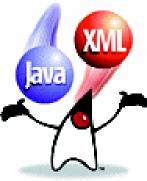
1



2 **Jakarta XML Registries**

3 **Proposed Final Draft: 4/10/2002**

4 This version: JAXR Version 1.0

5

6

7 Please send technical comments [to:](ftp://to:_jaxr-experts@east.sun.com/) jaxr-experts@east.sun.com

8 Please send business comments [to:](ftp://to:_jaxr-business@east.sun.com/) jaxr-business@east.sun.com

9

10

11

12

13

14

15

16



17 Farrukh Najmi <[Farrukh.Najmi@sun.com](mailto:Farrukh.Najmi@sun.com)>

18

JavaTM API for XML Registries April 10, 2002

24

25 JavaTM API for XML Registries (JAXR) Specification ("Specification")

26 Version: 1.0

27 Status: Proposed Final Draft

28 Release: 4/10/2002

29 Copyright 2002 Sun Microsystems, Inc.

30 901 San Antonio Road, Palo Alto, California 94303, U.S.A.

31 All rights reserved.

32

Sun Microsystems Page 4

|  |  |  |  |
| --- | --- | --- | --- |
| 143  144  145 | JavaTM API for XML Registries April 10, 2002 | |  |
| **Table of Contents**  **Introduction 13**  **Introduction 13** | | |
| 146 |  | 1.1 Status of this Document | 13 |
| 147 |  | 1.2 Abstract | 13 |
| 148 |  | 1.3 General Conventions | 13 |
| 149 |  | 1.4 Target Audience | 14 |
| 150 |  | 1.5 JAXR Expert Group | 14 |
| 151 |  | 1.6 Acknowledgements | 15 |
| 152 |  | 1.7 Relationship to Other Java APIs | 15 |
| 153 |  | 1.7.1 JAXP | 15 |
| 154 |  | 1.7.2 JAXB | 15 |
| 155 |  | 1.7.3 JAX-RPC | 16 |
| 156 |  | 1.7.4 JAXM | 16 |
| 157 |  | 1.8 Design Objectives | 16 |
| 158 |  | 1.8.1 Goals | 16 |
| 159 |  | 1.8.2 Non Goals | 17 |
| 160 |  | 1.9 Caveats and Assumptions | 17 |
| 161 | **2** | **Overview** | **18** |
| 162 |  | 2.1 What Is a Registry | 18 |
| 163 |  | 2.2 Registry Use Case Scenarios | 18 |
| 164 |  | 2.3 Participant Roles | 20 |
| 165 |  | 2.3.1 Submitting Organization | 20 |
| 166 |  | 2.3.2 Content Submitter | 20 |
| 167 |  | 2.3.3 Registry Operator | 20 |
| 168 |  | 2.3.4 Registry Guest | 20 |
| 169 |  | 2.4 Registry Vs. Repository | 20 |
| 170 |  | 2.4.1 Repository and Repository Items | 21 |
| 171 |  | 2.4.2 Registry and Registry Objects | 21 |
| 172 |  | 2.5 Functionality of a Registry | 21 |
| 173 |  | 2.5.1 Registry as Electronic Yellow Pages | 21 |
| 174 |  | 2.5.1.1 Flexible Classification Capability | 21 |
| 175 |  | 2.5.2 Registry as a Database of Relatively Static Data | 22 |
| 176 |  | 2.5.3 Registry as Electronic Bulletin Board | 22 |
| 177 |  | 2.6 Existing Registry Specifications | 22 |
| 178 |  | 2.7 Registry Provider | 23 |
| 179 |  | 2.8 JAXR Provider | 23 |
| 180 |  | 2.9 JAXR Client | 23 |
| 181 |  | 2.10 Support for Multiple Registry Specifications | 23 |

