad hoc query that specifies the objects that are to be relocated.
- 3397 2. Next, the sourceRegistry MUST relay the same RelocateObjectsRequest message to the
3398 LifeCycleManager interface of the destinationRegistry. This message enlists the destinationRegistry

- 3399 to participate in relocation protocol. The destinationRegistry MUST store the request information
3400 until the protocol is completed or until a registry specific period after which the protocol times out.
- 3401 3. The destinationRegistry MUST relay the RelocateObjectsRequest message to the
3402 ownerAtDestination. This notification MAY be done using the event notification feature of the
3403 registry as described in chapter 7. The notification MAY be done by invoking a listener Service for
3404 the ownerAtDestination or by sending an email to the ownerAtDestination. This concludes the first
3405 phase of the Object Relocation protocol.
- 3406 4. The ownerAtDestination at a later time MAY send an AcceptObjectsRequest message to the
3407 destinationRegistry. This request MUST identify the object relocation transaction via the
3408 *correlationId*. The value of this attribute MUST be the id of the original RelocateObjectsRequest.
- 3409 5. The destinationRegistry sends an AdhocQueryRequest message to the sourceRegistry. The source
3410 registry returns the objects being relocated as an AdhocQueryResponse. In the event of a large
3411 number of objects this may involve multiple AdhocQueryRequest/responses as described by the
3412 iterative query feature described in section 6.2.
- 3413 6. The destinationRegistry submits the relocated data to itself assigning the identity of the
3414 ownerAtDestination as the owner. The relocated data MAY be submitted to the destination registry
3415 using any registry specific means or a SubmitObjectsRequest. However, the effect SHOULD be the
3416 same as if a SubmitObjectsRequest was used.
- 3417 7. The destinationRegistry notifies the sourceRegistry that the relocated objects have been safely
3418 committed using the Event Notification feature of the registry as described in chapter 7.
- 3419 8. The sourceRegistry removes the relocated objects using any registry specific means and logging an
3420 AuditableEvent of type Relocated. This concludes the Object Relocation transaction.

3421 9.4.1 RelocateObjectsRequest

```
3422 <element name="RelocateObjectsRequest">
3423   <complexType>
3424     <complexContent>
3425       <extension base="rs:RegistryRequestType">
3426         <sequence>
3427           <element name="Query" type="rim:AdhocQueryType"/>
3428           <element name="SourceRegistry" type="rim:ObjectRefType"/>
3429           <element name="DestinationRegistry" type="rim:ObjectRefType"/>
3430           <element name="OwnerAtSource" type="rim:ObjectRefType"/>
3431           <element name="OwnerAtDestination" type="rim:ObjectRefType"/>
3432         </sequence>
3433       </extension>
3434     </complexContent>
3435   </complexType>
3436 </element>
```

3437

3438 9.4.1.1 Parameters:

- 3439 ▪ *id*: the attribute id provides the transaction identifier for this instance of the protocol.
 - 3440 ▪ *AdhocQuery*: This element specifies an ad hoc query that selects the RegistryObjects that are being
3441 relocated.
 - 3442 ▪ *sourceRegistry*: This element specifies the ObjectRef to the sourceRegistry Registry instance. The
3443 value of this attribute MUST be a local reference when the message is sent by the ownerAtSource to
3444 the sourceRegistry.
 - 3445 ▪ *destinationRegistry*: This element specifies the ObjectRef to the destinationRegistry Registry
3446 instance.
 - 3447 ▪ *ownerAtSource*: This element specifies the ObjectRef to the ownerAtSource User instance.
 - 3448 ▪ *ownerAtDestination*: This element specifies the ObjectRef to the ownerAtDestination User instance.
- 3449

3450 **9.4.1.2 Returns:**

3451 This request returns a RegistryResponse. See section 2.1.4 for details.

3452 **9.4.1.3 Exceptions:**

3453 In addition to the exceptions common to all requests, the following exceptions MAY be returned:

- 3454 *ObjectNotFoundException*: signifies that the specified Registry or User was not found in the
3455 registry.

3456

3457 **9.4.2 AcceptObjectsRequest**

```
3458 <element name="AcceptObjectsRequest">  
3459   <complexType>  
3460     <complexContent>  
3461       <extension base="rs:RegistryRequestType">  
3462         <attribute name="correlationId" use="required"  
3463 type="{http://www.w3.org/2001/XMLSchema}anyURI" />  
3464       </extension>  
3465     </complexContent>  
3466   </complexType>  
3467 </element>
```

3468

3469 **9.4.2.1 Parameters:**

- 3470 *correlationId*: Provides the transaction identifier for this instance of the protocol.

3471

3472 **9.4.2.2 Returns:**

3473 This request returns a RegistryResponse. See section 2.1.4 for details.

3474 **9.4.2.3 Exceptions:**

3475 In addition to the exceptions common to all requests, the following exceptions MAY be returned:

- 3476 *InvalidRequestException*: signifies that the specified correlationId was not found to match
3477 an ongoing RelocateObjectsRequest in the registry.

3478

3479 **9.4.3 Object Relocation and Remote ObjectRefs**

3480 The following scenario describes what typically happens when a person moves:

- 3481 1. When a person moves from one house to another, other persons may have their old postal
3482 addresses.
- 3483 2. When a person moves, they leave their new address as the forwarding address with the post office.
- 3484 3. The post office forwards their mail for some time to their new address.
- 3485 4. Eventually the forwarding request expires and the post office no longer forwards mail for that
3486 person.
- 3487 5. During this forwarding interval the person notifies interested parties of their change of address.

3488 The Object Relocation feature supports a similar model for relocation of RegistryObjects. The following
3489 steps describe the expected behavior when an object is relocated.

- 3490 1. When a RegistryObject O1 is relocated from one registry R1 to another registry R2, other
3491 RegistryObjects may have remote ObjectRefs to O1.

- 3492 2. The registry R1 MUST create an AuditableEvent of type Relocated that includes the home URI for
3493 the new registry R2.
- 3494 3. As long as the AuditableEvent exists in R1, if R1 gets a request to retrieve O1 by id, it MUST
3495 forward the request to R2 and transparently retrieve O1 from R2 and deliver it to the client. The
3496 object O1 MUST include the home URI to R2 within the optional home attribute of RegistryObject.
3497 Clients are advised to check the home attribute and update the home attribute of their local
3498 ObjectRef to match the new home URI value for the object.
- 3499 4. Eventually the AuditableEvent is cleaned up after a registry specific interval. R1 is no longer
3500 required to relay requests for O1 to R2 transparent to the client. Instead R1 MUST return an
3501 ObjectNotFoundException.
- 3502 5. Clients that are interested in the relocation of O1 and being notified of its new address may choose
3503 to be notified by having a prior subscription using the event notification facility of the registry. For
3504 example a Registry that has a remote ObjectRefs to O1 may create a subscription on relocation
3505 events for O1. This however, is not required behavior.

3506 **9.4.4 Notification of Object Relocation To ownerAtDestination**

3507 This section describes how the destinationRegistry uses the event notification feature of the registry to notify
3508 the ownerAtDestination of a Relocated event.

3509 The destinationRegistry MUST send a Notification with the following required characteristics:

- 3510 • The notification MUST be an instance of a Notification element.
- 3511 • The Notification instance MUST have at least one Slot as follows:
 - 3512 o The Slot MUST have the name:
3513 `urn:oasis:names:tc:ebxml-regrep:rs:events:correlationId`
 - 3514 o The Slot MUST have the correlationId for the Object Relocation transaction as the value of
3515 the Slot.

3516

3517 **9.4.5 Notification of Object Commit To sourceRegistry**

3518 This section describes how the destinationRegistry uses the event notification feature of the registry to notify
3519 the sourceRegistry that it has completed committing the relocated objects.

3520 The destinationRegistry MUST send a Notification with the following required characteristics:

- 3521 • The notification MUST be an instance of a Notification element.
- 3522 • The Notification instance MUST have at least one Slot as follows:
 - 3523 o The Slot MUST have the name
3524 `urn:oasis:names:tc:ebxml-regrep:rs:events:objectsCommitted`
 - 3525 o The Slot MUST have the value of *true*.

3526

3527 **9.4.6 Object Ownership and Owner Reassignment**

3528 A registry MUST determine the ownership of a RegistryObject based upon the most recent AuditableEvent
3529 that has the eventType matching the canonical EventType ClassificationNode for Create or Relocate events.

3530 A special case of Object Relocation is when an ObjectRelocationRequest to a registry specifies the same
3531 registry as sourceRegistry and destinationRegistry. In such cases the request is effectively to change the
3532 owner of the specified objects from current owner to a new owner.

3533 In such case if the client does not have the RegistryAdministrator role then the protocol requires the
3534 ownerAtDestination to issue an AcceptObjectsRequest as described earlier.

3535 However, if the client does have the RegistryAdministrator role then the registry MUST change the owner of

3536 the object to the user specified as ownerAtDestination without the ownerAtDestination to issue an
3537 AcceptObjectsRequest.

