



Creating A Single Global Electronic Market

1 **ebXML Registry And Repository** 2 **Registry Services Proposal**

3 **Working Draft 10/2/2000 12:35 PM**

4 **This version:**
5 RegistryServicesSpecification v0-8.doc
6

7 **Abstract**

8 This document is a draft proposal whose purpose is to solicit additional input and convey the
9 current state of the ebXML Registry Service recommendations.

10 This document defines the various Registry Services as interaction protocols and processes
11 between an ebXML capable party and the ebXML Registry. It is assumed that all interactions
12 between the party and the ebXML registry will be conducted using ebXML Messaging Service.

13 **Notational Conventions**

14 The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD",
15 "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be
16 interpreted as described in Key Words for Use in RFC's to Indicate Requirement Levels (RFC
17 2119).

18 **Status of this Document**

19 This document represents work in progress upon which no reliance should be made.

20 **Document Version History**

- 21 o Version 0.1: Initial version (not released)
- 22 o Version 0.2: Corrections based on internal review (not released)
- 23 o Version 0.3: Added DocumentManager, Removed SchemaManager and ProcessManager
- 24 o Version 0.4: Added support for Querying the Registry. Added DTDs for all documents
25 currently defined



- 26 o Version 0.5: Major changes based on feedback from team review resulted in a complete re-
27 write from previous version. Replaced DocumentManager with ObjectManager and
28 associated name changes (e.g. DocumentInfo replaced with ManagedObject). Factored out
29 information model aspects in separate document [3]. Added QueryManager. Removed TPA
30 Manager. Changed terminology where relevant to better align with [2]. Changed Registrar to
31 Registry, Registrant to RegistryClient. Completely redid the DTD definition based on
32 Repository Information Model spec [3].
- 33 o Version 0.6: Changes based on issue logged during review of v0.5.
- 34 o Version 0.7: Changed RequestErrorResponse with ebXMLError defined by draft TRP error
35 handling specification. Changed registerOrganization to registerParty. Changes made to
36 almost all DTDs with major changes in the area of classification. This was due to
37 simplifications in classification model. DTDs now allow for multiple object submission with
38 optional classifications defined. The new DTDs also allow for multiple object approval,
39 deprecation, and removal. Changed GetClassifiedObjectsRequest so that multiple
40 classifications may be specified.
- 41 o Version 0.8: Changes based on review dated 9/29/2000. Changed “managed object” to
42 “managed object content”. Provided asynchronous query support with method additions to
43 ObjectQueryManager and ObjectQueryManagerClient.
- 44



44 Table of Contents

45	1	Introduction	5
46	1.1	Purpose and Scope	5
47	1.1.1	Goals	5
48	1.2	Related ebXML Specifications	6
49	1.3	General Conventions	6
50	1.4	Guiding Principals.....	6
51	1.5	Specification Structure.....	6
52	2	Overview	8
53	2.1	Role of ebXML Registry	8
54	2.2	Use Cases for the Registry Services.....	8
55	2.2.1	Business Domain Workflow Use Cases	8
56	2.2.2	Parties Register With Registry.....	9
57	2.2.3	Schema Documents Are Submitted.....	9
58	2.2.4	Business Process Documents Are Submitted	9
59	2.2.5	TPA is Submitted	9
60	2.3	Interfaces Implemented By Registry Service	10
61	2.4	Interfaces Implemented By Clients of Registry Service.....	11
62	3	Registration Service.....	11
63	3.1	Interpretation of UML Diagrams Describing an ebXML Business Process	12
64	3.1.1	UML Class Diagram	12
65	3.1.2	UML Sequence Diagram	12
66	3.2	Interpretation of Bootstrap Registration Process Sequence Diagram.....	13
67	3.2.1	Service Interfaces Defined	13
68	3.2.2	The Actions Defined On Service Interfaces	13
69	3.2.3	Requests Defined For Action.....	13
70	3.2.3.1	Requests Messages Defined For Request	13
71	3.2.3.2	Responses Defined For Request	13
72	3.2.3.3	Exception Responses Defined For Request.....	14
73	3.2.4	Messages Defined For Requests and Responses	14
74	3.2.5	Registry Service Interface in TPA SPECIFICATION.....	14
75	3.2.6	RegistryClient Service Interface in TPA SPECIFICATION.....	15
76	4	Object Management Service.....	15
77	4.1	Life Cycle of a Managed Object.....	15
78	4.2	Object Attributes.....	16
79	4.3	The Submit Objects Protocol	16
80	4.3.1	ManagedObject.....	17
81	4.4	The Approve Objects Request	17
82	4.5	The Deprecate Objects Request	18
83	4.6	The Remove Objects Request.....	19
84	5	Object Query Management Service	19
85	5.1	Design Goals For Object Query Management Support.....	19
86	5.2	Browse and Drill Down Query Support	20
87	5.2.1	Get Root Classification Nodes Request.....	20
88	5.2.2	Get Classification Tree Request	21
89	5.2.3	Get Classified Objects Request.....	22
90	5.3	Ad Hoc Query Support.....	22
91	5.4	Keyword Search Based Query Support.....	22



92	5.5	Object Retrieval.....	23
93	6	References	23
94	2	Acknowledgments	23
95	Appendix A	Schemas and DTD Definitions	24
96	A.1	RequestAcceptedResponse Message DTD	25
97	A.2	ebXMLError Message DTD	25
98	A.3	ManagedObject DTD	26
99	A.4	RegisterPartyRequest Message DTD.....	27
100	A.5	SubmitObjectsRequest Message DTD	28
101	A.6	ApproveObjectsRequest Message DTD	29
102	A.7	DeprecateObjectsRequest Message DTD	29
103	A.8	RemoveObjectsRequest Message DTD	29
104	A.9	ClassificationNode DTD.....	30
105	A.10	GetRootClassificationNodesRequest Message DTD	31
106	A.11	GetRootClassificationNodesResponse Message DTD.....	32
107	A.12	GetClassificationTreeRequest Message DTD	32
108	A.13	GetClassificationTreeResponse Message DTD.....	32
109	A.14	GetClassifiedObjectsRequest Message DTD	32
110	A.15	GetClassifiedObjectsResponse Message DTD.....	33
111	Appendix B	TPA Between Registry Client And Registry	33
112	Appendix C	Example XML for Submitting a Classification Tree.....	33
113	Appendix D	Terminology Mapping.....	34
114	Appendix E	Open Issues.....	34

115 **Table of Figures**

116	Figure 1: Business Domain Workflow Use Cases	8
117	Figure 2: Bootstrap Registration Process Sequence Diagram	12
118	Figure 3: Life Cycle of a Managed Object.....	16
119	Figure 4: Submit Objects Sequence Diagram	17
120	Figure 5: Approve Objects Sequence Diagram	18
121	Figure 6: Deprecate Objects Sequence Diagram	18
122	Figure 7: Remove Objects Sequence Diagram	19
123	Figure 8: Get Root Classification Nodes Sequence Diagram	20
124	Figure 9: Get Root Classification Nodes Asynchronous Sequence Diagram	21
125	Figure 10: Get Classification Tree Sequence Diagram	21
126	Figure 11: Get Classification Tree Asynchronous Sequence Diagram.....	21
127	Figure 12: Get Classified Objects Sequence Diagram	22
128	Figure 13: Get Classified Objects Asynchronous Sequence Diagram	22
129		



130 Table of Tables

131 Table 1: Terminology Mapping Table

34

132 1 Introduction

133 This document defines the ebXML Registry Services as a set of specialized business processes.
134 Clients of the ebXML Registry are referred to as RegistryClients while the ebXML Registry itself
135 is referred to as Registry.

136 [Note] For interoperability reasons it is required
137 that a Registry implementation *must* support its
138 service interfaces using the ebXML Messaging
139 Service bindings defined in this document. A
140 Registry implementation is free to provide any
141 other technology bindings (e.g. LDAP) as a non-
142 interoperable implementation specific detail.

143 All interaction between RegistryClients and the Registry are treated as if they are B2B
144 interactions between trading partners. Thus the processes supported by the Registry are
145 described in terms of:

- 146 o A special TPA between the Registry and RegistryClient
- 147 o A set of business processes involving the Registry and RegistryClient
- 148 o A set of business messages that are exchanged between the Registry and
149 RegistryClient as part of a specific Registry business process

150 1.1 Purpose and Scope

151 This document provides sufficient detail to develop:

- 152 o The specified functionality of the ebXML Registry Services
- 153 o ebXML based applications that utilize the Registry Services functionality specified in this
154 document

155 Software practitioners MAY use this document in combination with other ebXML specification
156 documents when creating ebXML compliant software.