Sun Microsystems Page 5

JavaTM API for XML Registries April 10, 2002

182 2.10.1 Capability Profiles 24

183 2.10.1.1 Assignment of Capability Level to Methods 24

184 2.10.1.2 Assignment of Capability Level to Interfaces and

185 Classes 24

186 2.10.1.3 Declaration of Capability Level by a JAXR Provider 24

187 2.10.2 Level 0 Profile 25

188 2.10.3 Level 1 Profile 25

189 2.10.4 Capability Level and JAXR Clients 25

190 2.11 Capability Levels and Registry Standards 25

191 **3 Architecture 26**

192 3.1 JAXR Client 27

193 3.2 Interface *Connection* 27

194 3.3 Interface *RegistryService* 28

195 3.4 Capability-specific Interfaces 28

196 3.5 The JAXR Provider 28

197 3.5.1 JAXR Pluggable Provider 29

198 3.5.2 Registry-specific JAXR Provider 29

199 3.5.3 JAXR Bridge Providers 30

200 3.6 Registry Provider 30

201 3.7 JAXR API Package Structure 30

202 3.7.1 Responses and Exceptions 31

203 3.7.2 Main Interfaces 31

204 **4 Information Model 33**

205 4.1 Information Model: Public View 33

206 4.1.1 RegistryObject 34

207 4.1.2 Organization 35

208 4.1.3 Service 35

209 4.1.4 ServiceBinding 35

210 4.1.5 SpecificationLink 35

211 4.1.6 ClassificationScheme 35

212 4.1.7 Classification 36

213 4.1.8 Concept 36

214 4.1.9 Association 36

215 4.1.10 RegistryPackage 37

216 4.1.11 ExternalIdentifier 37

217 4.1.12 ExternalLink 37

218 4.1.13 Slot 37

219 4.1.14 ExtensibleObject 38

220 4.1.15 AuditableEvent 38

221 4.1.16 User 38

222 4.1.17 PostalAddress 38

223 4.2 Information Model: Inheritance View 38

Sun Microsystems Page 6

JavaTM API for XML Registries April 10, 2002

224 4.2.1 RegistryEntry Interface 39

225 4.2.2 ExtrinsicObject Interface 39

226 4.3 Internationalization (I18N) Support 40

227 4.3.1 Interface InternationalString 40

228 4.3.2 Interface LocalizedString 40

229 4.4 Registry Audit Trail 40

230 **5 Classification of Registry Objects 41**

231 5.1 Interface Classification 41

232 5.2 Interface ClassificationScheme 41

233 5.3 Taxonomy Structure and Elements 42

234 5.3.1 Internal Vs. External Taxonomies 43

235 5.3.2 Internal Vs. External Classifications 43

236 5.4 Interface Concept 43

237 5.5 Internal Classification 44

238 5.5.1 An Example of Internal Classification 45

239 5.6 External Classification 45

240 5.6.1 An Example of External Classification 46

241 5.7 An Example of Multiple Classifications 46

242 5.8 Context-sensitive Classification 47

243 **6 Association of Registry Objects 50**

244 6.1 Example of an Association 50

245 6.2 Source and Target Objects 50

246 6.3 Association Types 51

247 6.4 Intramural Associations 51

248 6.5 Extramural Association 52

249 6.6 Confirmation of an Association 53

250 6.6.1 Confirmation of Intramural Associations 53

251 6.6.2 Confirmation of Extramural Associations 54

252 6.6.3 Undoing Confirmation of Extramural Associations 54

253 6.7 Visibility of Unconfirmed Associations 54

254 6.8 Possible Confirmation States 54

255 **7 Connection Management. 56**

256 7.1 Looking Up a ConnectionFactory 56

257 7.1.1 Looking Up a ConnectionFactory Using the JNDI API 56

258 7.1.2 Looking Up a ConnectionFactory Without Using the JNDI API 56

259 7.2 Setting Connection Properties on ConnectionFactory 56

260 7.2.1.1 Standard Connection Properties 57

261 7.3 Creating a JAXR Connection 58

262 7.4 Synchronous Connections 58

263 7.5 Asynchronous Connections 58

264 7.5.1 JAXRResponse and Futures Design Pattern 59

Sun Microsystems Page 7

JavaTM API for XML Registries April 10, 2002

265 7.6 Security Credentials Specification 59

266 7.7 Federated Connections 60

267 7.7.1 Creating a FederatedConnection 60

268 7.7.1.1 Connection Configuration 60

269 7.8 Concurrent Connections 60

270 7.9 Using a Connection to Access the Registry 60

271 7.10 Closing a Connection 61

272 7.11 Connection Setup Sequence 61

273 7.11.1 Connection Creation Code Sample 63

274 **8 Life Cycle Management 64**

275 8.1 Unique Key Assignment 64

276 8.2 Interface *LifeCycleManager* 65

277 8.2.1 Requests, Responses and Exception Handling 65

278 8.2.2 Creating Objects Using Factory Methods 65

279 8.2.3 Saving Objects 66

280 8.2.3.1 Interface *BulkResponse* 66

281 8.2.3.2 Interface SaveException 66

282 8.2.3.3 Implicit Saving of Objects 67

283 8.2.4 Updating Objects 67

284 8.2.5 Deleting Objects 67

285 8.2.5.1 Interface DeleteException 68

286 8.2.6 Deprecating Objects 68

287 8.2.7 Undeprecating Objects 68

288 8.3 Interface *BusinessLifeCycleManager* 69

289 8.3.1 Save Methods 71

290 8.3.2 Delete Methods 71

291 8.4 Life Cycle Management and Federated Connections 71

292 **9 Query Management 72**

293 9.1 Interface *QueryManager* 72

294 9.2 Interface *BusinessQueryManager* 73

295 9.2.1 Find Methods 75

296 9.2.1.1 Collection Parameters 76

297 9.2.1.2 Interface FindException 76

298 9.2.2 Canonical Paths Syntax for Concepts 76

299 9.2.2.1 Example of Canonical Path Representation 76

300 9.3 Interface *DeclarativeQueryManager* 77

301 9.3.1 Interface Query 77

302 9.3.2 Creating a Query 78

303 9.3.3 Executing a Query 78

304 9.4 SQL Query Syntax 78

305 9.4.1 SQL Query Syntax Binding To Information Model 78

306 9.5 OASIS ebXML Registry Filter Query Syntax 78

Sun Microsystems Page 8

JavaTM API for XML Registries April 10, 2002

307 9.6 Query Result 78

308 9.7 Federated Queries 79

309 **10 Security Architecture 80**

310 10.1 Integrity 80

311 10.2 Confidentiality 80

312 10.3 Authentication 81

313 10.3.1 Authentication Methods 81

314 10.4 Authorization 81

315 10.5 Security Support in JAXR API 82

316 10.5.1 User Registration 82

317 10.5.2 Method Connection.setCredentials 82

318 **Appendix A Pre-defined Enumerations 83**

319 A.1 Identification of Pre-defined Enumerations 83

320 A.2 Enumeration ObjectType 83

321 A.3 Enumeration PhoneType 84

322 A.4 Enumeration AssociationType 84

323 A.5 Enumeration URLType 85

324 A.6 Enumeration PostalAddressAttributes 86

325 **Appendix B Semantic Equivalence of JAXR Concepts 86**

326 **Appendix C JAXR Mapping to ebXML Registry 88**

327 C.1.1 Mapping of Interfaces 88

328 C.1.2 Mapping of New Classes In JAXR To ebXML 89

329 C.1.3 ebXML Functionality Not Supported By JAXR 89

330 **Appendix D JAXR Mapping To UDDI 89**

331 D.1 Mapping of UDDI Inquiry API Calls To JAXR 89

332 D.2 Mapping of UDDI Publisher API Calls to JAXR 90

333 D.3 Simplified UML Model For UDDI Information Model 92

334 D.4 Mapping of JAXR Attributes to UDDI 93

335 D.5 Mapping of UDDI Attributes to JAXR 93

336 D.6 Mapping of Interfaces 93

337 D.6.1 UDDI businessEntity 94

338 D.6.1.1 UDDI discoveryURL 95

339 D.6.1.1.1 Getting a discoveryURL from UDDI 95

340 D.6.1.1.2 Saving discoveryURL to UDDI 96

341 D.6.1.2 UDDI contact 96

342 D.6.1.3 UDDI address 97

343 D.6.1.3.1 Mapping of PostalAddress During Save Operations 98

344 D.6.1.3.2 Mapping of UDDI address During Find Operations 99

345 D.6.2 UDDI businessService 100

346 D.6.3 UDDI bindingTemplate 101

347 D.6.3.1 tModelInstanceInfo and instanceDetails 102

Sun Microsystems Page 9

JavaTM API for XML Registries April 10, 2002

348 D.6.4 tModel 102

349 D.6.4.1 tModel Mapping to ClassificationScheme 102

350 D.6.4.2 tModel Mapping to Concept 103

351 D.6.4.3 Mapping of tModels During JAXR Find Operations 103

352 D.6.4.4 Mapping to tModels During JAXR Save Operations 104

353 D.6.4.5 overviewDoc 105

354 D.7 Mapping of Common Data Types 105

355 D.7.1 keyedReference 105

356 D.7.2 identifierBag 107

357 D.7.3 categoryBag 107

358 D.7.4 tModelBag 107

359 D.8 Mapping of UDDI phone Element 107

360 D.8.1 Mapping of phone During Save Operations 108

361 D.8.2 Mapping of phone During Find Operations 108

362 D.9 Mapping of name to PersonName 108

363 D.10 Example of JAXR-UDDI Mapping 109

364 D.11 Provider Generated id Attributes 111

365 D.12 Supporting Taxonomy Service In JAXR UDDI Providers 112

366 D.12.1 Normative Description 112

367 D.12.2 Non-normative Description 112

368 D.13 UDDI Functionality Not Supported By JAXR 113

369 **Appendix E Value-Added Features of the JAXR API 114**

370 E.1 Taxonomy Browsing 114

371 E.2 Taxonomy Validation 114

372 E.3 Smart Queries 114

373 E.4 Enhanced Data Integrity and Validation 114

374 E.5 Automatic Categorization of UDDI tModels 115

375 E.6 Simplified Programming Model 115

376 E.6.1.1 Unification of find and get Methods 115

377 E.6.1.2 Generic Handling of Object 115

378 E.7 Simplified User Authentication 115

379 E.8 Enforce No New References to Deprecated Objects 116

380 **Appendix F Frequently Asked Questions 116**

381 **11 References 118**

382 **Table of Figures**

383 Figure 1: Interoperability between diverse JAXR clients and diverse registries 17

384 Figure 2: A Registry Use Case Scenario 19

385 Figure 3: Registry specification family tree 23

Sun Microsystems Page 10

JavaTM API for XML Registries April 10, 2002

386 Figure 4: JAXR Architecture 26

387 Figure 5: Interface *Connection* 27

388 Figure 6: Interface *RegistryService* 28

389 Figure 7: Capability-specific Interfaces 28

390 Figure 8: JAXR Provider 28

391 Figure 9: JAXR Responses and Exceptions 31

392 Figure 10: Main Interfaces defined by the JAXR API 32

393 Figure 11: Information Model Public View 34

394 Figure 12: Information Model Inheritance View 39

395 Figure 13: Classification of Registry Objects 41

396 Figure 14: Role of ClassificationSchemes in Classification 42

397 Figure 15: Role of Concepts in Representing Taxonomy Structure 44

398 Figure 16: An Example of Internal Classification 45

399 Figure 17: An Example of External Classification 46

400 Figure 18: An Example of Multiple Classifications 47

401 Figure 19: Context Sensitive Classification 48

402 Figure 20: Example of RegistryObject Association 50

403 Figure 21: Example of Intramural Association 52

404 Figure 22: Example of Extramural Association 53

405 Figure 23: Connection Setup Sequence 62

406 Figure 24: Pre-defined enumeration ObjectType 84

407 Figure 25: Pre-defined enumeration PhoneType 84

408 Figure 26: Pre-defined enumeration AssociationType 85

409 Figure 27: Pre-defined enumeration URLType 86

410 Figure 28: Pre-defined enumerations for PostalAddressAttributes 86

411 Figure 29: Simplified UML Model for UDDI Information Model 92

412 Figure 30: UDDI Information Model for Address 97

413 Figure 31: Semantic Equivalence and Mapping of User Defined Postal Scheme

414 to PostalAddress Attribute 98

415 Figure 32: Example in terms of UDDI Data Structures 110

Sun Microsystems Page 11

JavaTM API for XML Registries April 10, 2002

416 Figure 33: UDDI Example Mapped to JAXR ...........................................................110

417

Sun Microsystems Page 12

JavaTM API for XML Registries April 10, 2002

418 **Introduction**

419 **1.1 Status of this Document**

420 This specification is being developed following the JavaTM Community ProcessSM

421 (JCP SM 2.0). Comments from Experts, Participants, and the broader Java

422 Developer community have been reviewed and incorporated into this

423 specification.

424

425 This document is the JAXR Specification, version 1.0 and is the final work

426 item of the JSR093 Expert Group (EG).

427 This document has been designated as Final Release.

428 **1.2 Abstract**

429 This document defines the objectives and functionality for Java API for XML

430 Registries or JAXR.

431 Currently there are numerous overlapping specifications for business registries.

432 Examples include ISO 11179, OASIS, eCo Framework, ebXML and UDDI. JAXR

433 provides a uniform and standard API for accessing such registries within the

434 Java platform.

435 **1.3 General Conventions**

436 1. The term “*registry provider”* is used to describe implementations of business

437 registries conforming to various registry specifications and emerging

438 standards.

439 2. The term “*JAXR provider”* is used to describe implementations of the JAXR

440 API. A JAXR provider provides access to a specific registry provider or to a

441 class of registry providers that are based on a common specification.

442 3. The term “*JAXR client”* is used to describe client programs that access

443 business registries using the JAXR API.

444 4. The term *“repository item”* is used to refer to actual content (e.g. an XML

445 Schema document, as opposed to metadata about the XML Schema

446 document) submitted to a registry. The term *“repository item instance”* is used

447 to refer to a single instance of some repository item.

448 5. The term *“registry object”* is used to refer to metadata that catalogs or

449 describes a repository item. It is reflected by the RegistryObject interface in

450 the JAXR information model and its sub-interfaces.

Sun Microsystems Page 13

JavaTM API for XML Registries April 10, 2002

451 6. The verb *“catalogs”* is often used when describing metadata classes. For

452 example, the statement “Class A catalogs B” is equivalent to the statement

453 “Class A provides metadata for B”.

454 7. This document does not include the complete API documentation generated

455 by the JavadocTM software. Partial API documentation fragments are included

456 occasionally to facilitate understanding. The reader is expected to read the

457 complete API documentation as a companion to this document.

458 The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD,

459 SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in

460 this document, are to be interpreted as described in RFC 2119 [Bra97].

461 **1.4 Target Audience**

462 The target audience for this specification is the community of software

463 developers who are:

464 1. Implementers of JAXR providers

465 2. Implementers of JAXR clients

466 **1.5 JAXR Expert Group**

467 The JAXR specification is the result of a collaborative effort and collective

468 wisdom of the JSR093 Expert Group and the companies and individuals who

469 have supported this work with their participation in the Java Community.