3538 **9.4.7 Object Relocation and Timeouts**

3539 No timeouts are specified for the Object Relocation protocol. Registry implementations MAY cleanup
3540 incomplete Object Relocation transactions in a registry specific manner as an administrative task using
3541 registry specific policies.

3542

3543

10 Registry Security

3544
3545
3546

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

3547
3548
3549
3550

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

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

3551

10.1 Security Use Cases

3552
3553

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

3554

10.1.1 Identity Management

3555
3556
3557
3558

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

3559

10.1.2 Message Security

3560
3561
3562
3563

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

3564

10.1.3 Repository Item Security

3565
3566
3567
3568

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

3569

10.1.4 Authentication

3570
3571
3572

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

3573

10.1.5 Authorization and Access Control

3574
3575
3576
3577

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

3578

10.1.6 Audit Trail

3579
3580
3581
3582

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

3583 **10.2 Identity Management**

3584 An ebXML Registry MUST provide an Identity Management mechanism that allows identities and credentials
3585 to be registered for authorized users of the registry and subsequently managed.

3586 If a registry implements the Registry SAML Profile as described in chapter 11 then the Identity Management
3587 capability MUST be provided by an Identity Provider service that integrates with the registry using the SAML
3588 2.0 protocols as defined by [SAMLCore].

3589 If a registry does not implement the Registry SAML Profile then it MUST provide User Registration and
3590 Identity Management functionality in an implementation specific manner.

3591 **10.3 Message Security**

3592 A registry MUST provide mechanisms to securely exchange messages between a Registry Requestor and
3593 the registry to ensure data and source integrity as described in this section.

3594 **10.3.1 Transport Layer Security**

3595 A registry MUST support HTTP/S communication between an HTTP Requestor and its HTTP interface
3596 binding. A registry MUST also support HTTP/S communication between a SOAP Requestor and its SOAP
3597 interface binding when the underlying transport protocol is HTTP.

3598 HTTP/S support SHOULD allow for both SSL and TLS as transport protocols.

3599 **10.3.2 SOAP Message Security**

3600 A registry MUST support signing and verification of all registry protocol messages (requests and responses)
3601 between a SOAP Requestor and its SOAP binding. Such mechanisms MUST conform to [WSI-BSP], [WSS-
3602 SMS], [WSS-SWA] and [XMLDSIG]. The reader should refer to these specifications for details on these
3603 message security mechanisms.

3604 **10.3.2.1 Request Message Signature**

3605 When a Registered User sends a request message to the registry, the requestor SHOULD sign the request
3606 message with a Message Signature. This ensures the integrity of the message and also enables the registry
3607 to perform authentication and authorization for the request. If the registry receives a request that does not
3608 include a Message signature then it MUST implicitly treat the request as coming from a Registry Guest. A
3609 Registered User need not sign a request message with a Message Signature when the SOAP
3610 communication is conducted over HTTP/S as the message security is handled by the transport layer
3611 security provided by HTTP/S in this case.

3612 When a Registered User sends a request message to the registry that contains a RepositoryItem as a
3613 SOAP Attachment, the requestor MUST also reference and sign the RepositoryItem from the message
3614 signature. This MUST conform to [RFC2392] and [WSS-SWA].

3615 If the registry receives a request containing an unsigned RepositoryItem then it MUST return an
3616 UnsignedRepositoryItemException.

3617 **10.3.2.2 Response Message Signature**

3618 When a Registered User sends a request message to the registry, the registry MAY use a pre-established
3619 preference policy or a default policy to determine whether the response message SHOULD be signed with a
3620 Message Signature. When a Registry Guest sends a request, the Registration Authority MAY use a default
3621 policy to determine whether the response contains a header signature. A registry need not sign a response
3622 message with a Message Signature when the SOAP communication is conducted over HTTP/S as the
3623 message security is handled by the transport layer security provided by HTTP/S in this case.

3624 When a registry sends a signed response message to a Registry Client that contains a RepositoryItem as a
3625 SOAP Attachment, the registry MUST also reference and sign the RepositoryItem from the message
3626 signature. This MUST conform to [RFC2392] and [WSS-SWA].

3627 If the Registry Client receives a signed response with a RepositoryItem that does not include a

3628 RepositoryItem Signature then it SHOULD not trust the integrity of the response and treat it as an error
3629 condition.

3630 10.3.2.3 KeyInfo Requirements

3631 The sender of a registry protocol message (Registry Requestor and Registry) SHOULD provide their public
3632 key under the <wsse:Security> element. If provided, it MUST be contained in a
3633 <wsse:BinarySecurityToken> element and MUST be referenced from the <ds:KeyInfo> element in the
3634 Message Signature. The value of wsu:Id attribute of the <wsse:BinarySecurityToken> containing the
3635 senders public key MUST be `urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert`.
3636 The <wsse:BinarySecurityToken> SHOULD contain a X509 Certificate.

3637 Listing 3 shows an example of Message signature including specifying the KeyInfo.

3638 10.3.2.4 Message Signature Validation

3639 Signature validation ensures message and attached RepositoryItems integrity and security, concerning both
3640 data and source.

3641 If the registry receives a request containing a Message Signature then it MUST validate the Message
3642 Signature as defined by [WSS-SMS]. In case the request contains an attached RepositoryItem it MUST
3643 validate the RepositoryItems signature as defined by [WSS-SWA].

3644 If the Registry Requestor receives a response containing a Message Signature then it SHOULD validate the
3645 Message Signature as defined by [WSS-SMS]. In case the response contains an attached RepositoryItem
3646 then it SHOULD validate the RepositoryItem signature as defined by [WSS-SWA].

3647 10.3.2.5 Message Signature Example

3648 The following example shows the format of a Message Signature:

```
3649 <soap:Envelope>  
3650 <soap:Header>  
3651 <wsse:Security>  
3652 <wsse:BinarySecurityToken EncodingType="http://docs.oasis-  
3653 open.org/wss/2004/01/oasis-200401-wss-soap-message-security-  
3654 1.0#Base64Binary" ValueType="http://docs.oasis-  
3655 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"  
3656 wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">  
3657 lui+Jy4WYKGGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnWSW  
3658 kXm9jAEdsm/  
3659 hs+f3NwvK23bh46mNmCQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yTcI  
3660 7XU7xZT54S9  
3661 hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrSFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI9W  
3662 zxPCfHdalN4  
3663 rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUc9QY3VjwNALgGDaEAT7gpURkCI85H  
3664 jdnSA5SM4cY  
3665 7jAsYX/CIpEkRJcBULLTEFrBZIBYDPzRWlSdsJRJngF7yCoGJW+/HYOyP8P4OM59F  
3666 Di0kM8GwOE0  
3667 WgYrJHH92qaVhoiPTLi7  
3668 </wsse:BinarySecurityToken>  
3669 <ds:Signature>  
3670 <!--The Message Signature -->  
3671 <ds:SignedInfo>  
3672 <ds:CanonicalizationMethod  
3673 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">  
3674 <c14n:InclusiveNamespaces PrefixList="wsse soap"  
3675 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"/>  
3676 </ds:CanonicalizationMethod>  
3677 <ds:SignatureMethod  
3678 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>  
3679 <ds:Reference URI="#TheBody">  
3680 <ds:Transforms>
```

```

3681         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
3682 c14n#">
3683             <c14n:InclusiveNamespaces PrefixList=""
3684 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3685         </ds:Transform>
3686     </ds:Transforms>
3687     <ds:DigestMethod
3688 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
3689     <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValue>
3690     </ds:Reference>
3691 </ds:SignedInfo>
3692 <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureValu
3693 e>
3694     <ds:KeyInfo>
3695         <wsse:SecurityTokenReference>
3696             <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
3697 regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
3698 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
3699         </wsse:SecurityTokenReference>
3700     </ds:KeyInfo>
3701 </ds:Signature>
3702 </wsse:Security>
3703 </soap:Header>
3704 <soap:Body wsu:Id="TheBody">
3705     <lcm:SubmitObjectsRequest />
3706 </soap:Body>
3707 </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 RespositoryItem:

```

3713 Content-Type: multipart/related; boundary="BoundaryStr" type="text/xml"
3714 --BoundaryStr
3715 Content-Type: text/xml
3716 <soap:Envelope>
3717     <soap:Header>
3718         <wsse:Security>
3719             <wsse:BinarySecurityToken EncodingType="http://docs.oasis-
3720 open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
3721 1.0#Base64Binary" ValueType="http://docs.oasis-
3722 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
3723 wsu:Id="urn:oasis:names:tc:ebxml-regrep:rs:security:SenderCert">
3724         lui+Jy4WYKKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV1A2RnWSW
3725         kXm9jAEdsm/
3726         hs+f3NwvK23bh46mNmNcQVsUYHbYAREZpykrd/eRwNgx8T+ByeFhmSviW77n6yTcI
3727         7XU7xZT54S9
3728         hTSyBLN2Sce1dEQpQXh5ssZK9aZTMrsFT1NBvNHC3Qq7w0Otr5V4axH3MXffsuI9W
3729         zxPCfHdalN4
3730         rLRfNY318pc6bn00zAMw0omUWwBEJZxxBGGUc9QY3VjwNALgGDaEAT7gpURkCI85H
3731         jdnSA5SM4cY
3732         7jAsYX/CIPekRJcBULLTEFrBZIBYDPzRWlSdsJRJngF7yCoGWJ+/HYOyP8P4OM59F
3733         Di0kM8GwOE0
3734         WgYrJHH92qaVhoiPTLi7
3735     </wsse:BinarySecurityToken>
3736     <ds:Signature>
3737         <!-- The Message Signature -->
3738     </ds:SignedInfo>

```