157 **NOTE: This version describes only some of the Registry Services that are likely to be**
158 **needed by the proposed Tokyo POC scenarios. Several Registry Services identified in [2]**
159 **are currently unspecified and will be added in a later version. Some of the missing**
160 **functionality includes:**

- 161 o **Security (assuming it will be based upon the security specification that will be**
162 **included in a future version of ebXML Messaging Service Specification)**
- 163 o **Transformation Service**
- 164 o **Workflow Service**
- 165 o **Quality Assurance Service.**



166 1.1.1 Goals

167 The goals of this version of the specification are to:

- 168 o Communicate functionality of registry services to software developers
- 169 o Meet the immediate requirements of the Tokyo POC demo scenarios
- 170 o Able to evolve in future to support more complete ebXML Registry and Repository
- 171 requirements
- 172 o Be compatible with other ebXML specifications

173 1.2 Related ebXML Specifications

174 The following set of related specifications may be of interest to the reader:

- 175 a) Registry and Repository Part 1: Business Domain Model [2]
- 176 b) ebXML Repository Information Model [3]
- 177 c) Messaging Service Specification [4]
- 178 d) Business Process Meta Model Specification [5]
- 179 e) Trading-Partner Specification [7]

180 1.3 General Conventions

- 181 o *“managed object content”* is used to refer to actual repository content (not meta data)
- 182 instance (e.g. a DTD)
- 183 o *“ManagedObject”* is used to refer to an object that provides meta data about content instance
- 184 (managed object content).
- 185 o The information model *does not* contain *any* elements that are the actual content of the
- 186 repository (managed object content). All elements of the information model represent meta
- 187 data about the content and not the content itself.
- 188 o UML diagrams are used as a way to concisely describe business processes, collaboration
- 189 and concepts. They are not intended to convey any specific implementation requirements.
- 190 o The Registry Service processes are described as UML diagrams. The intent is to describe
- 191 business processes in a concise manner in terms of ebXML TRP. The reader must acquaint
- 192 themselves with section 3.1 (abstract) and section 3.2 (concrete example), in order to
- 193 understand these processes based on their UML description.

194 1.4 Guiding Principals

195 The following principals guided the work represented by this document:

- 196 o Keep it simple while sufficient (KISS)
- 197 o All access to repository content is through interaction with Registry Services
- 198 o Interactions between Registry Services and its client are a special case of partner-to-partner
- 199 business process. In fact there is a TPA between clients and Registry Services
- 200 o Interactions between Registry Services and its client are based on ebXML TRP messaging



201 **1.5 Specification Structure**

202 This specification is organized around the following main topics:

- 203 o **Overview** - A high level description of the Registry Service
- 204 o **Registration Service**- A description of the initial registration functionality that bootstraps the
205 communication between a Party associated with a Submitting Organization and the Registry
206 Services. *This section is important to understand fully as it serves as a blueprint when*
207 *reading subsequent sections on how to interpret UML representation of the registry services*
208 *interfaces.*
- 209 o **Object Management Service**- A description of the Object Management functionality of the
210 ebXML Registry that simplifies the definition and subsequent sharing of business objects
211 (e.g. documents) between partners. The types of objects that may be shared are defined in
212 [3] and include Schema documents (e.g. DTDs), Business Process descriptions (e.g. XML
213 documents and software components compliant to a registered business process), Party
214 Profiles and TPAs.
- 215 o **Object Query Management Service**- A description of the Object Query Management
216 functionality of the ebXML Registry that enables querying the repository for managed object
217 contents.

218

218 **2 Overview**

219 **2.1 Role of ebXML Registry**

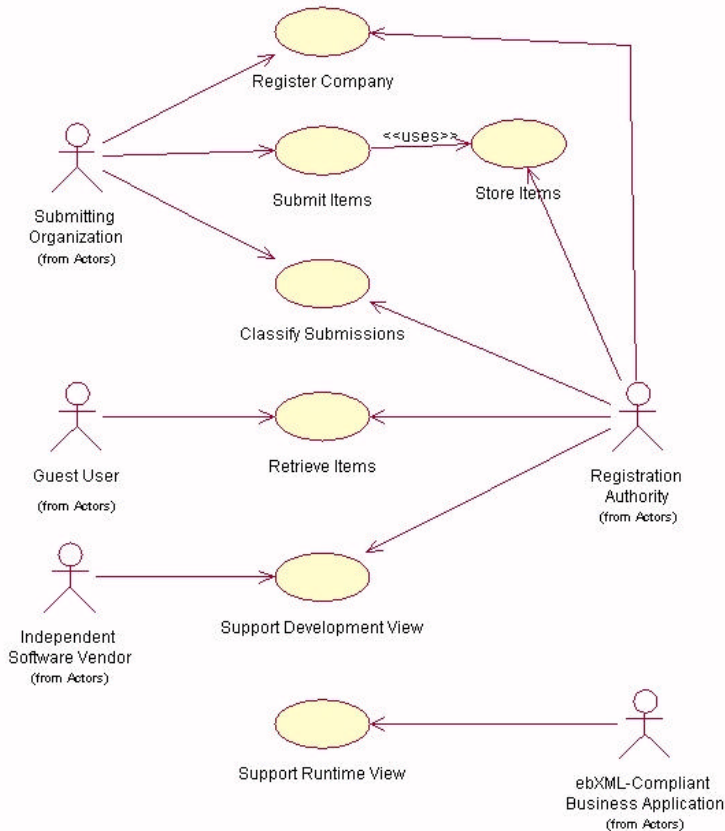
220 The ebXML Registry provides a set of distributed services that enable sharing of information
 221 between interested parties for the purpose of enabling business process integration between
 222 such parties based on the ebXML specifications. The shared information is maintained as objects
 223 in an ebXML Repository [3] that is managed by the ebXML Registry Services defined in this
 224 document and its future versions. All access to the repository is exposed via the interfaces
 225 defined for the Registry Services.

226 **2.2 Use Cases for the Registry Services**

227 This section describes at a high level some use cases illustrating how registry clients may make
 228 use of the registry Services to conduct B2B exchanges. It is meant to be illustrative and not
 229 prescriptive.

230 **2.2.1 Business Domain Workflow Use Cases**

231 Figure 1: shows the Business Domain Workflow Use Cases as identified in [2].



232
 233 **Figure 1: Business Domain Workflow Use Cases**



234 The following scenario textually exemplifies a *subset* of above use cases in terms of interaction
235 between registry clients and the registry. It is currently not a complete listing of use cases
236 envisioned in [2]. It assumes for purposes of example, a buyer and a seller that wish to conduct
237 B2B exchanges using the Rosetta Net PIP3A4 Purchase Order business protocol. It is assumed
238 that both buyer and seller use the same registry service provided by a 3rd party. Note that the
239 architecture supports other possibilities (e.g. each party uses their own private registry). It is
240 assumed that the Registry Service is always operational for the use cases described below.

241 **2.2.2 Parties Register With Registry**

242 Both parties register with the Registry using the Registration Service described in section 3.

243 **2.2.3 Schema Documents Are Submitted**

244 Either the buyer or the seller or a registered 3rd party can submit the necessary schema
245 documents required by the Rosetta Net PIP3A4 Purchase Order business protocol with the
246 Registry using the Object Manager service of the Registry described in section 4.3.