470

471 Joseph Baran - Extol, Inc.

472 Ben Bernhard - IONA

473 Marco Carrer - Oracle

474 Alex Ceponkus - Bowstreet

475 Joel Farrell - IBM Corporation

476 Tom Gaskins - Hewlett-Packard Company

477 Wooyoung Kim - Individual

478 Amelia A. Lewis - Tibco Extensibility Inc.

479 Sam Lee - Oracle

480 Dale Moberg - Cyclone Commerce

481 Farrukh Najmi - Sun Microsystems

482 Eric Newcomer - IONA Technologies

483 Sanjay Patil - IONA Technologies

484 Will Raymond - Tibco Extensibility Inc.

485 Waqar Sadiq - EDS

486 Krishna Sankar - CISCO

487 Nikola Stojanovic - Encoda Systems, Inc.

488 Omar Tazi - webGain

Sun Microsystems Page 14

JavaTM API for XML Registries April 10, 2002

489 Ravi Trivedi - Hewlett-Packard Company

490 Lyndon Washington - SilverStream

491 Prasad Yendluri - webMethods Corporation

492 Phil Zimmerman - BEA Systems

493 **1.6 Acknowledgements**

494 Graham Hamilton, Mark Hapner, Eduardo Pelegri-Llopart, Bill Shannon, Robert

495 Bissett, Irene Caruso, Peter Eakle, Joe Fialli, Maydene Fisher, Kim Haase, Steve

496 Hanna, Peter Kacandes, Nandkumar Kesavan, Tom Kincaid, Ramesh Mandava,

497 Bhakti Mehta, Ron Monzillo, Kevin Osborn, Cecilia Peltier, Karen Schaffer, Leslie

498 Schwenk, Karen Shipe, Christine Tomlinson, Sekhar Vajjhala, Peter Walker,

499 Kathy Walsh, (all from Sun Microsystems) have all made invaluable contributions

500 to the JAXR 1.0 specification. Thanks to Jeff Jackson, Karen Tegan and Connie

501 Weiss for their sponsorship and support of JAXR.

502 **1.7 Relationship to Other Java APIs**

503 JAXR is related to several other Java APIs for XML. In future, these APIs may

504 become part of the Java 2 Platform, Enterprise Edition (J2EETM platform).

505 **1.7.1 JAXP**

506 Java API for XML Processing or JAXP enables flexible XML processing from

507 within Java programs.

508 The JAXR API will make direct XML processing less important for JAXR clients.

509 However, JAXP may be used by implementers of JAXR providers and JAXR

510 clients for processing XML content that is submitted to or retrieved from the

511 registry. The JAXP API is likely to also be used in implementations of the JAXB

512 API described next.

513 **1.7.2 JAXB**

514 Java API for XML Binding or JAXB enables simplified XML processing using

515 Java classes that are generated from XML schemas.

516 The JAXR API will make direct XML processing less important for JAXR clients.

517 However, JAXB may be used by implementers of JAXR providers and JAXR

518 clients for processing XML content that is submitted to or retrieved from the

519 registry.

Sun Microsystems Page 15

JavaTM API for XML Registries April 10, 2002

520 **1.7.3 JAX-RPC**

521 Java API for XML-based RPC or JAX-RPC provides an API for XML-based RPC

522 communication in the Java platform.

523 Implementations of the JAXR providers may use JAX-RPC for communication

524 between JAXR providers and registry providers that export a SOAP-based RPC

525 like interface (e.g. UDDI).

526 **1.7.4 JAXM**

527 Java API for XML Messaging or JAXM provides an API for packaging and

528 transporting of message based business transactions using on-the-wire protocols

529 defined by emerging standards.

530 Implementations of the JAXR providers may use JAXM for communication

531 between JAXR providers and registry providers that export an XML Messaging

532 based interface (e.g. ebXML TRP).

533 **1.8 Design Objectives**

534 This section describes the high level design objectives for the JAXR API.

535 **1.8.1 Goals**

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

537 1. Define a general purpose Java API for accessing business registries that

538 allows any JAXR client to access and interoperate with any business registry

539 that is accessible via a JAXR provider.

540 2. Define a pluggable provider architecture that enables support for diverse

541 registry specifications and standards.

542 3. Support a union of the best features of dominant registry specifications rather

543 than a common intersection of features. *JAXR is not a least common*

544 *denominator API*.

545 4. Ensure support for dominant registry specifications such as ebXML and UDDI,

546 while maintaining sufficient generality to support other types of registries,

547 current or future.

548 5. Ensure synergy with other Java specifications related to XML.

549

550 Figure 1 below shows how diverse JAXR clients can interoperate with diverse

551 registries using the JAXR API.

Sun Microsystems Page 16

JavaTM API for XML Registries April 10, 2002

552

**Desktop Applications**

**J2EE**

**Components**

**Registry Browsers**

**Other**

**UDDI**

**ebXML**

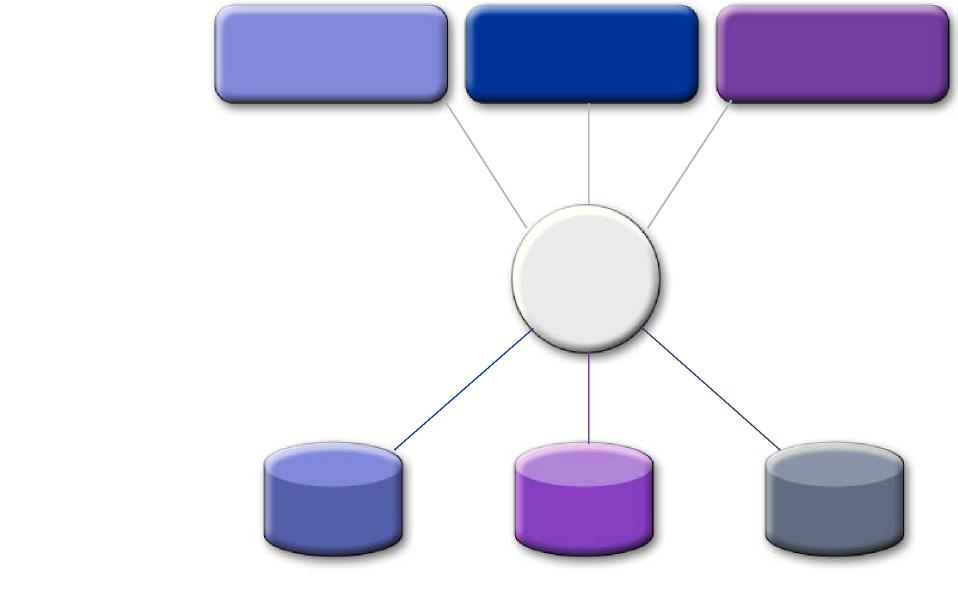
**Diverse Registries**

**JAXR API**

**Diverse Clients**

554 **Figure 1: Interoperability between diverse JAXR clients and diverse registries**

553



555 **1.8.2 Non Goals**

556 This specification does not aim to define either business registry standards or

557 XML messaging standards. These standards belong in standards bodies such as

558 OASIS, W3C or IETF. Instead, this specification aims to define standard Java

559 APIs to allow convenient access from Java to emerging registry standards.

560 **1.9 Caveats and Assumptions**

561 It is assumed that:

562 1. The reader is familiar with UML notation. UML notation is used throughout

563 this document for most of the diagrams.

564

Sun Microsystems Page 17

JavaTM API for XML Registries April 10, 2002

564 **2 Overview**

565 **2.1 What Is a Registry**

566 Most business-to-business (B2B) interactions are based on a collaborative

567 process between 2 parties that are engaged in a partnership. A registry is a

568 neutral 3rd party that helps facilitate such collaboration. A registry is available to

569 organizations as a shared resource often in the form of a web based service. A

570 registry is a key component in any Web Services architecture because it provides

571 organizations with the ability to publish, discover and utilize web services.

572 Registries enable dynamic and loosely coupled B2B collaboration.

573 [Note] While this document may present registry use

574 cases in a business-focused context, the JAXR

575 API is sufficiently general to support many

576 other types of use cases.

577 **2.2 Registry Use Case Scenarios**

578 Figure 2 below illustrates a few of the common use case scenarios involving a

579 business registry. The scenario shows how a registry facilitates a buyer company

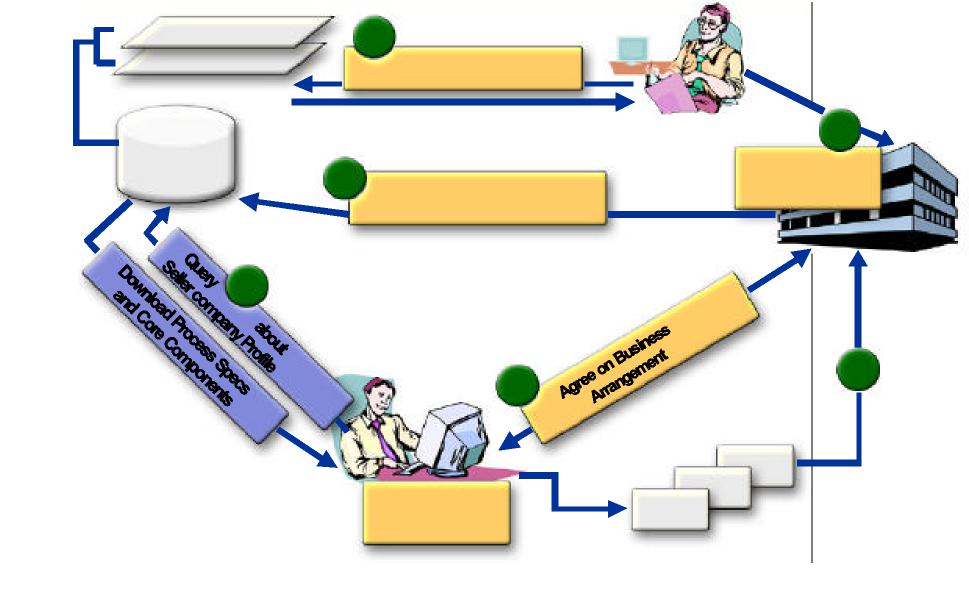
580 discovering a seller company and engaging in a collaborative B2B process.