```

3739         <ds:CanonicalizationMethod
3740 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
3741         <c14n:InclusiveNamespaces PrefixList="wsse soap"
3742 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3743         </ds:CanonicalizationMethod>
3744         <ds:SignatureMethod
3745 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
3746         <ds:Reference URI="#TheBody">
3747         <ds:Transforms>
3748         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
3749 c14n#">
3750         <c14n:InclusiveNamespaces PrefixList=""
3751 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3752         </ds:Transform>
3753         </ds:Transforms>
3754         <ds:DigestMethod
3755 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
3756         <ds:DigestValue>i3qi5GjhHnfoBn/jOjQp2mq0Na4=</ds:DigestValue>
3757         </ds:Reference>
3758         </ds:SignedInfo>
3759
3760         <!--A reference to a RepositoryItem (one for each RepositoryItem)
3761 -->
3762         <ds:SignedInfo>
3763         <ds:CanonicalizationMethod
3764 Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#&quot; ">
3765         <c14n:InclusiveNamespaces PrefixList="wsse soap"
3766 xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#" />
3767         </ds:CanonicalizationMethod>
3768         <ds:SignatureMethod
3769 Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
3770         <ds:Reference URI="cid:${REPOSITORY_ITEM1_ID}">
3771         <ds:Transforms>
3772         <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
3773 c14n#">
3774         <ds:Transform Algorithm="http://docs.oasis-
3775 open.org/wss/2004/XX/oasis-2004XX-wss-swa-profile-1.0#Attachment-Content-
3776 Only-Transform" />
3777         </ds:Transform>
3778         </ds:Transforms>
3779         <ds:DigestMethod
3780 Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
3781         <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
3782         </ds:Reference>
3783         </ds:SignedInfo>
3784
3785         <ds:SignatureValue>PipXJ2Sfc+LTDnq4pM5JcIYt9gg=</ds:SignatureValu
3786 e>
3787
3788         <ds:KeyInfo>
3789         <wsse:SecurityTokenReference>
3790         <wsse:Reference URI="#urn:oasis:names:tc:ebxml-
3791 regrep:rs:security:SenderCert" ValueType="http://docs.oasis-
3792 open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" />
3793         </wsse:SecurityTokenReference>
3794         </ds:KeyInfo>
3795
3796         </ds:Signature>
3797     </wsse:Security>
3798 </soap:Header>
3799 <soap:Body wsu:Id="TheBody">
3800     <lcm:SubmitObjectsRequest/>
3801 </soap:Body>

```