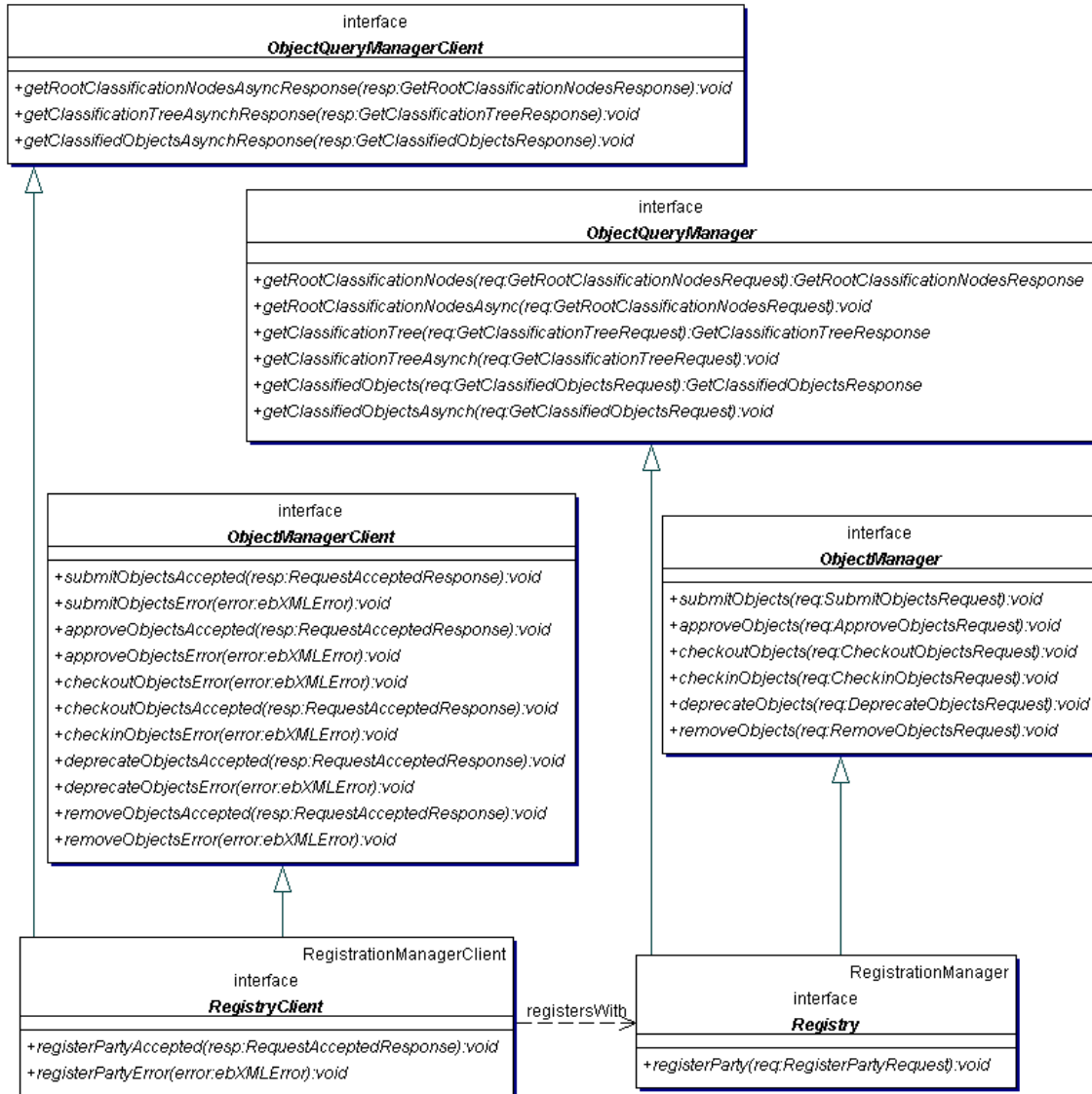
247 **2.2.4 Business Process Documents Are Submitted**

248 Either the buyer or the seller or a registered 3rd party can submit the necessary business process
249 documents required by the Rosetta Net PIP3A4 Purchase Order business protocol with the
250 Registry using the Object Manager service of the Registry described in section 4.3.

251 **2.2.5 TPA is Submitted**

252 Either the buyer or the seller can now create a concrete TPA document that will be used by both
253 parties to conduct B2B exchanges based on the Rosetta Net PIP3A4 Purchase Order business
254 protocol. This is done using a TPA Assembler tool that is a client of the registry services and
255 uses registry services to assemble a TPA from submitted TPA elements. The resulting TPA is
256 submitted to the Registry using the Object Manager service of the Registry described in section
257 4.3.

258 Once the TPA is submitted the parties may now begin to conduct B2B transaction as defined by
259 [4].



260
261

Figure 2: Registry Services Class Diagram

262 Figure 2: Registry Services Class Diagram shows an abstract class diagram for the Registry
 263 Service. It is intended as a high level overview and does not prescribe any particular
 264 implementation. The Registry makes use of a Repository (not shown in diagram) for storing and
 265 retrieving persistent information required by the Registry Services.

266 **2.3 Interfaces Implemented By Registry Service**

267 The ebXML Registry is shown to implement the following interfaces as its sub-services (Registry
 268 Services):

- 269 1. **Registry:** This is the principal bootstrapping interface used by clients of Registry to register
 270 themselves as RegistryClients.



- 271 2. **ObjectManager:** This is the Object Management interface of the Registry services. It
272 provides the functionality to manage the life cycle of any managed object content in the
273 repository.
- 274 3. **ObjectQueryManager:** This is the Object Query Management interface of the Registry
275 services. It provides the functionality to receive query requests and return managed object
276 contents that match the specified query as a response.

277 2.4 Interfaces Implemented By Clients of Registry Service

278 An ebXML application that is a client of the Registry Service must implement the following
279 interfaces:

- 280 1. **RegistryClient:** This is the principal interface implemented by a client of the Registry
281 Services. It is used by the Registry to communicate with the ebXML registry client
282 application during the initial bootstrapping registration process.
- 283 2. **ObjectManagerClient:** This is the interface implemented by a client of the Object
284 Management sub-service of the Registry Services. It is used by the ObjectManager to
285 communicate with the ebXML application during the Object Management process.
- 286 3. **ObjectQueryManagerClient:** This is the interface implemented by a client of the Object
287 Query sub-service of the Registry Services. It is used by the ObjectQueryManager to
288 communicate with the RegistryClient during the Object Query process.

289 3 Registration Service

290 In order to submit or otherwise change content, a RegistryClient must register itself as Party
291 associated with an Organization with the Registry. An Organization currently can have a role of a
292 SubmittingOrganization, ResponsibleOrganization or a RegistrationAuthority as defined by
293 [6]. This is an essential bootstrapping process that is required prior to any other interaction
294 between the RegistryClient and the Registry Service.

295 [Note] Clients that only intend to browse the
296 repository content do not have to register with
297 the repository.

298 Because there is no previously established TPA between the Registry and the RegistryClient, the
299 Registry must somehow make public at least one well-known Transport specific communication
300 address. It is recommended that the registry at least make public a URL to its Registry Service
301 Interface.

302 This section describes the bootstrapping Registration Protocol of the Registry Service that allows
303 a client to register itself with the Registry.

304 Since this is the first use of the stylized use of UML notation in this document, the diagram will
305 be followed by its interpretation. Future processes in the document will only be described
306 pictorially in the interest of brevity.



307

308

Figure 2: Bootstrap Registration Process Sequence Diagram

309

3.1 Interpretation of UML Diagrams Describing an ebXML Business Process

310

311 This section describes in *abstract terms* the conventions used to define ebXML business process
 312 description in UML.

3.1.1 UML Class Diagram

313

314 A UML class diagram is used to describe the Service Interfaces and Actions (as defined by [7])
 315 required to implement an ebXML business process. See Figure 2: Registry Services Class
 316 Diagram for an example. The UML class diagram contains:

- 317 1. A collection of UML classes where each class represents an ebXML document. Such
 318 class definitions, their attributes and their relationships can be used to create schema
 319 elements by a schema assembler tool that is a RegistryClient.
- 320 2. A collection of UML interfaces where each interface represents a Service Interface.
- 321 3. A collection of methods on each interface where each method represents an Action (as
 322 defined by [7]) within the Service Interface representing the UML interface.
- 323 4. Each method within a UML interface specifies one or more parameters, where the type
 324 of each method argument represents the ebXML message type that is exchanged as part
 325 of the Action corresponding to the method. Multiple arguments imply multiple payload
 326 documents within the body of the corresponding ebXML message.

3.1.2 UML Sequence Diagram

327

328 A UML sequence diagram is used to specify the business protocol representing the interactions
 329 between the UML interfaces for an ebXML business process. A UML sequence diagram provides
 330 the necessary information to determine the sequencing of messages, request to response
 331 association as well as request to error response association as described by [7].



332 Each sequence diagram shows the sequence for a specific conversation protocol as method
333 calls from the requestor to the responder. Method invocation may be synchronous or
334 asynchronous based on the UML notation used on the arrow-head for the link. Each method
335 invocation may be followed by a response method invocation from the responder to the
336 requestor to indicate the ResponseName for the previous Request. Possible error response is
337 indicated by a conditional response method invocation from the responder to the requestor. See
338 Figure 2 for an example.

339 **3.2 Interpretation of Bootstrap Registration Process Sequence Diagram**

340 This section describes in *concrete terms* the conventions used to define ebXML business process
341 description in UML diagrams. It uses the Bootstrap Registration Process as a concrete example.

342 There is an implicit TPA between the Registry and the ebXML application that is a client to the
343 Registry services. This TPA defines the Bootstrap Registration Process shown above.

344 **3.2.1 Service Interfaces Defined**

345 In the implicit TPA there are two Business Interfaces which are represented by the UML
346 interfaces in UML sequence diagram above. The Registry must export a Service Interface called
347 the Registry interface. The registry client must export a Service Interface called the
348 RegistryClient interface.