581

Sun Microsystems Page 18

JavaTM API for XML Registries April 10, 2002

**Register Implementation Details Register Seller Company Profile**



582

583 **Figure 2: A Registry Use Case Scenario**

**XML**

**1**

**Seller Company**

**2**

**ebXML Registry**

**3**

Build Local

System

**Implementation**

**4**

**5**

**6**

**Buyer Company**

ebXML

Compliant

**System**

**Business Process Spes Core Components**

**Request Business Details**

584 1. First, the Seller company queries a registry for specifications defining a

585 collaborative business process as well as core components that define

586 reusable XML elements used in business documents (e.g. Address,

587 Contact etc.). These specifications have previously been submitted by a

588 vertical standards organization.

589 2. The Seller then uses the specifications and core components downloaded

590 from the registry to implement their local eBusiness system with support

591 for the desired collaborative processes.

592 3. The Seller then registers information about their company, their products

593 and their services in the registry. Such information may be classified to

594 facilitate discovery by potential buyers.

595 4. A Buyer company may browse the registry by classifications etc. and

596 discover the Seller. They may also download technical specifications and

597 core components to implement their local system to support the

598 collaborative process.

599 5. The Buyer then negotiates with the Seller on an agreement to collaborate

600 in the chosen collaborative process implemented and agreed to by both

601 sides.

Sun Microsystems Page 19

JavaTM API for XML Registries April 10, 2002

602 6. The two parties finally engage in the desired collaborative business

603 process and exchange business documents.

604 **2.3 Participant Roles**

605 This section describes the key roles played by participants (actors) within various

606 registry use case scenarios. This section is not a complete list of roles.

607 **2.3.1 Submitting Organization**

608 A submitting organization (SO) is an organization that submits or publishes

609 content to a registry. An SO may be an enterprise or an entity within an

610 enterprise.

611 An SO owns the content that it publishes to a registry.

612 **2.3.2 Content Submitter**

613 A content submitter is a user who belongs to a submitting organization and is

614 authorized to submit content on behalf of the organization.

615 **2.3.3 Registry Operator**

616 A registry operator is responsible for operating a registry. A registry operator has

617 special access control and authorization privileges within the registry under their

618 operation.

619 **2.3.4 Registry Guest**

620 A registry guest is a non-privileged casual user of the registry who simply

621 browses the data within the registry.

622 **2.4 Registry Vs. Repository**

623 The terms registry and repository are often used together and sometimes

624 confused with each other.

625 The following sections describe the distinction between a registry and a

626 repository and introduce the content of each.

Sun Microsystems Page 20

JavaTM API for XML Registries April 10, 2002

627 **2.4.1 Repository and Repository Items**

628 Information published by an SO to a registry is stored in a stable store called a

629 repository. The registry maintains the repository. The repository is the holder of

630 content (e.g. DTDs, XML Schemas, WSDL documents etc.) submitted by an SO

631 to a registry. Instances of the content stored in the repository are called

632 *repository items*.

633 The JAXR API does not directly provide access to the repository. Instead, all

634 access to the repository is through the registry. As such, the repository is an

635 implementation detail of a registry. It is mentioned in this specification only as a

636 concept. However, neither the repository nor repository items are part of the

637 JAXR information model or API.

638 **2.4.2 Registry and Registry Objects**

639 When an SO submits repository items using the JAXR API, it also provides

640 additional metadata that *catalogs* or *describes* the repository items. Such

641 metadata is referred to as *registry objects* in the JAXR information model.

642 In summary, a repository is a holder of submitted content while a registry is a

643 catalog that describes the submitted content in the repository. It should be noted

644 that not all registries include the repository functionality.

645 **2.5 Functionality of a Registry**

646 This section describes the functionality that is provided by a registry.

647 **2.5.1 Registry as Electronic Yellow Pages**

648 Registries facilitate the creation of business relationships by providing an

649 independent online information exchange service that allows service providers

650 (e.g. sellers) to advertise their products and services, and service consumers

651 (e.g. buyers) to discover these products and services. Such an information

652 exchange service is sometimes referred to as “electronic yellow pages”.

653 **2.5.1.1 Flexible Classification Capability**

654 Registries provide a rich classification capability that allows content providers to

655 classify content such as organization and service descriptions in arbitrary and

656 flexible ways. For example, content submitted to the registry may be the

657 description of a business organization that is classified by the *industry* it belongs

658 to, the *geography* it is located in, the *business processes* it supports, and the

659 *products* it sells.

Sun Microsystems Page 21

JavaTM API for XML Registries April 10, 2002

660 Such flexible classification capabilities of registries facilitate discovery of content

661 by interested parties.

662 **2.5.2 Registry as a Database of Relatively Static Data**

663 A registry (and its repository) stores metadata and data. As such, it is much like a

664 database. It stores information about:

665 o Collaborative business process descriptions that describe in XML form a

666 specific business protocol (e.g. RosettaNet PIP3A4 for purchase orders)

667 o Parties in a collaborative business process

668 o XML Schemas that define the structure of the XML documents exchanged

669 during a collaborative Business Process

670 A registry plays a role in B2B applications that is similar to that played by

671 databases in enterprise applications; it provides a way for applications to store

672 relatively static information reliably and to enable sharing of such information.

673 **2.5.3 Registry as Electronic Bulletin Board**

674 A registry may also provide means to exchange dynamic content between

675 parties. Examples include generic event notification, price changes, discounts,

676 promotions etc. Such dynamic capabilities allow for more Just-In-Time B2B

677 partnerships.

678 **2.6 Existing Registry Specifications**

679 Currently there are numerous registry specifications. Examples include OASIS,

680 eCo Framework, ebXML and UDDI. While there may be some similarity between

681 these specifications, in general these are diverse specifications.

682 JAXR API aims to be the confluence of the various registry specifications as

683 shown in Figure 3 below:

Sun Microsystems Page 22

JavaTM API for XML Registries April 10, 2002

**eCo Framework**

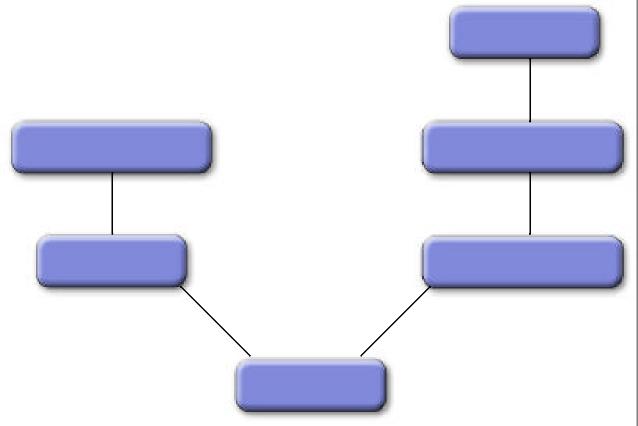
**UDDI ebXML Registry**

**JAXR**

**OASIS Registry**

**ISO 11179**

684



685 **Figure 3: Registry specification family tree**

686 **2.7 Registry Provider**

687 A registry provider provides an implementation of a registry specification or

688 standard. Examples include:

689 o A UDDI registry provider that implements the UDDI registry specifications

690 o An ebXML registry provider that implements the ebXML Registry

691 specifications

692 A registry provider is not required to implement the JAXR specification.

693 **2.8 JAXR Provider**

694 A JAXR Provider provides an implementation of the JAXR specification.

695 Typically, a JAXR provider is implemented to access an existing registry

696 provider.

697 **2.9 JAXR Client**

698 A JAXR client is a Java program that uses the JAXR API to access the services

699 provided by a JAXR provider.

700 **2.10 Support for Multiple Registry Specifications**

701 The JAXR API must support diverse registry specifications that vary significantly

702 in their capabilities and underlying information model.

Sun Microsystems Page 23

JavaTM API for XML Registries April 10, 2002

703 JAXR aims to support diverse registry specifications by providing a broad and

704 capable API rather than a “least-common-denominator” API. To use a Java

705 analogy, the JAXR philosophy is akin to designing a Java Virtual Machine that is

706 a union of the most useful capabilities of all supported hardware platforms rather

707 than an intersection of their capabilities. This approach to supporting diverse

708 registry specifications means that not all Registries are able to support all

709 aspects of the JAXR API.

710 **2.10.1 Capability Profiles**

711 The JAXR API categorizes its API methods by a small number of capability

712 profiles. Currently only two capability profiles are defined (level 0 and level1).

713 **2.10.1.1 Assignment of Capability Level to Methods**

714 Each method in the JAXR API is assigned a capability level. The capability level

715 is defined in the API documentation for each method in a class or interface in the

716 JAXR API.

717 **2.10.1.2 Assignment of Capability Level to Interfaces and Classes**

718 There is no assignment of capability level to interfaces and classes in the JAXR

719 API. Capability assignment is done at the method level only.

720 **2.10.1.3 Declaration of Capability Level by a JAXR Provider**

721 A JAXR provider must declare the capability level for its implementation of the

722 JAXR API. A JAXR client may discover a JAXR provider’s capability level by

723 invoking methods on an interface named CapabilityProfile as defined by the

724 JAXR API. If a JAXR provider declares support for a specific capability level then

725 it implicitly declares support for lower capability levels. For example, a JAXR

726 provider that declares support for the level 1 profile implicitly declares support for

727 level 0 profile.

728 A JAXR provider must implement the functionality described by the JAXR API for

729 each method that is assigned a capability level that is less than or equal to the

730 capability level declared by the JAXR provider.

731 A JAXR provider must implement all methods that are assigned a capability level

732 that is greater than the capability level declared by the JAXR provider, to throw

733 an UnsupportedCapabilityException. A JAXR provider must never implement any

734 other behavior for methods assigned a greater than the capability level declared

735 by the JAXR provider. The reason for this restriction is that it is necessary to

736 ensure portable behavior for JAXR clients for any JAXR provider within a specific

737 capability level.