```
3802 </soap:Envelope>
3803 --BoundaryStr
3804 Content-Type: image/png
3805 Content-ID: <${REPOSITORY_ITEM1_ID}>
3806 Content-Transfer-Encoding: base64
3807 the repository item (e.g. PNG Image) goes here..
```

Listing 4: RepositoryItem Signature Example

3809 **10.3.2.7 SOAP Message Security and HTTP/S**

3810 When using HTTP/S between a Registry Client and a registry, SOAP message security MUST NOT be
3811 used. Specifically:

- 3812 • The Registry Client MUST NOT sign the request message or any repository items in the request.
- 3813 • The registry MUST NOT verify request or RepositoryItem signatures.
- 3814 • The registry MUST NOT sign the response message or any repository items in the response.
- 3815 • The Registry Client MUST NOT verify response or RepositoryItem signatures.

3816 **10.3.3 Message Confidentiality**

3817 A registry SHOULD support encryption of protocol messages as defined section 9 of [WSI-BSP] as a
3818 mechanism to support confidentiality of protocol messages during transmission on the wire.

3819 A Registry Client MAY use encryption of RepositoryItems as defined by [WSS-SWA] as a mechanism to
3820 support confidentiality of RepositoryItems during transmission on the wire.

3821 A registry SHOULD support the submission of encrypted repository items.

3822 **10.3.4 Key Distribution Requirements**

3823 The registry and Registered Users MUST mutually exchange their public keys. This is necessary to enable:

- 3824 • Mutual Authentication of Registry Client and registry using SSL/TLS handshake for transport layer
3825 security over HTTP/S
- 3826 • Validation of Message Signature and RepositoryItem Signature (described in section).
- 3827 • Decryption of encrypted messages

3828 In order to enable Message Security the following requirements MUST be met:

- 3829 1. A Certificate is associated with the registry.
- 3830 2. A Certificate is associated with Registry Client.
- 3831 3. A Registry Client registers its public key certificate with the registry. This is typically done during User
3832 Registration and is implementation specific.
- 3833 4. Registry Client obtains the registry's public key certificate and stores it in its own local key store. This is
3834 done in an implementation specific manner.

3835

3836 **10.4 Authentication**

3837 The Registry MUST be able to authenticate the identity of the User associated with client requests in order
3838 to perform authorization and access control and to maintain an Audit Trail of registry access. In security
3839 terms a service that provides the ability to authenticate requestors is referred to as an Authentication
3840 Authority.

3841 A registry MUST provide one or more of the following Authentication mechanisms:

- 3842 • Registry as Authentication Authority
- 3843 • External Authentication Authority

3844

3845 **10.4.1 Registry as Authentication Authority**

3846 A registry MAY provide authentication capability by serving as an Authentication Authority. In this role the
3847 registry uses the <ds:KeyInfo> in the Message Signature as credentials to authenticate the requestor. This
3848 typically requires checking that the public key supplied in the <ds:KeyInfo> of the Message Signature
3849 matches the public key of a Registered User. This also requires that the registry maintain a “registry
3850 keystore” that contains the public keys of Registered Users. The remaining details of registry as an
3851 authentication authority are implementation specific.

3852 Alternatively, if the Registry Client communicates with the registry over HTTP/S, the registry MUST
3853 authenticate the Registry Client User if a registered certificate is provided through SSL Client Authentication.
3854 If the certificate is not known to the registry then the Registry MUST assign the RegistryGuest principal with
3855 the Registry Client.

3856 **10.4.2 External Authentication Authority**

3857 A registry MAY also use an external Authentication Authority to authenticate client requests. The use of an
3858 external Authentication Authority requires that the registry implement the Registry SAML Profile as described
3859 in chapter 11.

3860 **10.4.3 Authenticated Session Support**

3861 Once a request is authenticated a Registry SHOULD establish an authenticated session using
3862 implementation specific means to avoid having to re-authenticate subsequent request from the same
3863 requestor. When the underlying transport protocol is HTTP, a registry SHOULD implement authenticated
3864 session support based upon HTTP session capability as defined by [RFC2965].

3865 **10.5 Authorization and Access Control**

3866 Once a registry has authenticated the identity of the Registered User associated with a client request it
3867 MUST perform authorization and subsequently enforce access control rules based upon the authorization
3868 decision.

3869 Authorization and access control is an operation conducted by the registry that decides WHO can do WHAT
3870 ACTION on WHICH RESOURCE.

- 3871 • The WHO is the User determined by the authentication step.
- 3872 • The WHAT ACTION is determined by the registry protocol request sent by the client.
- 3873 • The WHICH RESOURCE consists of the RegistryObjects and RepositoryItems impacted by the
3874 registry protocol request.

3875 The Access Control Policy associated with the resource that is impacted by the action determines
3876 authorization and access control.

3877 A registry MUST provide an access control and authorization mechanism based upon chapter titled “Access
3878 Control Information Model” in [ebRIM]. This model defines a default access control policy that MUST be
3879 supported by the registry. In addition it also defines a binding to [XACML] that allows fine-grained access
3880 control policies to be defined.

3881 **10.6 Audit Trail**

3882 Once a registry has performed authorization checks, enforced access control and allowed a client request to
3883 proceed it services the client request. A registry MUST create an Audit Trail of all LifecycleManager
3884 operations. A registry MAY create an Audit Trail of QueryManager operations. To conserve storage
3885 resources, a registry MAY prune the Audit Trail information it stores in an implementation specific manner. A
3886 registry SHOULD perform such pruning by removing the older information in its Audit Trail content. However,
3887 it MUST not remove the original Create Event at the beginning of the audit trail since the Create Event
3888 establishes the owner of the RegistryObject.

3889 Details of how a registry maintains an Audit Trail of client requests is described in the chapter title “Event
3890 Information Model” of [ebRIM].

3891

11 Registry SAML Profile

3892

3893

3894

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].

3895

11.1 Terminology

3896

3897

3898

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.

3899

3900

11.2 Use Cases for SAML Profile

3901

3902

The Registry SAML Profile is intended to address following use cases using the protocols defined by [SAMLCore].

3903

11.2.1 Registry as SSO Participant:

3904

3905

3906

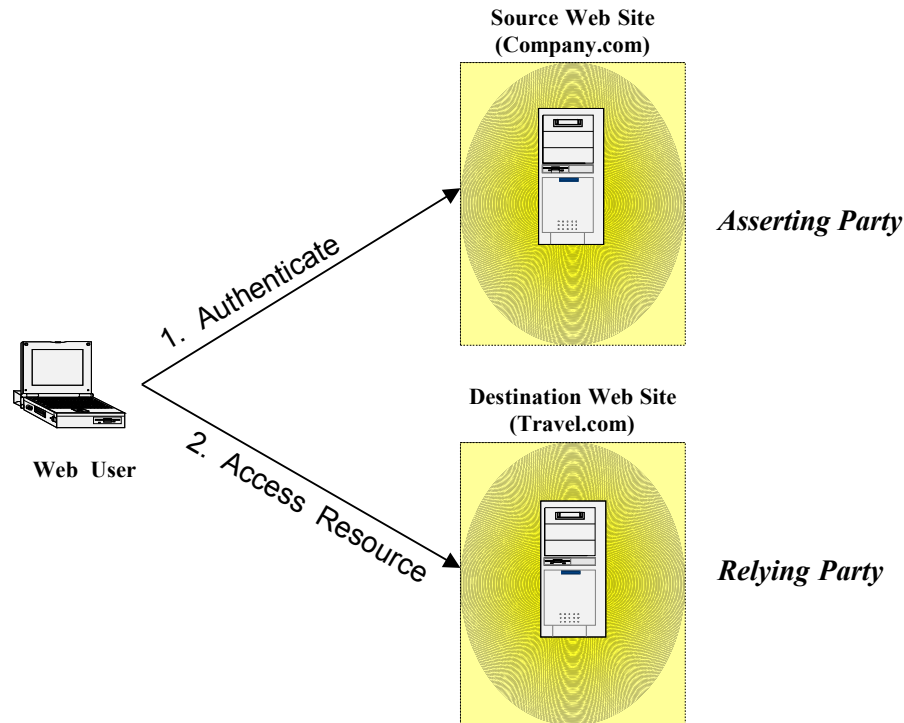
3907

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 enterprise also has an existing Authentication Authority (which may be the same service as the Identity Provider) that is used to authenticate users and enable Single Sign On (SSO) across all their enterprise services

3908 applications.

3909 The enterprise wishes to use its existing Identity Provider to manage registry users and to avoid duplicating
3910 the user database contained in the Identity Provider within the registry. The enterprise also wishes to use its
3911 existing Authentication Authority to authenticate registry users and expects the registry to participate in SSO
3912 capability provided by their Authentication Authority service.

3913



3914
3915

Figure 27: SAML SSO Typical Scenario

3916 11.3 SAML Roles Played By Registry

3917 In order to conform to the registry SAML Profile an ebXML Registry plays the Service Provider (SP) role
3918 based upon conformance with SAML 2.0 protocols.

3919 11.3.1 Service Provider Role

3920 The Service Provider role enables the registry to participate in SAML protocols. Specifically it allows the