349 **3.2.2 The Actions Defined On Service Interfaces**

350 The Registry interface must support an action named registerParty. The RegistryClient uses the
351 registerParty action of the Registry interface to register its Party and associated Organization
352 with the Registry.

353 The RegistryClient interface must support the following actions for receiving responses to
354 requests made to the Registry:

- 355 1. A registerPartyAccepted action that is used by Registry to notify RegistryClient of
356 successful registration. This serves as a business level Acknowledgement Response to
357 the register action.
- 358 2. A registerPartyError action that is used by Registry to notify RegistryClient of a failure
359 during registration. This serves as an Error Response to the register action.

360 **3.2.3 Requests Defined For Action**

361 The Registry interface defines an action with id of registerParty (corresponding to the interface
362 method).

363 **3.2.3.1 Requests Messages Defined For Request**

364 The register action has a request whose RequestName and RequestMessage are both
365 RegisterPartyRequest. This name is derived from the action name (with first letter capitalized)
366 appended with the suffix Request.

367 **3.2.3.2 Responses Defined For Request**

368 Each Request message may have a Response message associated with it. The ResponseName
369 is inferred from the sequence diagram from the name of the method called upon success of the
370 Request method invocation.



371 Most Registry Requests in this document consistently use a standard business level
372 acknowledgement response message of type RequestAcceptedResponse.

373 3.2.3.3 Exception Responses Defined For Request

374 Each Request message may have one or more Exception Response messages associated with
375 it. The ExceptionResponseName is inferred from the sequence diagram from the name of the
376 method called upon failure of the Request method invocation.

377 Most Registry Requests in this document consistently use the generic ebXMLError message
378 defined by [8] as a business level error response message.

379 3.2.4 Messages Defined For Requests and Responses

380 The type of ebXML message exchanged during each interaction (method invocation in diagram)
381 can be inferred from the type of argument for each method invocation.

382 The RegisterPartyRequest message must contain either an Organization element that provides
383 information on the Organization being registered, or a ManagedObjectRef element that is a
384 reference to a previously defined Organization. Note that it is possible to have multiple Parties
385 be associated with the same Organization.

386 The RegisterPartyRequest message must also contain a Party Profile element that describes the
387 RegistryClient's half of the special TPA between the Registry and the RegistryClient. See
388 Appendix AA.4 for details.

389 3.2.5 Registry Service Interface in TPA SPECIFICATION

390 The above specification of the Registry service interface is expressed in TPA SPECIFICATION
391 [7] as follows¹:

```
392 <BusinessInterface>  
393   <ServiceInterface InterfaceId = "Registry">  
394     <OrgName Partyname = "Registry">Registry</OrgName>  
395     <TaskName>RegisterPartyRequest</TaskName>  
396     <ActionMenu>  
397       <Action id = "registerParty" Type = "basic" Invocation = "asyncOnly">  
398         <Request>  
399           <RequestName>RegisterPartyRequest</RequestName>  
400           <RequestMessage>RegisterPartyRequest</RequestMessage>  
401         </Request>  
402         <Response Required = "yes">  
403           <ResponseName>RequestAcceptedResponse</ResponseName>  
404         </Response>  
405         <ExceptionResponse>  
406           <ExceptionResponseName>ebXMLError  
407           </ExceptionResponseName>  
408         </Response>  
409       </Action>  
410     </ActionMenu>  
411   </ServiceInterface>  
412 </BusinessInterface>
```

¹ It should be noted that this XML fragment is tentative since the exact grammar of the ebXML TPA has not yet been defined.



413 3.2.6 RegistryClient Service Interface in TPA SPECIFICATION

414 The above specification of the RegistryClient service interface is expressed in TPA
415 SPECIFICATION [7] as follows²:

```
416 <BusinessInterface>  
417 <ServiceInterface InterfaceId = "RegistryClient">  
418 <OrgName Partyname = "RegistryClient">RegistryClient</OrgName>  
419 <TaskName>RegisterPartyResponse</TaskName>  
420 <ActionMenu>  
421 <Action id = "registerPartyAccepted" Type = "basic" Invocation = "asyncOnly">  
422 <Request>  
423 <RequestName>RequestAcceptedResponse</RequestName>  
424 <RequestMessage> RequestAcceptedResponse </RequestMessage>  
425 </Request>  
426 <Response Required = "no"/>  
427 </Action>  
428 <Action id = "registerPartyError" Type = "basic" Invocation = "asyncOnly">  
429 <Request>  
430 <RequestName>ebXMLError</RequestName>  
431 <RequestMessage> ebXMLError</RequestMessage>  
432 </Request>  
433 <Response Required = "no"/>  
434 </Action>  
435 </ActionMenu>  
436 </ServiceInterface>  
437 </BusinessInterface>
```

438 4 Object Management Service

439 [Note] The workflow envisioned in [2] is not addressed
440 completely in this version. This chapter is a
441 simplified sub-set of that workflow.

442 This section defines the Object Management service of the Registry. The Object Management
443 Service is a sub-service of the Registry service. It provides the functionality required by
444 RegistryClient's to manage the life cycle of managed object contents (e.g. documents) required
445 for ebXML business processes. The Object Management Service can be used with all types of
446 managed object contents including the built-in managed object contents specified in [3] such as
447 Classification and Association.

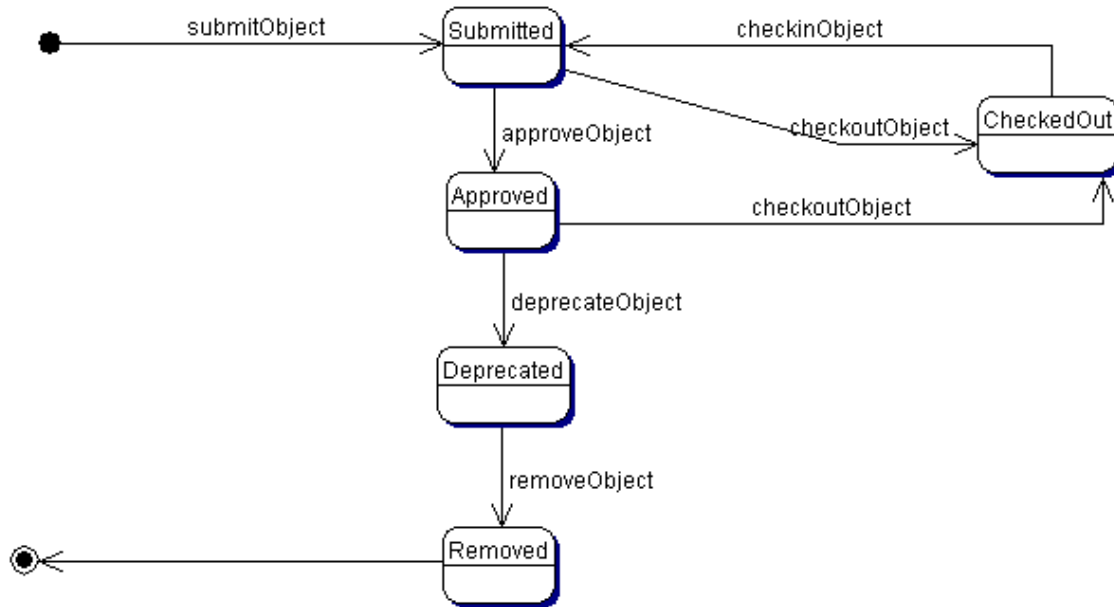
448 Once an ebXML client of the Registry Services has successfully been through the Bootstrapping
449 Registration Process, it is now capable of using the Object Management services of the Registry.

450 4.1 Life Cycle of a Managed Object

451 The main purpose of the Object Management service is to manage the life cycle of managed
452 object contents in the repository.

453 Figure 3 shows the typical life cycle of a managed object content.

² It should be noted that this XML fragment is tentative since the exact grammar of the ebXML TPA has not yet been defined.