Sun Microsystems Page 24

JavaTM API for XML Registries April 10, 2002

738 **2.10.2 Level 0 Profile**

739 Support for the level 0 profile is required to be supported by all JAXR providers.

740 The methods assigned to this profile provide the most basic registry capabilities.

741 **2.10.3 Level 1 Profile**

742 Support for the level 1 profile is optional for JAXR providers. The methods

743 assigned to this profile provide more advanced registry capabilities that are

744 needed by more demanding JAXR clients. Support for the Level 1 profile also

745 implies full support for the Level 0 profile.

746 **2.10.4 Capability Level and JAXR Clients**

747 A JAXR client may be written to use only those methods that are assigned a level

748 0. Such a client is able to access any JAXR provider in a portable manner.

749 An advanced JAXR client may also be written to methods that are assigned a

750 level 1. This level 1 client is able to access only level 1 compliant JAXR providers

751 in a portable manner.

752 **2.11 Capability Levels and Registry Standards**

753 JAXR providers for UDDI must be level 0 compliant. JAXR providers for ebXML

754 must be level 1 compliant.

755

Sun Microsystems Page 25

JavaTM API for XML Registries April 10, 2002

755 **3 Architecture**

756 This chapter describes the high-level architecture for JAXR and introduces some

757 of the core interfaces.

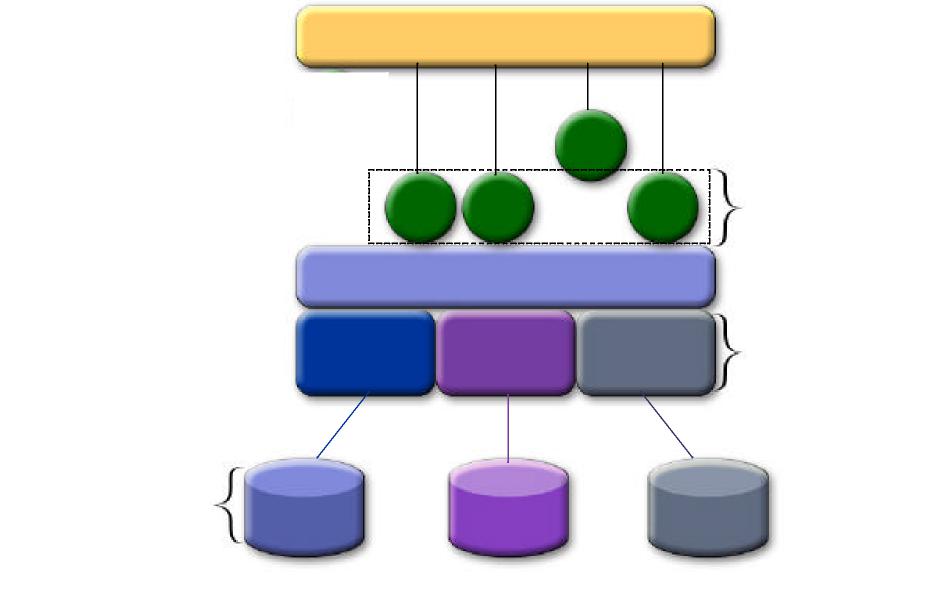
758 Figure 4 below shows the high-level view of the JAXR architecture. It is

759 frequently referred to, within this specification.

763 The circles represent the various interfaces implemented by the JAXR client and

**C1 C2 Cn**

760



761

762 **Figure 4: JAXR Architecture**

**Diverse**

**Registries ebXML**

**ebXML/SOAP UDDI/**

**ebXML Provide**

**JAXR Pluggable Provider**

**r**

**JAXR Client**

**UDDI Provide**

**UDDI Other**

**r**

**r**

**SOAP ???**

**RS**

**Other Provide**

**Registry Specific JAXR Provider**

**Capability Specific Interfaces**

764 the JAXR provider:

765 o RS represents the RegistryService interface implemented by the

766 JAXR provider.

767 o C1, C2 through Cn represent the JAXR interfaces implemented by the

768 JAXR provider that provide the various registry capabilities. These

769 interfaces are introduced later in this specification.

770 [Note] The JAXR client and the JAXR provider are

771 expected to be co-located within the same JVM

772 process in most implementations. The only

Sun Microsystems Page 26

JavaTM API for XML Registries April 10, 2002

773 distribution point over the web is between the

774 JAXR provider and the registry provider

775 (between the middle and bottom tiers).

776 In the following sections, each component of the JAXR architecture is described

777 in a top-down order. Most descriptions are accompanied by a thumbnail version

778 of Figure 4. The thumbnail figure highlights the component being described by

779 pointing to it with a block arrow and showing it in red color.

780 **3.1 JAXR Client**

781 At the top of Figure 4 is the JAXR client. The JAXR client may be any standalone

782 Java application or an enterprise component based on J2EE technology. The

783 JAXR client uses the JAXR API to access a registry via a JAXR provider.

784 ***3.2* Interface *Connection***

|  |  |
| --- | --- |
| 785 |  |

786 Figure 5: Interface *Connection*

787 A Connection object (pointed to by block arrow in Figure 5) represents a client

788 session with a registry provider using a JAXR provider. It maintains state

789 information for a specific connection.

790 A client must create a JAXR Connection to a registry provider using an

791 appropriate JAXR provider in order to employ the services of that registry using

792 the JAXR API. Chapter 7 describes the role of Connections in further detail.

793 The client uses the JAXR ConnectionFactory interface to create a Connection.

794 The JAXR Connection is not explicitly shown in Figure 4. However, the

795 RegistryService interface defined by the JAXR API is contained within a JAXR

796 Connection.

797 1. The Connection interface provides various setter methods (e.g.

798 setSynchronous, setCredentials) that allow the JAXR client to dynamically

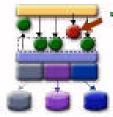
799 alter its state, context, and preferences with the JAXR provider at any time.

Sun Microsystems Page 27

JavaTM API for XML Registries April 10, 2002

800 ***3.3* Interface *RegistryService***

801



802 Figure 6: Interface *RegistryService*

803 The RegistryService interface (pointed to by block arrow in Figure 6) is the

804 principal interface implemented by a JAXR provider. A registry client can get this

805 interface from its Connection to a JAXR provider.

806 The RegistryService interface provides the getter methods that are used by

807 the client to discover various capability-specific interfaces implemented by the

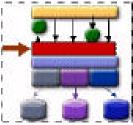
808 JAXR provider. It also provides a getCapabilityProfile method that allows

809 the JAXR client to access the capability profile that describes the capabilities

810 supported by the JAXR provider.

811 **3.4 Capability-specific Interfaces**

812



813 Figure 7: Capability-specific Interfaces

814 The capability-specific interfaces (pointed by block arrow in Figure 7) provide

815 specific capabilities such as:

816 o Life cycle management, which is discussed in Chapter 8.

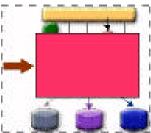
817 o Query management, which is discussed in Chapter 9.

818 Capability specific interfaces are usually named xxManager where xx represents

819 the specific capability provided by that interface.

820 **3.5 The JAXR Provider**

821



822 Figure 8: JAXR Provider

Sun Microsystems Page 28

JavaTM API for XML Registries April 10, 2002

823 The JAXR provider (pointed to by block arrow in Figure 8) is an implementation

824 of the JAXR API. A JAXR client accesses a registry via a JAXR provider.

825 Figure 4 shows the JAXR provider as the union of the JAXR pluggable provider

826 and the registry-specific JAXR providers.

827 [Note] The following sections describing JAXR

828 Pluggable provider, registry specific providers

829 and JAXR bridge providers are non-normative.

830 They describe an architectural vision for the

831 JAXR API. However, it should be noted that the

832 JAXR API does not define a service provider

833 interface (SPI) for plugging registry specific

834 providers into a JAXR pluggable provider. This

835 SPI is deferred to a future release of the JAXR

836 specification.

837 **3.5.1 JAXR Pluggable Provider**

838 The JAXR Pluggable provider implements features of the JAXR API that are

839 independent of any specific registry type. The Pluggable provider provides a

840 single abstraction for multiple registry-specific JAXR providers. It allows the client

841 to avoid being exposed to the fact that there are multiple registry-specific JAXR

842 providers performing the actual registry access.

843 An important feature of the JAXR Pluggable provider is providing a Pluggable

844 ConnectionFactory implementation that can create JAXR Connections using the

845 appropriate registry-specific JAXR provider.

846 **3.5.2 Registry-specific JAXR Provider**

847 The registry-specific JAXR providers implement the JAXR API in a registry-

848 specific manner. A registry-specific JAXR provider plugs into the JAXR Pluggable

849 provider and is used by the JAXR Pluggable provider in a delegation pattern. The

850 contract between the JAXR Pluggable provider and a registry-specific JAXR

851 provider is currently not defined. It will be defined in a future version of this

852 specification.

853 A registry-specific JAXR provider accepts JAXR requests from the client and

854 transforms them into equivalent requests based on the specifications of the

855 target registry. It dispatches the registry-specific requests to the registry provider

856 using registry-specific protocols.

Sun Microsystems Page 29

JavaTM API for XML Registries April 10, 2002

857 A registry provider processes a request from the registry-specific JAXR provider

858 and sends back a registry-specific response. The registry-specific JAXR provider

859 then transforms the registry-specific response into an equivalent JAXR response

860 that is delivered to the JAXR client.

861 From the registry provider’s perspective, its client is the registry-specific JAXR

862 provider.