3921 registry to utilize an Identity Provider to perform client authentication on its behalf.

3922 11.3.1.1 Service Provider Requirements

3923 The following are a list of requirements for the Service Provider role of the registry:

- 3924 • MUST support the protocols, messages and bindings that are the responsibility of the Service
3925 Provider as defined by Web SSO Profile in [SAMLProf]. Specifically it MUST be able to initiate and
3926 participate in the Authentication Request Protocol with an Identity Provider.
- 3927 • MUST be able to use a SAML Identity Provider to authenticate client requests.
- 3928 • MUST support the ability to maintain a security context for registry clients across multiple client
3929 requests.

3930

3931 **11.4 Registry SAML Interface**

3932 In order to conform to the registry SAML Profile an ebXML Registry MUST implement a new SAML interface
 3933 in addition to its service interfaces such as QueryManager and LifeCycleManager.

3934 Details of the registry's SAML interface are not described by this specification. Instead they are described by
 3935 the SAML 2.0 specifications and MUST support SAML HTTP and SOAP requests.

3936 A registry uses its SAML interface to participate in SAML protocols with SAML Clients and SAML Identity
 3937 Providers. Specifically, an IdentityProvider uses the registry's SAML Service Provider interface to deliver the
 3938 Response to an Authentication Request.

3939 **11.5 Requirements for Registry SAML Profile**

3940 In order to conform to the Registry SAML Profile a registry MUST implement specific SAML protocol that
 3941 support specific SAML protocol message exchanges using specific protocol bindings.

3942 Table 7 lists the matrix of SAML Profiles, Protocols Messages and their Bindings that a registry MUST
 3943 support in order to conform to the registry SAML Profile.

3944 The reader should refer to:

- 3945 • [SAMLProf] for description of profiles listed
- 3946 • [SAMLCore] for description of Message Flows listed
- 3947 • [SAMLBind] for description of Bindings listed

3948

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

3949

3950

Table 7: Required SAML Profiles, Protocols and Bindings

3951 **11.6 SSO Operation**

3952 This section describes the interaction sequence for various types of SSO operations.

3953 **11.6.1 Scenario Actors**

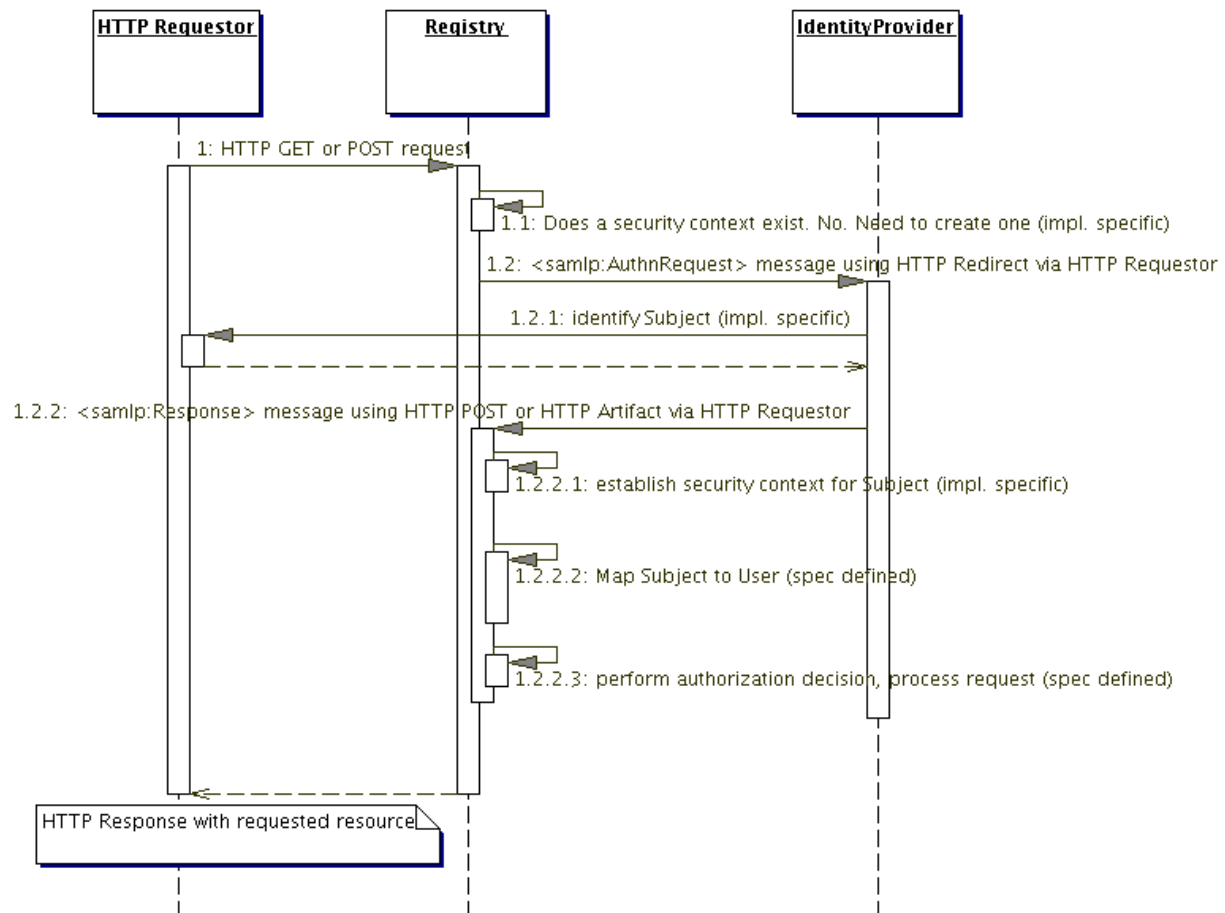
3954 The following are the actors that will be participating the various SSO Operation scenarios described in

3955 subsequent section:

- 3956 • HTTP Requestor: This represents a Registry Client that accesses the registry using the HTTP
3957 binding of the registry protocols typically through a User Agent such as a Web Browser.
- 3958 • SOAP Requestor: This represents a Registry Client that accesses the registry using the SOAP
3959 binding of the registry protocols.
- 3960 • Registry: This represents a Registry and includes all Registry interfaces such as QueryManager,
3961 LifeCycleManager and the registry's SAML Service Provider. The Registry participates in ebXML
3962 Registry protocols as well as SAML protocols.
- 3963 • IdentityProvider: This represents the IdentityProvider used by the registry to perform Authentication
3964 on its behalf.

3965 11.6.2 SSO Operation – Unauthenticated HTTP Requestor

3966 Figure 28 shows a high level view of the Single Sign On (SSO) operation when the SOAP Requestor is
3967 unauthenticated and accesses the registry over HTTP via a User Agent such as a Web Browser.



3968
3969

Figure 28: SSO Operation – Unauthenticated HTTP Requestor

3970 11.6.2.1 Scenario Sequence

3971 Figure 28 shows the following sequence of steps for the operation:

- 3972 1 The HTTP Requestor sends a HTTP GET or POST request to a Registry interface such as the
3973 QueryManager or LifeCycleManager.
- 3974 1.1 The Registry checks to see if it already has a security context established for the Subject associated
3975 with the request. It determines that there is no pre-existing security context.
- 3976 1.2 In order to establish a security context, the Registry therefor initiates the <samlp:AuthnRequest>
3977 protocol with the IdentityProvider. The <AuthnRequest> is sent using HTTP Redirect via the User
3978 Agent (e.g. Web Browser) used by the HTTP Requestor.
- 3979 1.2.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this
3980 requires communicating with the User Agent being used by the HTTP Requestor to get the
3981 credentials associated with the Subject and then using the credentials to authenticate that the
3982 IdentityProvider knows the Subject. In case of SSL/TLS based communication the credetials are
3983 acquired without any user intervention directly from the User Agent. The figure assumes that the
3984 IdentityProvider is able to authenticate the Subject.
- 3985 1.2.2 The IdentityProvider sends a <samlp:Response> message containing a
3986 <saml:AuthenticationStatement> to the Registry using either HTTP POST or HTTP Artifact SAML
3987 Binding via the User Agent.
- 3988 1.2.2.1 The Registry uses implementation specific means to establish a security context for the Subject
3989 authenticated by the IdentityProvider based upon the information contained about the Subject in
3990 the <samlp:Response> message. This may include creating an HTTP Session for the HTTP
3991 Requestor.
- 3992 1.2.2.2 The Registry maps the information about the Subject in the <samlp:Response> message into a
3993 <rim:User> instance. This establishes the <rim:User>context for the security context.
- 3994 1.2.2.3 The Registry then performs authorization decision based upon the original HTTP request and the
3995 <rim:User>. The figure assumes that authorization decision was to allow the request to be
3996 processed. The Registry processes the request and subsequently return the requested resource
3997 to the HTTP Requestor via the HTTP response.
3998

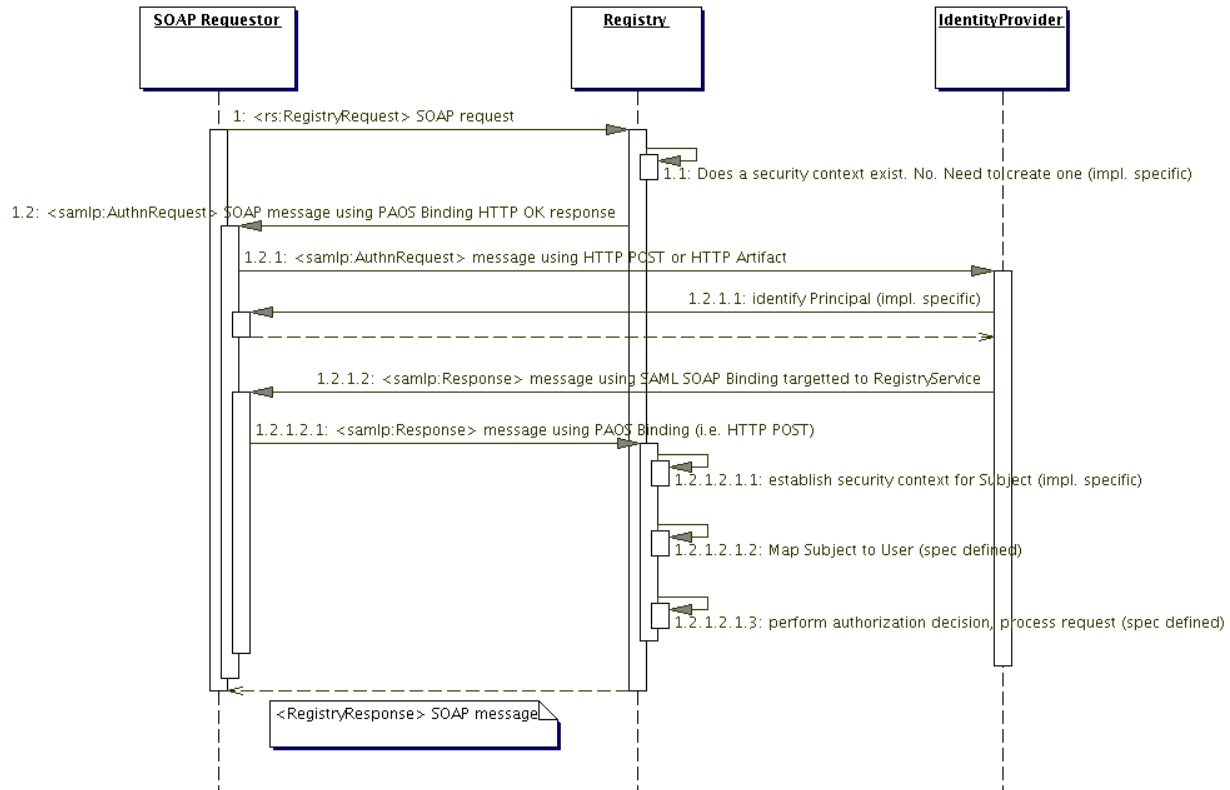
3999 **11.6.3 SSO Operation – Authenticated HTTP Requestor**

4000 This is the case where the HTTP Requestor first authenticates with an IdentityProvider and then accesses
4001 the registry over HTTP via a User Agent such as a Web Browser.

4002 Currently there are no standard means defined for carrying SAML Assertions resulting from the Registry
4003 Requestor authenticating with an IdentityProvider over HTTP protocol to a Service Provider such as the
4004 registry. A registry MAY support this scenario in an implementation specific manner. Typically, the Identity
4005 Provider will define any such implementation specific manner.

4006 **11.6.4 SSO Operation – Unauthenticated SOAP Requestor**

4007 This is the case where an unauthenticated Registry Requestor accesses the registry over SOAP.
4008 Figure 29 shows the steps involved.



4009
4010

Figure 29: SSO Operation - Unauthenticated SOAP Requestor

4011 **11.6.4.1 Scenario Sequence**

4012 Figure 29 shows the following sequence of steps for the operation:

- 4013 1 The SOAP Requestor sends a <rs:RegistryRequest> SOAP message such as a
4014 <lc:SubmitObjectsRequest> to a Registry interface such as the LifeCycleManagerManager. In the
4015 request header the SOAP Requestor declares that it is an ECP requestor as defined by the ECP Profile
4016 in [SAMLProf].
- 4017 1.1 The Registry checks to see if it already has a security context established for the Subject associated
4018 with the request. It determines that there is no pre-existing security context.
- 4019 1.2 Because the request is from an ECP client, the registry uses the ECP Profile defined by [SAMLProf]
4020 and sends a <samlp:AuthnRequest> SOAP message as response to the <rs:RegistryRequest>
4021 SOAP message to the SOAP Requestor using the PAOS Binding as defined by [SAMLBind]. The
4022 response has an HTTP Response status of OK.
- 4023 1.2.1 The SOAP Requestor then initiates the <samlp:AuthnRequest> protocol with the IdentityProvider.
4024 The <samlp:AuthnRequest> is sent using HTTP POST or Artifact Binding directly to the
4025 IdentityProvider.
- 4026 1.2.1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this
4027 requires communicating with the SOAP Requestor to get the credentials associated with the