454

455

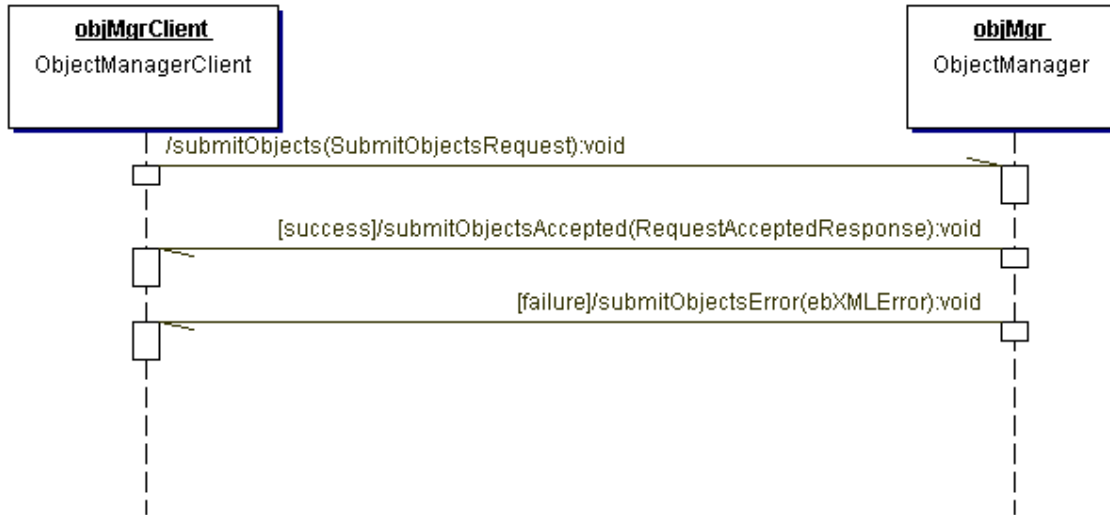
Figure 3: Life Cycle of a Managed Object

456 **4.2 Object Attributes**

457 A managed object content is associated with a set of standard meta-data defined as attributes of
 458 the ManagedObject class described in [3]. These attributes reside outside of the actual object
 459 content and provide valuable meta-data about the managed object content. An XML schema
 460 element called ManagedObject (See Appendix AA.3 for details.) is defined that encapsulates all
 461 object meta-data attributes defined in [3] as attributes of the schema element.

462 **4.3 The Submit Objects Protocol**

463 This section describes the protocol of the Registry Service that allows a RegistryClient to submit
 464 one or more managed object contents in the repository using the *Object Manager* on behalf of a
 465 Submitting Organization. It is expressed in UML notation as described in section 3.1.



466

467

Figure 4: Submit Objects Sequence Diagram

468 For details on the schema for the business documents shown in this process refer to Appendix
469 AA.5.

470 **4.3.1 ManagedObject**

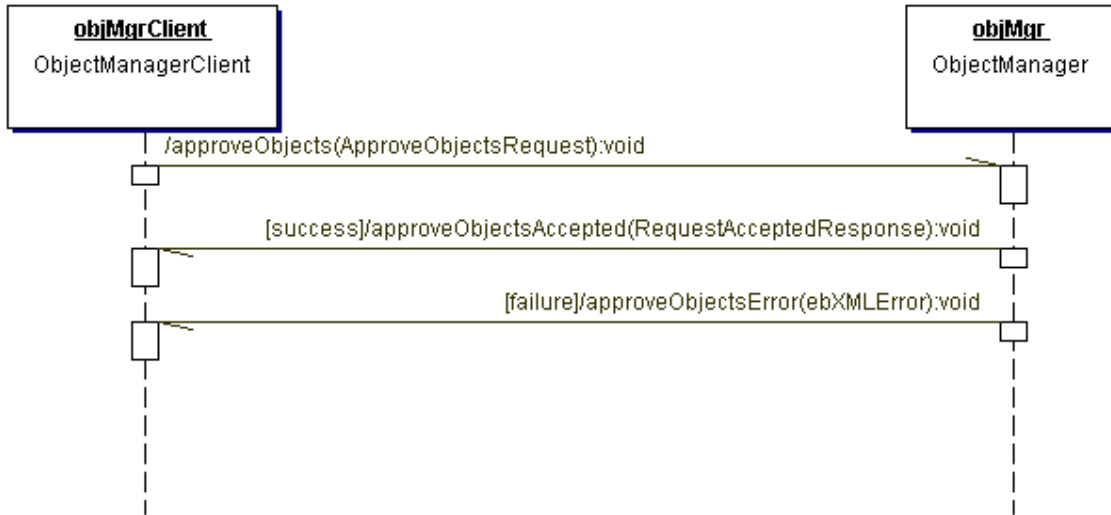
471 The SubmitObjectRequest message includes a ManagedObject element which identifies the
472 Submitting Party. It also includes 1 or more SubmittedObject elements.

473 Each SubmittedObject element specifies a ManagedObject element which provides standard
474 meta-data about the object being submitted to the repository as defined by [3]. Note that these
475 standard ManagedObject attributes are separate from the managed object content itself, thus
476 allowing the ebXML Repository to catalog arbitrary objects. In addition each SubmittedObject in
477 the request may optionally specify any number of Classifications or Associations for the
478 SubmittedObject.

479 In summary each managed object content in the Repository is associated with a standard set of
480 meta-data attributes collectively described by a ManagedObject element in XML. A simple URI
481 reference to a ManagedObject is represented in XML as a ManagedObjectRef element.

482 **4.4 The Approve Objects Request**

483 This section describes the protocol of the Registry Service that allows a client to approve one or
484 more previously submitted managed object contents using the Object Manager. Once a
485 managed object content is approved it will become available for use by business parties (e.g.
486 during the assembly of new TPAs and Party Profiles). It is expressed in UML notation as
487 described in section 3.1.



488

489

Figure 5: Approve Objects Sequence Diagram

490

For details on the schema for the business documents shown in this process refer to Appendix AA.6.

491

492

4.5 The Deprecate Objects Request

493

This section describes the protocol of the Registry Service that allows a client to deprecate one or more previously submitted managed object content using the Object Manager. Once an object is deprecated, no *new* associations to that object can be submitted. In effect the managed object content is marked as removed or deleted. However, existing references to a deprecated managed object content continue to function normally. The deprecate object protocol is expressed in UML notation as described in section 3.1.

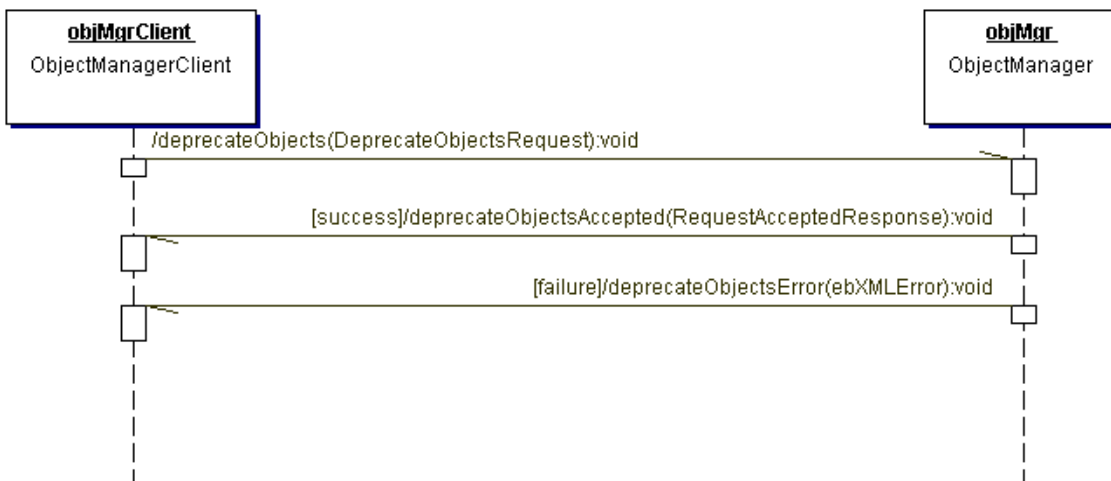
494

495

496

497

498



499

500

Figure 6: Deprecate Objects Sequence Diagram

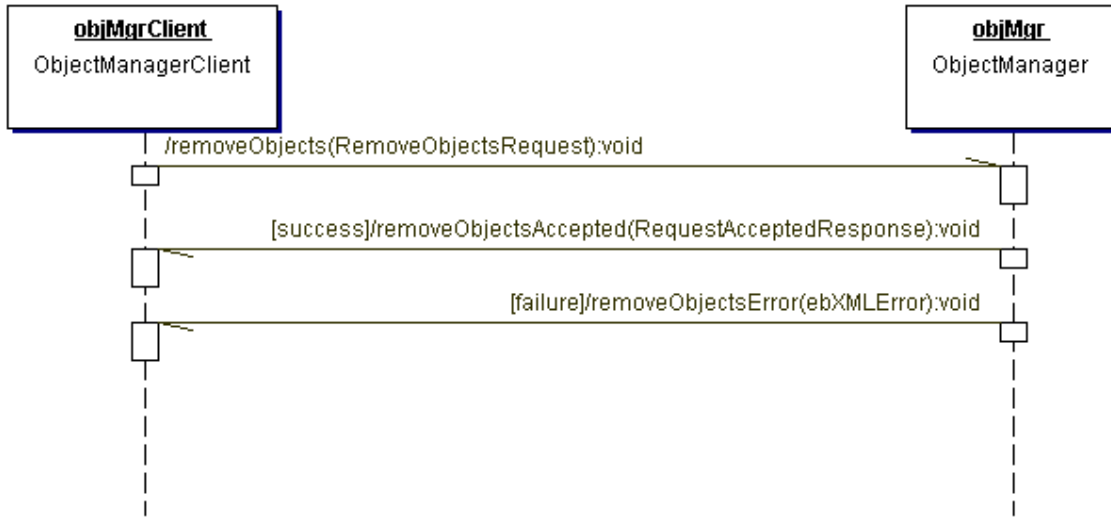
501

For details on the schema for the business documents shown in this process refer to Appendix AA.7.