863 **3.5.3 JAXR Bridge Providers**

864 A JAXR Bridge Provider is a type of registry-specific JAXR Provider.

865 It is likely that most registry-specific JAXR providers will be developed as bridges

866 to existing registry providers. Such JAXR providers are referred to as JAXR

867 Bridge providers. A bridge provider is not specific to any particular registry

868 instance. Instead, a bridge provider is specific to a class of registries (e.g. OASIS

869 ebXML registry, UDDI registry) and may be used to access any registry instance

870 that is compliant with the specification that defines that class of registries.

871 For example, an ebXML Bridge provider gives access to any ebXML compliant

872 registry implementation, while a UDDI bridge provider gives access to any UDDI

873 compliant registry implementation.

874 **3.6 Registry Provider**

875 Registry providers are shown as the bottom layer in Figure 4. These are

876 implementations of various registry specifications such as ebXML and UDDI.

877 **3.7 JAXR API Package Structure**

878 The JAXR API is divided into two main packages:

879 1. The javax.xml.registry.infomodel package consists of interfaces

880 that define the information model for JAXR. These interfaces define the

881 types of objects that reside in a registry and how they relate to each other.

882 The information model is discussed in detail in Chapter 4.

883 2. The javax.xml.registry package consists of the interfaces and

884 classes that define the registry access interface.

885 While the information model describes what types of objects reside in the

886 registry, the access interfaces in javax.xml.registry package define how

887 those objects are submitted to the registry and subsequently managed. Figure 10

888 shows the interfaces and classes defined by the JAXR API as defined by the

889 java.xml.registry package. The information model interfaces are described

890 in detail in Chapter 4.

Sun Microsystems Page 30

JavaTM API for XML Registries April 10, 2002

891 **3.7.1 Responses and Exceptions**

892 Figure 9 shows the various interfaces and classes defined by the JAXR API for

893 communicating Exceptions and responses. The RegistryException class is the

894 common base class for all JAXRExceptions that represent exceptions and errors

895 that occurred on the registry provider side rather than the JAXR provider side

896 during a JAXR API call.

|  |  |
| --- | --- |
| 897 |  |

898 **Figure 9: JAXR Responses and Exceptions**

899 **3.7.2 Main Interfaces**

900 Figure 10 shows the main interfaces defined by the JAXR API. Part of the API is

901 identified as the *Query API* while part of it is identified as *Life Cycle Management*

902 *API*. Also part of the API is identified as providing a Business focused API while

903 part of it is identified as providing a more generic API. Details of these interfaces

904 will be provided later in this specification or in the API documentation for the

905 JAXR API.

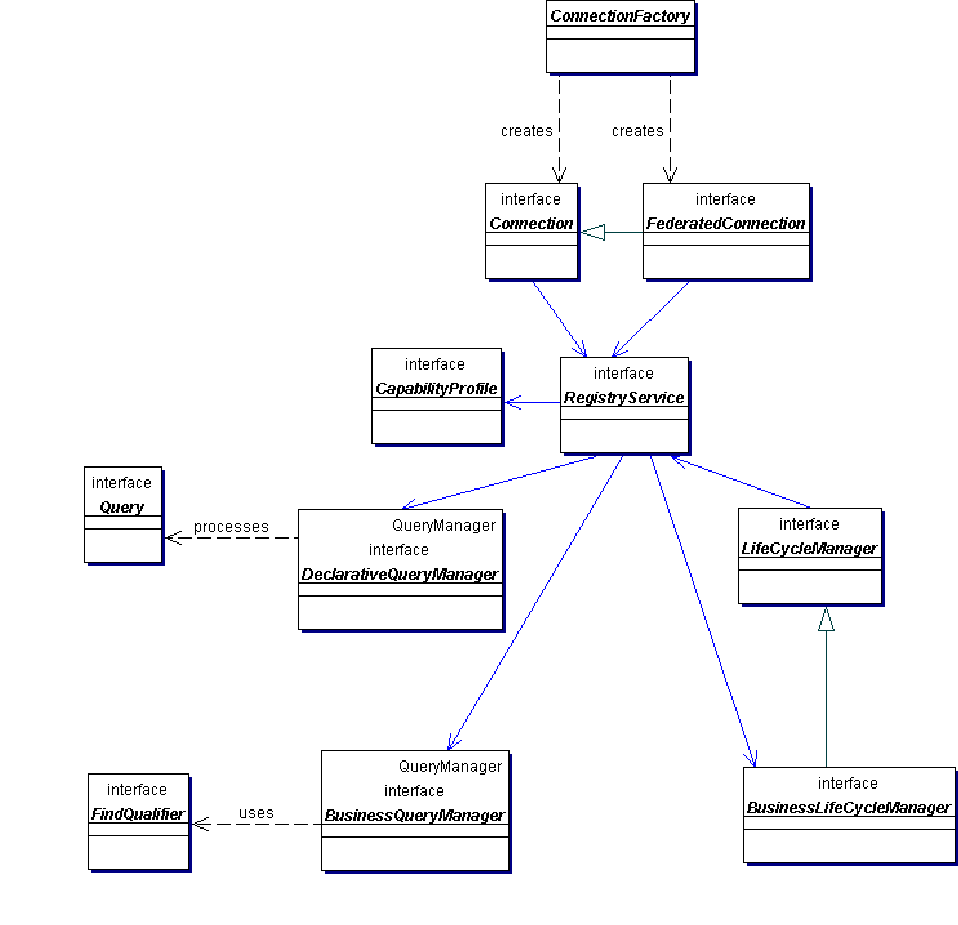
Sun Microsystems Page 31

JavaTM API for XML Registries April 10, 2002

Sun Microsystems Page 32

908

909



906

907 **Figure 10: Main Interfaces defined by the JAXR API**

JavaTM API for XML Registries April 10, 2002

909 **4 Information Model**

910 The JAXR information model is largely based on the ebXML Registry Information

911 model as defined by [RIM] and extended to add concepts borrowed from UDDI

912 as defined by [UDDI-DS]. A normative binding to both [RIM] and [UDDI-DS] is

913 defined in Appendix C and Appendix D.

914 The information model related interfaces are defined in the JAXR package

915 java.xml.registry.infomodel. These interfaces may be viewed as

916 providing a simple Java binding to a unified information model from the dominant

917 registry specifications. The JAXR information model is the confluence of these

918 registry specifications.

919 **4.1 Information Model: Public View**

920 This section provides a high-level public view of the most visible objects in the

921 registry.

922 Figure 11 shows the public view of the objects in the registry and their

923 relationships as a UML class diagram. It does not show inheritance, class

924 attributes or class methods.

925 The reader is reminded that the information model does not model actual

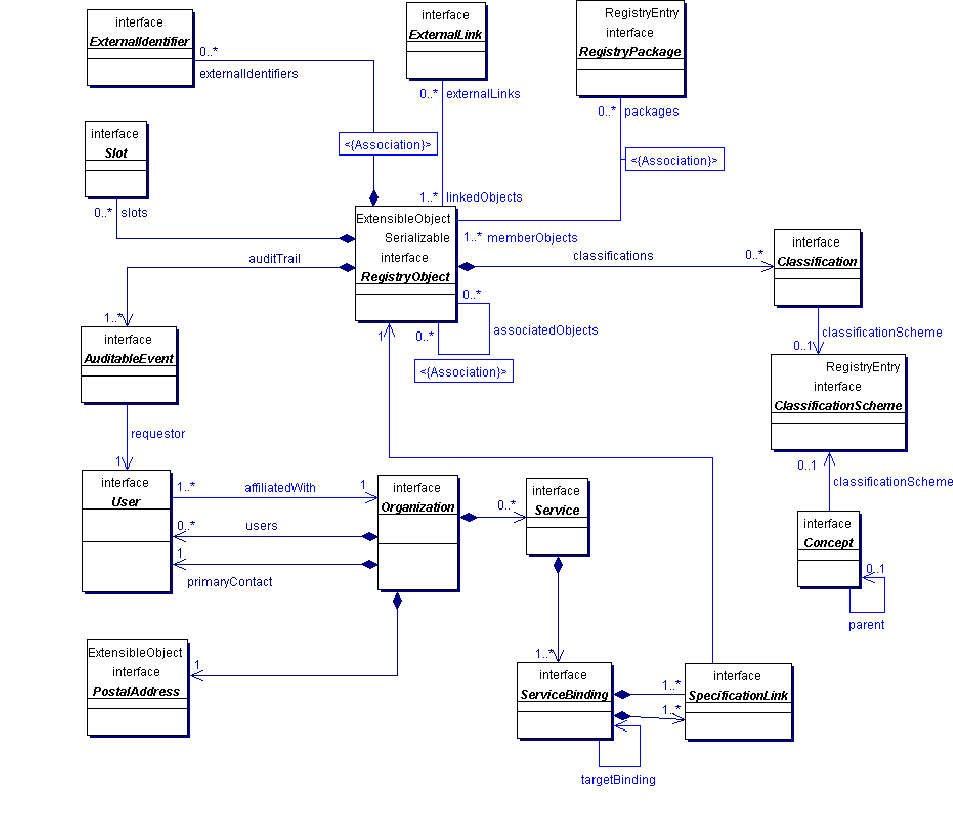
926 repository items.

927

Sun Microsystems Page 33

JavaTM API for XML Registries April 10, 2002

930 The following sections provide high-level information on the information model



928

929 **Figure 11: Information Model Public View**

931 interfaces. More background and detail may be found in the API documentation.

932 **4.1.1 RegistryObject**

933 The RegistryObject class is an abstract base class used by most classes in the

934 model. It provides minimal metadata for registry objects. It also provides methods

935 for accessing related objects that provide additional dynamic metadata for the

936 registry object.

Sun Microsystems Page 34

JavaTM API for XML Registries April 10, 2002

937 **4.1.2 Organization**

938 Organization instances are RegistryObjects that provide information on

939 organizations such as a Submitting Organization. Each Organization instance

940 may have a reference to a parent Organization. An Organization may have a set

941 of Service instances.