4028 Subject and then using the credentials to authenticate that the IdentityProvider knows the Subject.
4029 In case of SSL/TLS based communication the credentials are acquired without any user
4030 intervention directly from the SOAP Requestor. The figure assumes that the IdentityProvider is
4031 able to authenticate the Subject.
- 4032 1.2.1.2 The IdentityProvider sends a <samlp:Response> message containing a
4033 <saml:AuthenticationStatement> to the SOAP Requestor using SAML SOAP Binding. The HTTP

4034 header specifies the Registry as the ultimate target of the response.

4035 1.2.1.2.1 The SOAP Requestor forwards the <saml:Response> message containing a
4036 <saml:AuthenticationStatement> to the Registry using PAOS Binding via HTTP POST.

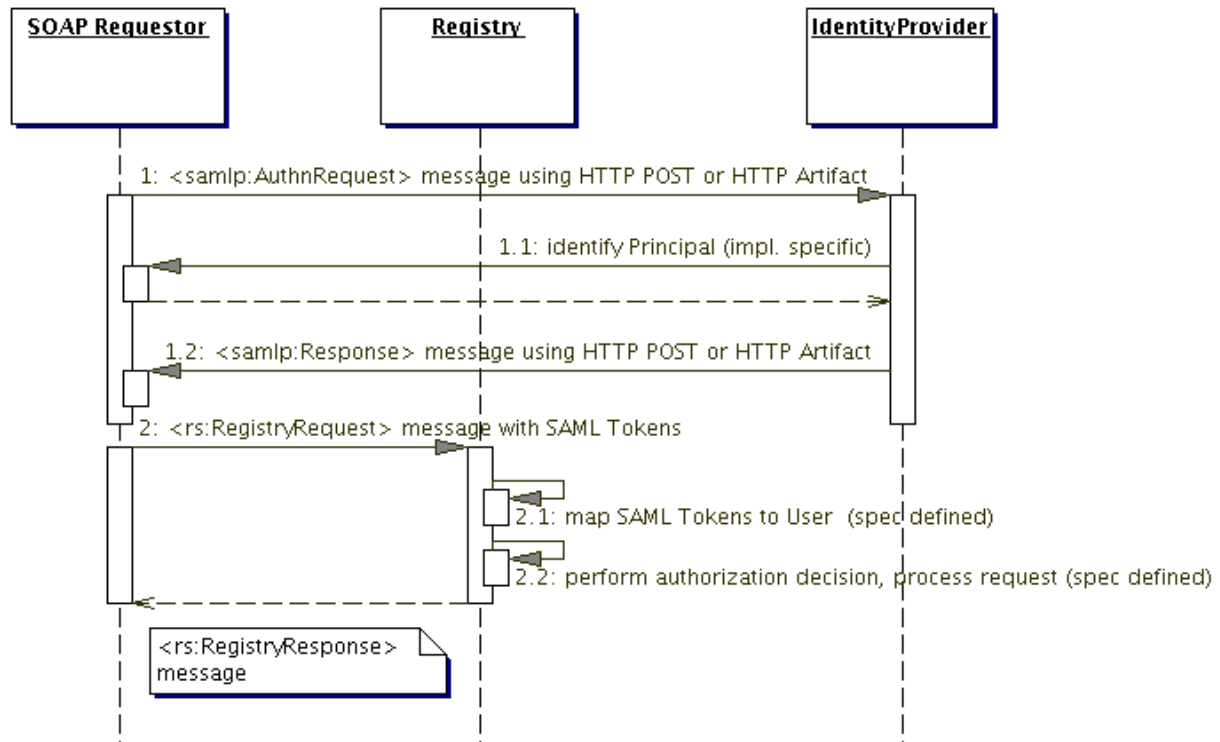
4037 1.2.1.2.1.1 The Registry uses implementation specific means to establish a security context for the
4038 Subject authenticated by the IdentityProvider based upon the information contained about the
4039 Subject in the <samlp:Response> message. This may include creating an HTTP Session for
4040 the HTTP Requestor.

4041 1.2.1.2.1.2 The Registry maps the information about the Subject in the <samlp:Response> message into
4042 a <rim:User> instance. This establishes the <rim:User>context for the security context.

4043 1.2.1.2.1.3 The Registry then performs authorization decision based upon the original SOAP request and
4044 the <rim:User>. The figure assumes that authorization decision was to allow the request to be
4045 processed. The Registry processes the request and subsequently return a
4046 <rs:RegistryResponse> SOAP message as response to the original <rs:RegistryRequest>
4047 SOAP request.
4048

4049 **11.6.5 SSO Operation – Authenticated SOAP Requestor**

4050 This is the case where the Registry Requestor first authenticates with an IdentityProvider directly and then
4051 makes a request to the registry using SOAP.



4052
4053

Figure 30: SSO Operation - Authenticated SOAP Requestor

4054 **11.6.5.1 Scenario Sequence**

4055 The figure shows the following sequence of steps for the operation:

- 4056 1 The SOAP Requestor then initiates the <samlp:AuthnRequest> protocol directly with the
4057 IdentityProvider. The <samlp:AuthnRequest> is sent using HTTP POST or Artifact Binding.
- 4058 1.1 The IdentityProvider uses implementation specific means to identify the Subject. Typically this
4059 requires communicating with the SOAP Requestor to get the credentials associated with the Subject
4060 and then using the credentials to authenticate that the IdentityProvider knows the Subject. In case of
4061 SSL/TLS based communication the credentials are acquired without any user intervention directly from
4062 the SOAP Requestor. The figure assumes that the IdentityProvider is able to authenticate the
4063 Subject.
- 4064 1.2 The IdentityProvider sends a <samlp:Response> message containing a
4065 <saml:AuthenticationStatement> to the SOAP Requestor using SAML HTTP POST or HTTP Artifact
4066 Binding.
- 4067 2 The SOAP Requestor sends a <rs:RegistryRequest> SOAP message such as a
4068 <lcm:SubmitObjectsRequest> to a Registry interface such as the LifeCycleManagerManager. The

4069 <rs:RegistryRequest> SOAP message includes SAML Tokens in the <soap:Header> of the SOAP
4070 message as defined by [WSS-SAML]. The SAML Tokens are based upon the <saml:Response>
4071 during authentication.

4072 2.1 The registry maps the SAML Tokens from the <soap:Header> of the <rs:RegistryRequest> to a
4073 <rim:User> instance. This establishes the <rim:User> context for the request.

4074 2.2 The Registry then performs authorization decision based upon the original SOAP request and the
4075 <rim:User>. The figure assumes that authorization decision was to allow the request to be processed.
4076 The Registry processes the request and subsequently return a <rs:RegistryResponse> SOAP
4077 message as response to the original <rs:RegistryRequest> SOAP request.
4078

4079 **11.6.6 <saml:AuthnRequest> Generation Rules**

4080 The following rules MUST be observed when the registry or Registry Client issues a
4081 <saml:AuthnRequest>:
4082

- 4083 • A registry MUST specify a NameIDPolicy within the <saml:AuthRequest>
- 4084 • The Format of the NameIDPolicy MUST be urn:oasis:names:tc:SAML:2.0:nameid-format:persistent
4085 as defined by section in [SAMLCore]. Note that it is the Persistent Identifier that maps to the id
4086 attribute of <rim:User>.

4087 —

4088 **11.6.7 <saml:Response> Processing Rules**

4089 This section describes how the registry processes the <saml:Response> to a <saml:AuthnRequest>:

4090 **<saml:Response> Processing**

- 4091 • Response Processing: The registry MUST verify the <ds:Signature> for the <saml:Response> if
4092 present.
- 4093 • The registry MUST check the <saml:Status> associated with <saml:Response> for errors. If the
4094 <saml:Status> has a top level <saml:StatusCode> whose value is NOT
4095 urn:oasis:names:tc:SAML:2.0:status:Success then the registry MUST throw an
4096 AuthenticationException. The AuthenticationException message SHOULD include the information
4097 from the StatusCode, StatusMessage and StatusDetail from the <saml:Status>.

4098 **<saml:Assertion> Processing**

- 4099 • The registry SHOULD check the <saml:Assertion> for Conditions and honour any standard
4100 Conditions defined by [SAMLCore] if any are specified.

4101 **<saml:AuthnStatement> Processing**

- 4102 • The registry MUST check the SessionNotOnOrAfter attribute of the <saml:AuthnStatement> for
4103 validity of the authenticated session.

4104 **<saml:Subject> Processing**

- 4105 • A registry MUST map the <saml:Subject> to a <rim:User> instance as described in 11.6.8.

4106 **11.6.8 Mapping Subject to User**

4107 As required by [SAMLCore] a <saml:Response> to a <saml:AuthnRequest> MUST contain a
4108 <saml:Subject> that identifies the Subject that was authenticated by the IdentityProvider. In addition it MUST
4109 contain a <saml:AuthnStatement> which asserts that the IdentityProvider indeed authenticated the
4110 Subject.

4111 The following table defines the mapping between a <saml:Subject> and a <rim:User>:

4112

– Subject Attribute	– User Attribute	– Description
– NameID content	– id attribute	NameID Format MUST be “urn:oasis:names:tc:SAML:1.1:nameid-format:persistent”

4113

Table 8: Mapping Subject to User

4114

Note that any attribute of Subject not specified above SHOULD be ignored when mapping Subject to User.

4115

Note that any attribute of User not specified above MUST be left unspecified when mapping Subject to User.

4116

11.7 External Users

4117

The SAML Profile allows registry Users to be registered in an Identity Provider external to the registry.

4118

These are referred to as “External Users”. A registry dynamically creates such External Users by mapping a

4119

SAML Subject to a User instance dynamically.

4120

The following are some restrictions on External User instances:

4121

- External User instances are transient from the registry’s perspective and MUST not be stored within the registry as User instances

4122

4123

- A RegistryObject MUST not have a reference to an External User unless it is composed within that RegistryObject. Composed RegistryObjects such as Classification instances are allowed to reference their parent External User instance.

4124

4125

4126

- Since External User instances are transient they MUST not match a registry Query.

4127

4128

4129

4130

4131

4132 **12 Native Language Support (NLS)**

4133 This chapter describes the Native Languages Support (NLS) features of ebXML Registry.

4134 **12.1 Terminology**

4135 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].

4136

4137 **12.2 NLS and Registry Protocol Messages**

4138 For the accurate processing of data in both registry client and registry services, it is essential for the
4139 recipient of a protocol message to know the character set being used by it.

4140 A Registry Client SHOULD specify charset parameter in MIME header when they specify text/xml as
4141 Content-Type. A registry MUST specify charset parameter in MIME header when they specify text/xml as
4142 Content-Type.

4143 The following is an example of specifying the character set in the MIME header.

4144