502

503 **4.6 The Remove Objects Request**

504 This section describes the protocol of the Registry Service that allows a client to remove one or
 505 more previously deprecated managed object contents using the Object Manager. An object
 506 cannot be removed as long as there exists 1 or more objects with associations to that object.
 507 Once an object is removed it will be not be present at all in the Registry. The remove object
 508 protocol is expressed in UML notation as described in section 3.1.



509

510

Figure 7: Remove Objects Sequence Diagram

511 For details on the schema for the business documents shown in this process refer to Appendix
 512 AA.8.

513 **5 Object Query Management Service**

514 This section describes the capabilities of the Registry Service that allows a client
 515 (ObjectQueryManagerClient) to search for (query) managed object contents in the ebXML
 516 Repository using the ObjectQueryManager *interface of the Registry*.

517 Due to the synchronous nature of queries all interactions between the
 518 ObjectQueryManagerClient and the ObjectQueryManager are synchronous in nature as reflected
 519 in the UML sequence diagrams that follow. Any errors in the query request messages are
 520 indicated in the corresponding query response message.

521 **5.1 Design Goals For Object Query Management Support**

522 The following goal motivated the design of Object Query Management service:

- 523 1. Make it simple for Registry clients to query the registry for managed object contents
- 524 2. Do not invent a new query language (a very large wheel to reinvent)
- 525 3. Make it simple for Registry providers to implement the Repository using a relational
526 database
- 527 4. Support all query mechanisms described in [3]



528 5. Leverage security mechanism that will be specified in ebXML TRP rather than inventing
529 a different model for the Registry

530 5.2 Browse and Drill Down Query Support

531 The browse and drill down query style is completely supported by a set of primitive interactions
532 between the ObjectQueryManagerClient and the ObjectQueryManager as described next. Note
533 that for each query request/response there is both a synchronous and asynchronous version of
534 the interaction.

535 [Note] For the Tokyo POC it is only required that the
536 asynchronous queries must be supported by
537 registry client and registry services.

538 5.2.1 Get Root Classification Nodes Request

539 An ObjectQueryManagerClient send this request to get a list of root ClassificationNodes (nodes
540 with no parent) defined in the repository. Note that it is possible to specify a namePattern
541 attribute that can filter on the name attribute of the root ClassificationNodes using a wildcard
542 pattern defined by SQL-92 LIKE clause. It is expressed in UML notation as described in section
543 3.1.



544

545

Figure 8: Get Root Classification Nodes Sequence Diagram



546



547 **Figure 9: Get Root Classification Nodes Asynchronous Sequence Diagram**

548 For details on the schema for the business documents shown in this process refer to A.10 and
549 A.11.

550 **5.2.2 Get Classification Tree Request**

551 An ObjectQueryManagerClient send this request to get the ClassificationNode sub-tree defined
552 in the repository under the ClassificationNode specified in the request. Note that a
553 GetClassificationTreeRequest can specify an integer attribute called *depth* to get the sub-tree
554 upto the specified depth. If depth is 1 (default) then only the immediate children of the specified
555 ClassificationItemRef are returned. If depth is 0 then the entire sub-tree is retrieved.

557 It is expressed in UML notation as described in section 3.1.



558

559 **Figure 10: Get Classification Tree Sequence Diagram**



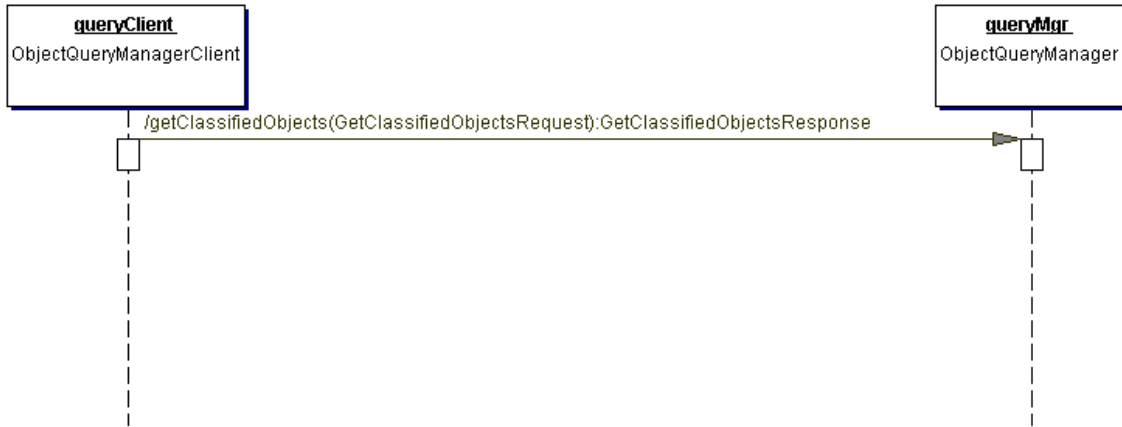
560

561 **Figure 11: Get Classification Tree Asynchronous Sequence Diagram**

562 For details on the schema for the business documents shown in this process refer to A.12 and
563 A.13.

564 **5.2.3 Get Classified Objects Request**

565 An ObjectQueryManagerClient send this request to get a list of references to managed object
 566 contents defined in the repository that are classified by the specified ClassificationNodes in the
 567 ManagedObjectRefList in the request. Note that it is possible to get managed object contents
 568 based on matches with multiple classifications. It is expressed in UML notation as described in
 569 section 3.1.



570

571

Figure 12: Get Classified Objects Sequence Diagram



572

573

Figure 13: Get Classified Objects Asynchronous Sequence Diagram

574 For details on the schema for the business documents shown in this process refer to A.14 and
 575 A.15.

576 **5.3 Ad Hoc Query Support**

577 Details will be specified post Tokyo.

578 **5.4 Keyword Search Based Query Support**

579 The Registry provides a search engine functionality to search for managed object contents that
 580 contain specified keywords in their content or attributes. Details will be specified post Tokyo.



581 **5.5 Object Retrieval**

582 The response messages that is returned by ObjectQueryManager contain zero or more
583 ManagedObjectRef elements, one for each object that matched the query. Each
584 ManagedObjectRef element contains a URI for the object matching the query in the repository.
585 The client can use the URI to retrieve the object. Note that security issues will be addressed by
586 ebXML TRP security mechanisms that are outside the scope of this document.

587 **6 References**

- 588 [1] ebXML Glossary, see http://www.ebxml.org/documents/199909/terms_of_reference.htm
- 589 [2] Registry and Repository Part 1: Business Domain Model
590 <http://www.ebxml.org/specdrafts/RegRepv1-0.pdf>
- 591 [3] ebXML Repository Information Model
- 592 [4] ebXML Messaging Service Specification, Version 0.21,
593 [http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-](http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf)
594 [21.pdf](http://ebxml.org/project_teams/transport/private/ebXML_Messaging_Service_Specification_v0-21.pdf)
- 595 [5] ebXML Business Process Meta-model
596 <http://www.ebxml.org/specdrafts/Busv2-0.pdf>
- 597 [6] ebXML Core Components Meta-model (unable to find reference despite best attempt)
- 598 [7] Trading-Partner Specification
599 http://www.ebxml.org/project_teams/trade_partner/private/
- 600 [8] ebXML TRP Error Handling Specification

601 **2 Acknowledgments**

602 The members of the Registry and Repository team wish to acknowledge the support of the
603 members of the various ebXML working groups who have contributed ideas and suggestions for
604 this proposal.



605 **Appendix A Schemas and DTD Definitions**

606 The following are definitions for the various ebXML Message payloads described in this
607 document.

608 [Note] The DTDs for messages should be considered as
609 works-in-progress. They are open to change
610 based on collaboration with and input from the
611 team and various other working groups.