942 **4.1.3 Service**

943 Service instances are RegistryObjects that provide information on services (e.g.

944 web services) offered by an Organization. A Service may have a set of

945 ServiceBinding instances.

946 **4.1.4 ServiceBinding**

947 ServiceBinding instances are RegistryObjects that represent technical

948 information on a specific way to access a specific interface offered by a Service

949 instance. A ServiceBinding may have a set of SpecificationLink instances.

950 **4.1.5 SpecificationLink**

951 A SpecificationLink provides the linkage between a ServiceBinding and one of its

952 technical specifications that describes how to use the service using the

953 ServiceBinding. For example, a ServiceBinding may have a SpecificationLink

954 instance that describes how to access the service using a technical specification

955 in the form of a WSDL document or a CORBA IDL document.

956 **4.1.6 ClassificationScheme**

957 A ClassificationScheme instance represents a taxonomy that may be used to

958 classify or categorize RegistryObject instances.

959 A very common example of a classification scheme in science is the

960 Classification of living things where living things are categorized in a tree-like

961 structure. Another example is the Dewey Decimal system used in libraries to

962 categorize books and other publications. ClassificationScheme is described in

963 detail in Chapter 5. A common example in eBusiness is the North American

964 Industry Classification System (NAICS), which is a classification scheme used to

965 classify businesses and services by the industry to which they belong.

Sun Microsystems Page 35

JavaTM API for XML Registries April 10, 2002

966 **4.1.7 Classification**

967 Classification instances are used to classify a RegistryObject instance using a

968 classification scheme. The ability to classify RegistryObjects in a registry is one

969 of the most significant features provided by a registry. This is because

970 classification facilitates rapid discovery of RegistryObjects within the registry.

971 Classification is described in detail in chapter 5.

972 **4.1.8 Concept**

973 A Concept instance represents an arbitrary notion (or concept). It can be virtually

974 anything. While concepts may be used for many purposes, the following list

975 summarizes some of the main uses of Concepts at this time:

976 1. Concepts may be used to define the hierarchical tree structure and

977 detailed elements of a classification scheme as described earlier in

978 Section 4.1.6. The root of the tree structure is defined by the

979 ClassificationScheme instance while descendent nodes in the tree

980 structure are Concept instances. This use is described in detail in section

981 5.

982 2. Concepts may be used to define extensible enumerations for use in

983 values for certain attributes (e.g. objectType attribute in RegistryObject).

984 This is essentially a special case of defining the structure of a

985 classification scheme. This use is described in Appendix A.

986 3. Concepts may be used to serve as a proxy for content that is external to a

987 level 0 registry by providing a unique ID for the external content. This is

988 similar to the role played by tModels in UDDI when used for the purposes

989 of providing a technical finger print for content external to the UDDI

990 registry such as a WSDL document.

991 4. Concepts may be used to define namespaces for external identifiers such

992 as DUNS.

993 **4.1.9 Association**

994 Association instances are used to define many-to-many associations between

995 objects in the information model. Associations are described in detail in chapter

996 6.

Sun Microsystems Page 36

JavaTM API for XML Registries April 10, 2002

997 **4.1.10 RegistryPackage**

998 RegistryPackage instances are used to group logically related RegistryObjects

999 together. A RegistryPackage may contain any number of RegistryObjects. A

1000 RegistryObject may be a member of any number of RegistryPackages.

1001 **4.1.11 ExternalIdentifier**

1002 ExternalIdentifier instances provide identification information to a RegistryObject.

1003 Such identification may be based on well-known identification schemes such as 1004 DUNS number and Social Security Number. Such identification may also be 1005 based on proprietary identification schemes. The JAXR information model reuses 1006 the ClassificationScheme class for representation of identification schemes.

1007 **4.1.12 ExternalLink**

1008 ExternalLink instances provide a link to content that is managed outside the

1009 registry using a URI to the external content. This URI is contained within the

1010 externalURI attribute of ExternalLink. Unlike content managed in the repository,

1011 such external content may change or be deleted at any time without the

1012 knowledge of the registry. A RegistryObject may be associated with any number

1013 of ExternalLinks.

1014 Consider the case where a Submitting Organization submits a repository item 1015 (e.g. a WSDL document) and wants to associate some external content with that 1016 object (e.g. the Submitting Organization's home page). The ExternalLink enables 1017 this capability. A potential use of the ExternalLink capability may be in a GUI tool 1018 that displays the ExternalLinks defined for a RegistryObject. The user may click 1019 on such links and navigate to an external web page referenced by the link.

1020 When a JAXR client sets the externalURI attribute in an ExternalLink, either by a

1021 LifeCycleManager.createExternalLink call, or the ExternalLink.setExternalURI

1022 call, the JAXR provider must check if it is an HTTP URL. If so, the provider must 1023 validate that the HTTP URL points to a valid and accessible resource. If the 1024 HTTP URL is found to be invalid or inaccessible, the JAXR provider must throw 1025 an InvalidRequestException.

1026 **4.1.13 Slot**

1027 Slot instances provide a dynamic way to add arbitrary attributes to

1028 RegistryObject instances at runtime. This ability to add attributes dynamically to

1029 RegistryObject instances enables extensibility within the information model.

Sun Microsystems Page 37

JavaTM API for XML Registries April 10, 2002

1030 **4.1.14 ExtensibleObject**

1031 The interface ExtensibleObject is extended by most interfaces in the JAXR

1032 information model. It provides methods that allow the addition, deletion and

1033 lookup of Slot instances. The ExtensibleObject interface provides extensibility to 1034 the JAXR information model.

1035 **4.1.15 AuditableEvent**

1036 AuditableEvent instances are RegistryObjects that are used to provide an audit

1037 trail for RegistryObjects. AuditableEvent is described in detail in section 4.2.1.

1038 **4.1.16 User**

1039 User instances are RegistryObjects that are used to provide information about

1040 registered users within the registry. Each User is affiliated with an Organization.

1041 User objects are used i n the audit trail for a RegistryObject.

1042 **4.1.17 PostalAddress**

1043 PostalAddress defines attributes of a postal address. Currently, it is used to 1044 provide address information for a User and an Organization.

1045 **4.2 Information Model: Inheritance View**

1046 Figure 12 shows the inheritance or “is a” relationships between the classes in the 1047 information model. Note that it does not show the other types of relationships, 1048 such as “has a” relationships, since they have already been shown in Figure 11. 1049 Class attributes and class methods are also not shown. Detailed descriptions of 1050 methods and attributes of most interfaces and classes are available in the JAXR 1051 API documentation.

1052 The reader is again reminded that the information model does not model actual

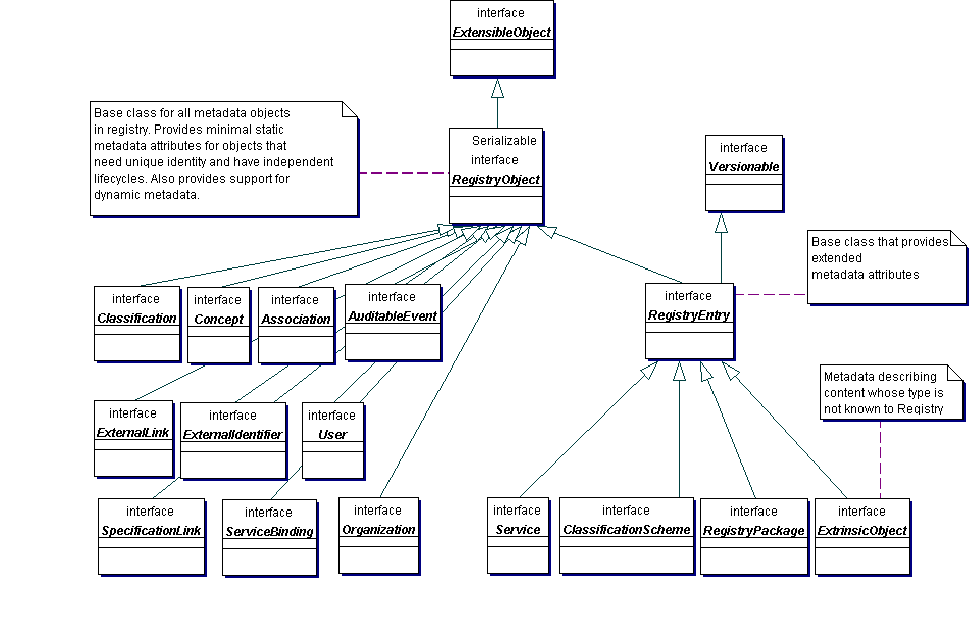
1053 repository items.

Sun Microsystems Page 38

JavaTM API for XML Registries April 10, 2002

1057 **4.2.1 RegistryEntry Interface**

1056



1054

1055 **Figure 12: Information Model Inheritance View**

1058 A few interfaces in the model represent high-level (coarse-grained) objects in the 1059 registry that require additional metadata such as version information and

1060 indication of the stability or volatility of the information.

1061 The RegistryEntry interface is a base interface for the interfaces in the model that

1062 require additional metadata beyond what is provided by the relatively lighter-1063 weight and more fine-grained RegistryObject interface.

1064 **4.2.2 ExtrinsicObject Interface**

1065 ExtrinsicObject instances provide metadata for a repository item (e.g. a WSDL 1066 document or an XML schema document) about which the registry has no prior 1067 knowledge. The ExtrinsicObject interface provides access to a repository item in 1068 the JAXR API.

1069 An ExtrinsicObject instance is required for each repository item.

Sun Microsystems Page 39

JavaTM API for XML Registries April 10, 2002

1070 **4.3 Internationalization (I18N) Support**

1071 Some information model classes have String attributes that are I18N capable and 1072 may be localized into multiple native languages. Examples include the name and 1073 description attributes of the RegistryObject interface as defined by the

1074 set/getName and set/getDescription methods of the RegistryObject interface.