```
4145 Content-Type: text/xml; charset=ISO-2022-JP
```

4146

4147

4148 If a registry receives a protocol message with the charset parameter omitted then it MUST use the default
4149 charset value of "us-ascii" as defined in [RFC 3023].

4150 Also, when an application/xml entity is used, the charset parameter is optional, and registry client and
4151 registry services MUST follow the requirements in Section 4.3.3 of [REC-XML] which directly address this
4152 contingency.

4153 If another Content-Type is used, then usage of charset MUST follow [RFC 3023].

4154 **12.3 NLS Support in RegistryObjects**

4155 The information model XML Schema [RR-RIM-XSD] defines the <rim:InternationalStringType> for defining
4156 elements that contains a locale sensitive string value.

4157

```
4158 <complexType name="InternationalStringType">  
4159 <sequence maxOccurs="unbounded" minOccurs="0">  
4160 <element ref="tns:LocalizedString"/>
```

```
4161     </sequence>
4162 </complexType>
```

4163

4164 An InternationalStringType may contain zero or more LocalizedStrings within it where each LocalizedString
4165 contain a string value is a specified local language and character set.

4166

```
4167 <complexType name="LocalizedStringType">
4168   <attribute ref="xml:lang" default="en-US"/>
4169   <attribute default="UTF-8" name="charset"/>
4170   <attribute name="value" type="tns:FreeFormText" use="required"/>
4171 </complexType>
```

4172

4173 Examples of such attributes are the "name" and "description" attributes of the RegistryObject class defined
4174 by [ebRIM] as shown below.

```
4175   <complexType name="InternationalStringType">
4176     <sequence maxOccurs="unbounded" minOccurs="0">
4177       <element ref="tns:LocalizedString"/>
4178     </sequence>
4179   </complexType>
4180   <element name="InternationalString"
4181 type="tns:InternationalStringType"/>
4182   <element name="Name" type="tns:InternationalStringType"/>
4183   <element name="Description" type="tns:InternationalStringType"/>
4184
4185   <complexType name="LocalizedStringType">
4186     <attribute ref="xml:lang" default="en-US"/>
4187     <!--attribute name = "lang" default = "en-US" form = "qualified" type
4188 = "language"/-->
4189     <attribute default="UTF-8" name="charset"/>
4190     <attribute name="value" type="tns:FreeFormText" use="required"/>
4191   </complexType>
4192   <element name="LocalizedString" type="tns:LocalizedStringType"/>
```

4193

4194 An element InternationalString is capable of supporting multiple locales within its collection of
4195 LocalizedStrings.

4196 The above schema allows a single RegistryObject instance to include values for any NLS sensitive element
4197 in multiple locales.

4198 The following example illustrates how a single RegistryObject can contain NLS sensitive <rim:Name> and
4199 "<rim:Description>" elements with their value specified in multiple locales. Note that the <rim:Name> and
4200 <rim:Description> use the <rim:InternationalStringType> as their type.

```
4201   <rim:ExtrinsicObject id="{ID}" mimeType="text/xml">
4202     <rim:Name>
4203       <rim:LocalizedString xml:lang="en-US" value="customACP1.xml"/>
4204       <rim:LocalizedString xml:lang="fi-FI" value="customACP1.xml"/>
4205       <rim:LocalizedString xml:lang="pt-BR" value="customACP1.xml"/>
4206     </rim:Name>
4207     <rim:Description>
4208       <rim:LocalizedString xml:lang="en-US" value="A sample custom
4209 ACP"/>
4210       <rim:LocalizedString xml:lang="fi-FI" value="Esimerkki custom
4211 ACP"/>
4212       <rim:LocalizedString xml:lang="pt-BR" value="Exemplo de ACP
4213 customizado
4214 "/>
4215     </rim:Description>
4216   </rim:ExtrinsicObject>
```

4217

4218 Since locale information is specified at the sub-element level there is no language or character set
4219 associated with a specific RegistryObject instance.

4220 **12.3.1 Character Set of *LocalizedString***

4221 The character set used by a locale specific String (*LocalizedString*) is defined by the charset attribute.
4222 Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of *LocalizedStrings*
4223 for maximum interoperability.

4224 **12.3.2 Language of *LocalizedString***

4225 The language MAY be specified in xml:lang attribute (Section 2.12 [REC-XML]).

4226 **12.4 NLS and Repository Items**

4227 While a single instance of an *ExtrinsicObject* is capable of supporting multiple locales, it is always
4228 associated with a single repository item. The repository item MAY be in a single locale or MAY be in multiple
4229 locales. This specification does not specify any NLS requirements for repository items.

4230 **12.4.1 Character Set of Repository Items**

4231 When a submitter submits a repository item, they MAY specify the character set used by the repository
4232 item using the MIME *Content-Type* mime header for the mime multipart containing the repository item as
4233 shown below:

4234