612 [Note] Several DTDs use a common boiler plate DTD
613 ManagedObject.dtd that must be read in order to
614 understand the main DTD. This common DTD is
615 defined in Appendix AA.3



616 **A.1 RequestAcceptedResponse Message DTD**

```
617 <!ELEMENT RequestAcceptedResponse EMPTY>
618 <!ATTLIST RequestAcceptedResponse xml:lang NMTOKEN #REQUIRED
619         interfaced CDATA #REQUIRED
620         requestMessage CDATA #REQUIRED
621         actionId CDATA #REQUIRED >
622 <!ENTITY % interfaced "">
```

623 **A.2 ebXMLError Message DTD**

```
624 <!ELEMENT ebXMLError (ErrorHeader , ErrorLocation* )>
625 <!ATTLIST ebXMLError xml:lang NMTOKEN #REQUIRED >
626 <!ELEMENT ErrorHeader (Severity , ErrorCode , ErrorDesc? , MinRetrySecs? )>
627 <!ATTLIST ErrorHeader ID NMTOKEN #REQUIRED >
628 <!ELEMENT Severity (#PCDATA )>
629
630 <!-- Either Warning, TransientError or HardError -->
631 <!ELEMENT ErrorCode (#PCDATA )>
632
633 <!-- string max 14 char -->
634 <!ELEMENT ErrorDesc (#PCDATA )>
635
636 <!-- string max 256 (?) char -->
637 <!ELEMENT MinRetrySecs (#PCDATA )>
638
639 <!-- An integer -->
640 <!ELEMENT SwVendorErrorRef (#PCDATA )>
641
642 <!-- string max 256 (?) chars -->
643 <!ELEMENT ErrorLocation (RefToMessageId? , (Href | XMLDocumentErrorLocn ) )>
644 <!ATTLIST ErrorLocation ID NMTOKEN #REQUIRED >
645 <!ELEMENT RefToMessageId (#PCDATA )>
646
647 <!ELEMENT Href (#PCDATA )>
648
649 <!ELEMENT XMLDocumentErrorLocn (DocumentId , Xpath )>
650
651 <!ELEMENT DocumentId (#PCDATA )>
652
653 <!ELEMENT Xpath (#PCDATA )>
```

654



655 A.3 ManagedObject DTD

```
656 <!ENTITY % ebXMLError SYSTEM "ebXMLError.dtd">
657
658 <!-- Pulls in the module at this spot in my DTD: -->
659 %ebXMLError;
660
661 <!ELEMENT ManagedObject EMPTY>
662
663 <!--
664 The following are standard document attributes that provide meta-data
665 about the document. These are based on the ebXML Repository Information Model
666 specification:
667 -->
668 <!ATTLIST ManagedObject guid          CDATA #REQUIRED>
669
670 <!ATTLIST ManagedObject uri           CDATA #REQUIRED>
671
672 <!ATTLIST ManagedObject type          (UserDefined |
673                                     Schema |
674                                     Process |
675                                     PartyProfile |
676                                     ServiceInterface |
677                                     BusinessService |
678                                     Role |
679                                     Transport |
680                                     Association |
681                                     ClassificationNode |
682                                     Classification ) #REQUIRED>
683
684 <!ATTLIST ManagedObject name          CDATA #REQUIRED>
685
686 <!ATTLIST ManagedObject description   CDATA #IMPLIED>
687
688 <!ATTLIST ManagedObject mimeType      CDATA #IMPLIED>
689
690 <!ATTLIST ManagedObject majorVersion  CDATA "0">
691
692 <!ATTLIST ManagedObject minorVersion  CDATA "1">
693
694 <!ATTLIST ManagedObject registryStatus (Submitted | Approved | Deployed | Deprecated )
695 "Submitted">
696
697 <!ELEMENT ManagedObjectRef EMPTY>
698 <!ATTLIST ManagedObjectRef guid CDATA #REQUIRED
699                               uri CDATA #IMPLIED
700                               name CDATA #IMPLIED >
701 <!ELEMENT ManagedObjectRefList (ManagedObjectRef )*>
702
703 <!ELEMENT ExternalObject EMPTY>
704 <!ATTLIST ExternalObject guid CDATA #REQUIRED
705                               uri CDATA #IMPLIED
706                               description CDATA #IMPLIED >
707 <!ELEMENT ExternalObjectList (ExternalObject )*>
```



708
709 <!--
710 A Classification specifies references to two previously submitted
711 managed object contents.
712
713 The first ManagedObjectRef is ref to a ManagedObject being classified
714 The second ManagedObjectRef is a ref to ClassificationNode
715
716 The first ManagedObjectRef is optional when Classification is defined part of
717 a SubmittedObject.
718 -->
719 <!ELEMENT Classification (ManagedObjectRef? , ManagedObjectRef)>
720
721 <!ELEMENT ClassificationList (Classification)*>
722
723 <!--
724 A Classification specifies references to two previously submitted
725 managed object contents.
726
727 The first ManagedObjectRef is ref to the "from" ManagedObject in association
728 The first ManagedObjectRef is ref to the "to" ManagedObject in association
729
730 The first ManagedObjectRef is optional when Classification is defined part of
731 a SubmittedObject.
732 -->
733 <!ELEMENT Association (ManagedObjectRef? , ManagedObjectRef)>
734 <!ATTLIST Association fromLabel CDATA #IMPLIED
735 toLabel CDATA #IMPLIED
736 type CDATA #IMPLIED
737 bidirection CDATA #IMPLIED >
738 <!ELEMENT AssociationList (Association)*>
739
740

741 **A.4 RegisterPartyRequest Message DTD**

742 The RegisterPartyRequest Message includes a Party Profile specified by a TPA element, which
743 conforms to the DTD specified for TPA SPECIFICATION in [7].



```
744 <!-- Pulls in the module at this spot in my DTD: -->
745 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
746
747 %managedObject;
748
749 <!ENTITY % tpa SYSTEM "tpa_1_0_6.dtd">
750
751 %tpa;
752
753 <!--
754 The Organization element needs to be defined by CC team.
755 For now this is a place holder.
756 -->
757 <!ELEMENT Organization EMPTY>
758
759 <!--
760 The party must be associated with an Organization. The Organization
761 may be defined in the request or may refer to a previously defined
762 Organization referred to by ManagedObjectRef.
763
764 The party must provide a PartyProfile which currently is represented
765 by a TPA. TP team needs to define PartyProfile post Tokyo.
766 -->
767 <!ELEMENT RegisterPartyRequest ( (Organization | ManagedObjectRef ) , TPA )>
```

768 **A.5 SubmitObjectsRequest Message DTD**

```
769 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
770
771 <!-- Pulls in the module at this spot in my DTD: -->
772 %managedObject;
773
774 <!--
775 The ManagedObjectRef must be a ref to a previously registered Party
776 which is the Submitting party
777
778 The SubmittedObject provides meta data for submitted object
779
780 Note object being submitted is in a separate document that is not
781 in this DTD.
782 -->
783 <!ELEMENT SubmitObjectsRequest (ManagedObjectRef , SubmittedObject+ )>
784
785 <!--
786 The ManagedObject provides meta data about the object being submitted
787
788 ClassificationList can be optionally be specified to define Classifications
789 for the SubmittedObject
790
791 AssociationList can be optionally be specified to define Associations
792 for the SubmittedObject
793
794 The ExternalObjectList provides zero or more external objects related to
```



```
795 the object being submitted.
796
797 -->
798 <!ELEMENT SubmittedObject (ManagedObject , ClassificationList?,
799 AssociationList?, ExternalObjectList? )>
```

800 **A.6 ApproveObjectsRequest Message DTD**

```
801 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
802
803 <!-- Pulls in the module at this spot in my DTD: -->
804 %managedObject;
805
806 <!--
807 The ManagedObjectRef is to a previously registered Party
808 which is the approving party.
809
810 The ManagedObjectRefList is the list of
811 refs to the managed object contents being approved.
812 -->
813
814 <!ELEMENT ApproveObjectsRequest (ManagedObjectRef , ManagedObjectRefList )>
```

815 **A.7 DeprecateObjectsRequest Message DTD**

```
816 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
817
818 <!-- Pulls in the module at this spot in my DTD: -->
819 %managedObject;
820
821 <!--
822 The ManagedObjectRef is to a previously registered Party
823 which is the deprecating party.
824
825 The ManagedObjectRefList is the list of
826 refs to the managed object contents being deprecated.
827 -->
828 <!ELEMENT DeprecateObjectsRequest (ManagedObjectRef , ManagedObjectRefList )>
```

829 **A.8 RemoveObjectsRequest Message DTD**

```
830 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
831
832 <!-- Pulls in the module at this spot in my DTD: -->
833 %managedObject;
834
835 <!--
836 The ManagedObjectRef is to a previously registered Party
837 which is the removing party.
838
839 The ManagedObjectRefList is the list of
840 refs to the managed object contents being removed
```



841
842

```
-->  
<!ELEMENT RemoveObjectsRequest (ManagedObjectRef , ManagedObjectRefList )>
```

843 **A.9 ClassificationNode DTD**

844 This DTD is used to submit ClassificationNodes. It is capable of submitting a single node or an
845 entire sub-tree.



```
846 <!ENTITY % managedObject SYSTEM "ManagedObject.dtd">
847
848 <!-- Pulls in the module at this spot in my DTD: -->
849 %managedObject;
850
851 <!--
852 ClassificationNode is used to submit a Classification tree to the Registry.
853 Note that this is a recursive schema definition.
854 The parent attribute of a node in tree is implied by the enclosing ClassificationNode
855 The children nodes of a node are implied by enclosing immediate child elements
856 of type ClassificationNode.
857 -->
858 <!ELEMENT ClassificationNode (ClassificationNode* )>
859
860 <!--
861 The name of the ClassificationNode. Maps to the name attribute
862 of the ManagedObject meta data class.
863 -->
864 <!ATTLIST ClassificationNode name CDATA #REQUIRED>
865
866 <!--
867 ClassificationNodeRef is used by the ObjectQueryManager
868 for various query responses. It represents a tree of
869 ClassificationNodeRef.
870 -->
871 <!ELEMENT ClassificationNodeRef (ManagedObjectRef , ClassificationNodeRef* )>
872
873 <!--
874 ClassificationNodeRefList is used to send a list of
875 ClassificationNodeRef when returning immediate children nodes
876 of a node.
877 -->
878 <!ELEMENT ClassificationNodeRefList (ClassificationNodeRef )*>
```

879 **A.10 GetRootClassificationNodesRequest Message DTD**

```
880 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
881
882 %classificationNode;
883
884 <!--
885 The query request that gets the specified root ClassificationNodes
886 -->
887 <!ELEMENT GetRootClassificationNodesRequest EMPTY>
888
889 <!--
890 The namePattern follows SQL-92 syntax for the pattern specified in
891 LIKE clause. It allows for selecting only those root nodes that match
892 the namePattern. The default value of '*' matches all root nodes.
893 -->
894 <!ATTLIST GetRootClassificationNodesRequest namePattern CDATA '*' >
```



895 **A.11 GetRootClassificationNodesResponse Message DTD**

```
896 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
897
898 %classificationNode;
899
900 <!--
901 The response includes a ManagedObjectRefList which has zero or more
902 references to ManagedObjects that represent ClassificationSchemeRefs
903 -->
904 <!ELEMENT GetRootClassificationNodesResponse (ManagedObjectRefList | ebXMLError )>
```

905 **A.12 GetClassificationTreeRequest Message DTD**

```
906 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
907
908 %classificationNode;
909
910 <!--
911 Get the ClassificationItemTreeRef under the ClassificationItemRef specified
912 by ManagedObjectRef.
913 -->
914 <!ELEMENT GetClassificationTreeRequest (ManagedObjectRef )>
915
916 <!--
917 If depth is 1 just fetch immediate child
918 nodes, otherwise fetch the descendant tree upto specified depth level.
919 If depth is 0 that implies fetch entire sub-tree
920 -->
921 <!ATTLIST GetClassificationTreeRequest depth CDATA '1' >
```

922 **A.13 GetClassificationTreeResponse Message DTD**

```
923 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
924
925 %classificationNode;
926
927 <!--
928 The response includes a ClassificationNodeRefList which includes only
929 immediate ClassificationNodeRef children nodes if depth attribute in
930 GetClassificationTreeRequest was 1, otherwise the decendent nodes
931 upto specifed depth level are returned.
932 -->
933 <!ELEMENT GetClassificationTreeResponse (ClassificationNodeRefList | ebXMLError )>
```

934 **A.14 GetClassifiedObjectsRequest Message DTD**

```
935 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
936
937 %classificationNode;
938
939 <!--
940 Get refs to all managed object contents that are classified by all the
941 ClassificationNodeRef specified by ManagedObjectRefList.
```



```
942
943 Note this is an implicit logical AND operation
944 -->
945 <!ELEMENT GetClassifiedObjectsRequest (ManagedObjectRefList )>
946
947 <!--
948 objectType attribute can specify the type of objects that the registry
949 client is interested in, that is classified by this ClassificationNode.
950 It is a String that matches a choice in the type attribute of ManagedObject.
951 The default value of '*' implies that client is interested in all types
952 of managed object contents that are classified by the specified ClassificationNode.
953 -->
954 <!ATTLIST GetClassifiedObjectsRequest objectType CDATA "*">
```

955 **A.15 GetClassifiedObjectsResponse Message DTD**

```
956 <!ENTITY % classificationNode SYSTEM "ClassificationNode.dtd">
957
958 %classificationNode;
959
960 <!--
961 The response includes a ManagedObjectRefList which has zero or more
962 references to ManagedObjects that are classified by the ClassificationNodeRef
963 specified in the GetClassifiedObjectsRequest.
964 -->
965 <!ELEMENT GetClassifiedObjectsResponse (ManagedObjectRefList | ebXMLError )>
```

966 **Appendix B TPA Between Registry Client And Registry**

967

968 **Appendix C Example XML for Submitting a** 969 **Classification Tree**

970 The following XML message submits the classification tree for the classification tree example in
971 [3].

```
972
973 <?xml version = "1.0"?>
974 <!DOCTYPE ClassificationNode SYSTEM "ClassificationNode.dtd">
975
976 <!--
977 This is a sample XML that is used to submit a Classification tree to the Registry.
978 Note that this is a recursive schema definition.
979 The parent attribute of a node in tree is implied by the enclosing ClassificationNode
980 The children nodes of a node are implied by enclosing immediate child elements
981 of type ClassificationNode.
982 -->
983 <ClassificationNode name = "Industry">
984     <ClassificationNode name = "Automotive">
985         <ClassificationNode name = "Geography">
986             <ClassificationNode name = "US"/>
987             <ClassificationNode name = "Europe"/>
```



```

988     </ClassificationNode>
989     </ClassificationNode>
990     <ClassificationNode name = "Health Care"/>
991     <ClassificationNode name = "Retail"/>
992 </ClassificationNode>

```

993 **Appendix D Terminology Mapping**

994 While every attempt has been made to use the same terminology used in previous works there
 995 are some terminology differences.

996 The following table shows the terminology mapping between this specification and that used in
 997 other specifications and working groups.

This Document	OASIS	ISO 11179
"managed object content"	Registered Object	
ManagedObject	Registry Item	Administered Component
ExternalObject	Related Data	N/A
Object.guid	RaitemId	
Object.uri	ObjectLocation	
ManagedObject.type	DefnSource, PrimaryClass, SubClass	
Object.name	CommonName	
ManagedObject.description	Description	
ManagedObject.mimeType	MimeType	
ManagedObject.majorVersion	partially to Version	
ManagedObject.minorVersion	partially to Version	
ManagedObject.registryStatus	RegStatus	

998

999

Table 1: Terminology Mapping Table

1000

1001 **Appendix E Open Issues**

1002 The following are some open issues that will be evaluated based on further investigation and
 1003 within the context of broader team discussions:

- 1004 o Should Library Control Service be broken out into a separate service from the Object
 1005 Manager (SH). . This is a post Tokyo issue.
- 1006 o Add more comprehensive and common error and warning messages throughout. Consider
 1007 adding error responses such as "Not Authenticated," and "Registry Not Available," etc. Align
 1008 these messages with Technical Architecture WG. This is a post Tokyo issue.



- 1009 o Why are we requiring that an "Authorization Mechanism" be accomplished as a separate
1010 (internet-based) external call? (SH) Why can't this approval process be modeled as an
1011 Internal Implementation detail?
- 1012 o When someone tries to request a deprecated object, they should be notified that the object
1013 has been deprecated. (LG) "SO" can deprecate objects but where is the Notification
1014 processing related to others who will attempt to associate with those deprecate objects? This
1015 is a post Tokyo issue.
- 1016 o When an object is deprecated and/or then removed, what do we maintain within the
1017 Registry/Repository? (LG) When an object is removed, the Meta Data for the object should
1018 be retained and marked as "Removed." Farrukh's opinion is that this is exactly what
1019 Deprecation was designed to do. It is essentially a way to mark an object as logically deleted
1020 so future use (references) are prevented while existing uses would still work. Should we
1021 close this issue?
- 1022 o Remove the requirement that RS must be accessible over ebXML Messaging Service (SH)
- 1023 o Inconsistency noted by DW that objects retrieval is not done over ebXML Messaging Service
1024 while all other interactions are. This is a post Tokyo issue.