```
4235 Content-Type: text/xml; charset="UTF-8"
```

4236

4237

4238 Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of *LocalizedStrings*
4239 for maximum interoperability. A registry MUST preserve the charset of a repository item as it is originally
4240 specified when it is submitted to the registry.

4241 **12.4.2 Language of Repository Items**

4242 The Content-language mime header for the mime bodypart containing the repository item MAY specify the
4243 language for a locale specific repository item. The value of the Content-language mime header property
4244 MUST conform to [RFC 1766].

4245 This document currently specifies only the method of sending the information of character set and language,
4246 and how it is stored in a registry. However, the language information MAY be used as one of the query
4247 criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation procedure, like
4248 registry client is asking a favorite language for messages from registry services, could be another
4249 functionality for the future revision of this document.

4250 **13 Conformance**

4251 This chapter defines the technical conformance requirements for ebXML Registry. Note that it does not
4252 define specific conformance tests to verify compliance with various conformance profiles.

4253 **13.1 Conformance Profiles**

4254 An ebXML Registry **MUST** comply with one of the following conformance profiles:

- 4255 • Registry Lite – This conformance profile requires the registry to implement a minimal set of core features
4256 defined by this specification.
- 4257 • Registry Full – This conformance profile requires the registry to implement additional set of features in
4258 addition to those required by the Registry Lite conformance profile.

4259 **13.2 Feature Matrix**

4260 The following table identifies the implementation requirements for each feature defined by this specification
4261 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

4262

14 References

14.1 Normative References

- 4265 **[RFC2119]** S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC
4266 2119, March 1997, <http://www.ietf.org/rfc/rfc2119.txt>.
- 4267 **[ebRIM]** ebXML Registry Information Model Version 3.0.42
4268 <http://www.oasis-open.org/committees/regrep/documents/3.0.1/specs/regrep-rim-3.0.1-cs-042.pdf>
- 4270 **[REC-XML]** W3C Recommendation. Extensible Markup language(XML)1.0(Second Edition)
4271 <http://www.w3.org/TR/REC-xml>
- 4272 **[RFC 1766]** IETF (Internet Engineering Task Force). RFC 1766:
4273 Tags for the Identification of Languages, ed. H. Alvestrand. 1995.
4274 <http://www.cis.ohio-state.edu/htbin/rfc/rfc1766.html>
- 4275 **[RFC 2130]** IETF (Internet Engineering Task Force). RFC 2130
4276 The Report of the IAB Character Set Workshop held 29 February - 1 March, 1996
4277 <http://www.faqs.org/rfcs/rfc2130.html>
- 4278 **[RFC 2277]** IETF (Internet Engineering Task Force). RFC 2277:
4279 IETF policy on character sets and languages, ed. H. Alvestrand. 1998.
4280 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2277.html>
- 4281 **[RFC 2278]** IETF (Internet Engineering Task Force). RFC 2278:
4282 IANA Charset Registration Procedures, ed. N. Freed and J. Postel. 1998.
4283 <http://www.cis.ohio-state.edu/htbin/rfc/rfc2278.html>
- 4284 **[RFC2616]** IETF (Internet Engineering Task Force). RFC 2616:
4285 Fielding et al. *Hypertext Transfer Protocol -- HTTP/1.1*. 1999.
4286 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 4287 **[RFC2965]** IETF (Internet Engineering Task Force). RFC 2965:
4288 D. Kristol et al. *HTTP State Management Mechanism*. 2000.
4289 <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- 4290 **[RR-CMS-XSD]** ebXML Registry Content Management Services XML Schema
4291 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>
- 4292 **[RR-LCM-XSD]** ebXML Registry LifeCycleManager XML Schema
4293 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/lcm.xsd>
- 4294 **[RR-RIM-XSD]** ebXML Registry Information Model XML Schema
4295 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rim.xsd>
- 4296 **[RR-RS-XSD]** ebXML Registry Service Protocol XML Schema
4297 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/rs.xsd>
- 4298 **[RR-QM-XSD]** ebXML Registry QueryManager XML Schema
4299 <http://www.oasis-open.org/committees/regrep/documents/3.0/schema/query.xsd>
- 4300 **[SAMLBind]** S. Cantor et al., *Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-bindings-2.0-cd-03.
4301
4302
4303 <http://www.oasis-open.org/committees/security/>.
- 4304 Note: when this document is finalized, this URL will be updated.
- 4305 **[SAMLConform]** P. Mishra et al. *Conformance Requirements for the OASIS Security Assertion Markup Language (SAML) V2.0*. OASIS SSTC, September 2004. Document ID sstc-saml-conformance-2.0-cd-03.
4306
4307
4308 <http://www.oasis-open.org/committees/security/>.
- 4309 Note: when this document is finalized, this URL will be updated.
- 4310 **[SAMLCore]** S. Cantor et al., *Assertions and Protocols for the OASIS Security*

4311		<i>Assertion Markup Language (SAML) V2.0. OASIS SSTC, December 2004.</i>
4312		<i>Document ID sstc-saml-core-2.0-cd-03.</i>
4313		http://www.oasis-open.org/committees/security/ .
4314		Note: when this document is finalized, this URL will be updated.
4315	[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.
4316		
4317		
4318		http://www.oasis-open.org/committees/security/ .
4319		Note: when this document is finalized, this URL will be updated.
4320	[SAML-P-XSD]	S. Cantor et al., SAML protocols schema. OASIS SSTC, September 2004.
4321		Document ID sstc-saml-schema-protocol-2.0.
4322		http://www.oasis-open.org/committees/security/ .
4323		Note: when this document is finalized, this URL will be updated.
4324	[SAML-XSD]	S. Cantor et al., SAML assertions schema. OASIS SSTC, September 2004.
4325		Document ID sstc-saml-schema-assertion-2.0.
4326		http://www.oasis-open.org/committees/security/ .
4327		Note: when this document is finalized, this URL will be updated.
4328	[SOAP11]	W3C Note. Simple Object Access Protocol, May 2000 http://www.w3.org/TR/SOAP
4329	[SwA]	W3C Note: SOAP with Attachments, Dec 2000
4330		http://www.w3.org/TR/SOAP-attachments
4331	[SQL]	Structured Query Language (FIPS PUB 127-2)
4332		http://www.itl.nist.gov/fipspubs/fip127-2.htm
4333	[SQL/PSM]	Database Language SQL — Part 4: Persistent Stored Modules
4334		(SQL/PSM) [ISO/IEC 9075-4:1996]
4335	[UUID]	DCE 128 bit Universal Unique Identifier
4336		http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh_20
4337	[WSDL]	W3C Note. Web Services Description Language (WSDL) 1.1
4338		http://www.w3.org/TR/wsdl
4339	[XML]	T. Bray, et al. Extensible Markup Language (XML) 1.0 (Second Edition). World
4340		Wide Web Consortium, October 2000.
4341		http://www.w3.org/TR/REC-xml
4342	[XMLDSIG]	XML-Signature Syntax and Processing
4343		http://www.w3.org/TR/2001/PR-xmlsig-core-20010820/
4344	[WSI-BSP]	WS-I: Basic Security Profile 1.0
4345		http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html
4346		Note: when this document is finalized, this URL will be updated.
4347	[WSS-SMS]	Web Services Security: SOAP Message Security 1.0
4348		http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
4349		
4350	[WSS-SWA]	Web Services Security: SOAP Message with Attachments (SwA) Profile 1.0
4351		http://www.oasis-open.org/apps/org/workgroup/wss/download.php/10902/wss-swa-profile-1.0-cd-01.pdf
4352		
4353		Note: when this document is finalized, this URL will be updated.

4354 **14.2 Informative**

4355	[ebBPSS]	ebXML Business Process Specification Schema
4356		http://www.ebxml.org/specs
4357	[ebCPP]	ebXML Collaboration-Protocol Profile and Agreement Specification
4358		http://www.ebxml.org/specs/
4359	[ebMS]	ebXML Messaging Service Specification, Version 1.0

4360		http://www.ebxml.org/specs/
4361	[DeltaV]	Versioning Extension to WebDAV, IETF RFC 3253
4362		http://www.webdav.org/deltav/protocol/rfc3253.html
4363	[XPT]	XML Path Language (XPath) Version 1.0
4364		http://www.w3.org/TR/xpath
4365	[IANA]	IANA (Internet Assigned Numbers Authority).
4366		Official Names for Character Sets, ed. Keld Simonsen et al.
4367		http://www.iana.org/
4368	[RFC2392]	E. Levinson, Content-ID and Message-ID Uniform Resource Locators, IETF RFC 2392,
4369		http://www.ietf.org/rfc/rfc2392.txt
4370		
4371	[RFC 2828]	IETF (Internet Engineering Task Force). RFC 2828:
4372		Internet Security Glossary, ed. R. Shirey. May 2000.
4373		http://www.cis.ohio-state.edu/htbin/rfc/rfc2828.html
4374	[RFC 3023]	IETF (Internet Engineering Task Force). RFC 3023:
4375		XML Media Types, ed. M. Murata. 2001.
4376		ftp://ftp.isi.edu/in-notes/rfc3023.txt
4377	[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.
4378		
4379		http://www.oasis-open.org/committees/security/ .
4380		
4381	[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.
4382		
4383		http://www.oasis-open.org/committees/security/ .
4384		
4385	[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.
4386		
4387		http://www.oasis-open.org/committees/security/ .
4388		
4389	[SAMLTech]	J.Hughes et al., Technical Overview of the OASIS Security Assertion Markup Language (SAML)V2.0.
4390		
4391		http://www.oasis-open.org/committees/download.php/7874/sstc-saml-tech-overview-2.0-draft-01.pdf
4392		
4393	[UML]	Unified Modeling Language
4394		http://www.uml.org
4395		http://www.omg.org/cgi-bin/doc?formal/03-03-01
4396		

4397

A. Acknowledgments

4398

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.

4399

4400

- Finally, the editors wish to acknowledge the following people for their contributions of material used as input to the OASIS ebXML Registry specifications:

4401

4402

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

4403

•

4404

B. Notices

4405 OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might
4406 be claimed to pertain to the implementation or use of the technology described in this document or the
4407 extent to which any license under such rights might or might not be available; neither does it represent that it
4408 has made any effort to identify any such rights. Information on OASIS's procedures with respect to rights in
4409 OASIS specifications can be found at the OASIS website. Copies of claims of rights made available for
4410 publication and any assurances of licenses to be made available, or the result of an attempt made to obtain
4411 a general license or permission for the use of such proprietary rights by implementors or users of this
4412 specification, can be obtained from the OASIS Executive Director.

4413 OASIS invites any interested party to bring to its attention any copyrights, patents or patent applications, or
4414 other proprietary rights which may cover technology that may be required to implement this specification.
4415 Please address the information to the OASIS Executive Director.

4416 **Copyright © OASIS Open 2004. All Rights Reserved.**

4417 This document and translations of it may be copied and furnished to others, and derivative works that
4418 comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and
4419 distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and
4420 this paragraph are included on all such copies and derivative works. However, this document itself does not
4421 be modified in any way, such as by removing the copyright notice or references to OASIS, except as needed
4422 for the purpose of developing OASIS specifications, in which case the procedures for copyrights defined in
4423 the OASIS Intellectual Property Rights document must be followed, or as required to translate it into
4424 languages other than English.

4425 The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or
4426 assigns.

4427 This document and the information contained herein is provided on an "AS IS" basis and OASIS
4428 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY
4429 WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR
4430 ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.