1075 The information model defines the InternationalString and the LocalizedString

1076 interfaces to support I18N capable attributes within the information model 1077 interfaces. These classes are defined below.

1078 **4.3.1 Interface InternationalString**

1079 This interface is used as a replacement for the String type whenever a String 1080 attribute needs to be I18N capable. An instance of the InternationalString

1081 interface composes a Collection of LocalizedString instances within it. Each

1082 LocalizedString instance provides a String value that is specific to a particular

1083 Locale and character set. The InternationalString interface provides set/get

1084 methods for adding or getting locale and character set specific String values for

1085 the InternationalString instance. Each LocalizedString within an

1086 InternationalString must have a unique Locale and character set name

1087 combination within that InternationalString.

1088 **4.3.2 Interface LocalizedString**

1089 This interface is used as a simple wrapper interface that associates a String with

1090 its Locale and character set. The interface is needed in the InternationalString

1091 interface where a Collection of LocalizedString instances are kept. Each

1092 LocalizedString instance has a Locale, a character set name and a String value.

1093 **4.4 Registry Audit Trail**

1094 This section describes the information model elements that support the audit trail

1095 capability of the registry.

1096 The getAuditTrail method of a RegistryObject returns an ordered Collection

1097 of AuditableEvents. These AuditableEvents constitute the audit trail for the

1098 RegistryObject. AuditableEvents include a timestamp for the event. Each

1099 AuditableEvent has a reference to a User instance that identifies the specific user

1100 that performed the action that resulted in an AuditableEvent. Each User is

1101 affiliated with an Organization.

1102

Sun Microsystems Page 40

JavaTM API for XML Registries April 10, 2002

1102 **5 Classification of Registry Objects**

1103 This chapter describes in more detail, how the information model supports the

1104 classification of RegistryObjects. The ability to classify RegistryObjects is one of 1105 the most significant features provided by a registry. This is because classification

1106 facilitates rapid discovery of RegistryObjects within the registry.

1107 **5.1 Interface Classification**

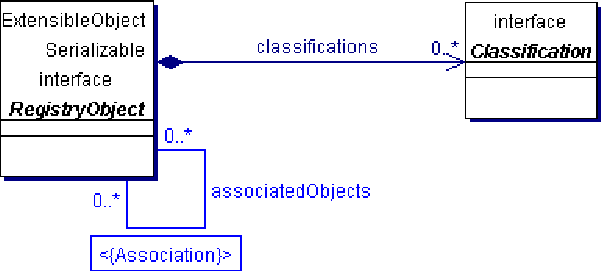
1108 The Classification interface is used to classify RegistryObject instances. A 1109 RegistryObject may be classified along multiple dimensions by adding zero or 1110 more Classification instances to the RegistryObject. For example, an

1111 Organization may be classified by its industry, by the products it sells, by its

1112 geographical location and any other criteria. In this example, the RegistryObject 1113 would have at least three Classification instances added to it (industry, product 1114 and geography).

1115 The RegistryObject interface provides several addClassification methods to

1116 allow a client to add Classification instances to a RegistryObject.



1117

1118 **Figure 13: Classification of Registry Objects**

1119 Figure 13 shows how a RegistryObject may have zero or more Classification

1120 instances defined to classify it along multiple dimensions.

1121 **5.2 Interface ClassificationScheme**

1122 The ClassificationScheme interface is used to represent taxonomies that may be 1123 used to provide taxonomy values that can be used to classify or categorize

1124 RegistryObject instances.

1125 [Note] The term taxonomy and ClassificationScheme are

Sun Microsystems Page 41

JavaTM API for XML Registries April 10, 2002

1126 synonymous in this specification. Taxonomies

1127 are represented using a ClassificationScheme

1128 instance in the JAXR information model.

1129 A Classification instance uses a ClassificationScheme instance to identify the 1130 taxonomy used to classify its RegistryObject. The ClassificationScheme instance

1131 provides the Classification with a taxonomy system that is used by the

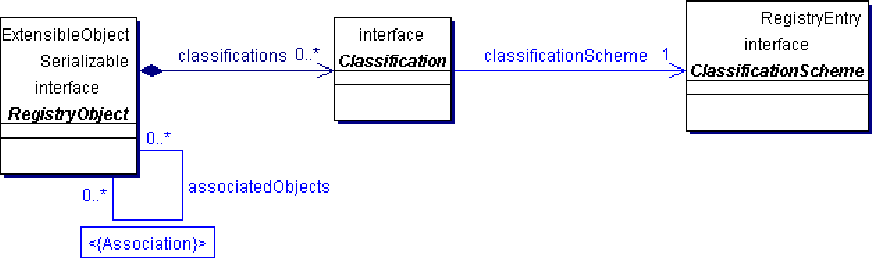
1132 Classification. For example, a Geography ClassificationScheme can provide a

1133 taxonomy system that defines a geography structure with continents, countries

1134 within continents, states (or provinces or other internal subdivisions) within

1135 countries and cities and towns within states.

1136



1137

1138 **Figure 14: Role of ClassificationSchemes in Classification**

1139 Figure 14 shows how a Classification is associated with exactly one

1140 ClassificationScheme instance to identify the taxonomy used to classify a

1141 RegistryObject.

1142 **5.3 Taxonomy Structure and Elements**

1143 A taxonomy must define its structure in terms of its constituent taxonomy 1144 elements and their relationship to each other. For example, in a *Geography* 1145 taxonomy the country elements are contained within continent elements as

1146 illustrated in Figure 15.

1147 A Classification instance needs some way to identify a specific taxonomy 1148 element within a taxonomy, in order to classify a RegistryObject. While a 1149 Classification uses a ClassificationScheme to identify a taxonomy for the

1150 classification, it needs additional information to identify a specific taxonomy

1151 element within that taxonomy.

Sun Microsystems Page 42

JavaTM API for XML Registries April 10, 2002 1152 **5.3.1 Internal Vs. External Taxonomies**

1153 A taxonomy may be represented within a JAXR provider in one of the following 1154 ways:

1155 a. The taxonomy elements and their structural relationship with each other

1156 are available within the JAXR provider. This case is referred to as *Internal*

1157 *Taxonomy* since the structure of the taxonomy is available internally to the

1158 JAXR provider.

1159 b. The taxonomy elements and their structural relationship with each other

1160 are represented somewhere external to the JAXR provider and are not

1161 available to it. This case is referred to as *External Taxonomy* since the

1162 structure of the taxonomy is not available to the JAXR provider.

1163 Internal taxonomies provide more functionality or value to the client in the form of 1164 the ability to browse the taxonomy structure and to validate that references to a 1165 taxonomy element in a Classification are meaningful and correct. The downside 1166 of internal taxonomies is that someone needs to submit that taxonomy to the

1167 registry and to be its maintainer.

1168 In contrast, the upside of external taxonomies is that they are more resilient to 1169 changes in the taxonomy. Once a ClassificationScheme is submitted, the client 1170 can use it immediately without having to import the complete structure of that 1171 taxonomy and to maintain it as the taxonomy structure evolves. The downside of 1172 external taxonomies is that they do not support the ability to browse the 1173 taxonomy or to validate that references to a taxonomy element in a Classification 1174 are meaningful and correct.

1175 **5.3.2 Internal Vs. External Classifications**

1176 The Classification interface allows the classification of RegistryObjects using a 1177 ClassificationScheme whether the ClassificationScheme represents an internal 1178 taxonomy or an external taxonomy. A Classification instance that uses a Concept

1179 within an internal ClassificationScheme is referred to as an internal Classification. 1180 A Classification instance that uses a value within an external

1181 ClassificationScheme, is referred to as an external Classification.

1182 **5.4 Interface Concept**

1183 The Concept interface is used to represent taxonomy elements and their

1184 structural relationship with each other in order to describe an internal taxonomy.

1185 Concept instances are used to define tree structures where the root of the tree is 1186 a ClassificationScheme instance and each node in the tree is a Concept

1187 instance.

Sun Microsystems Page 43

JavaTM API for XML Registries April 10, 2002

1188 Two Concepts may be defined as equivalent, as described in Appendix B. This is 1189 useful in use cases where we need to create a mapping between two different 1190 information model elements. For example, Concept equivalence is used in

1191 mapping the attributes of the PostalAddress interface in the information model to 1192 a Concept in an internal postal address ClassificationScheme. This is defined in

1193 detail in D.6.1.3.

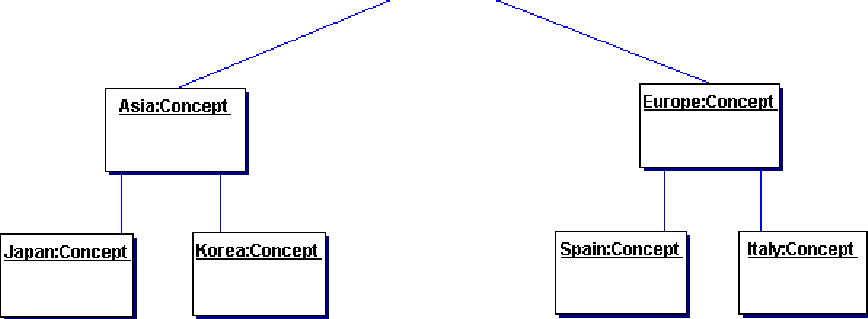
1194 Figure 15 shows how Concept instances are used to represent taxonomy

1195 elements and their structural relationship with each other in order to describe an

1196 internal taxonomy.



1197



1198 **Figure 15: Role of Concepts in Representing Taxonomy Structure**

1199 **5.5 Internal Classification**

1200 A Classification instance that is used to classify a RegistryObject using an

1201 internal taxonomy is referred to as an internal Classification. A client may call the 1202 setConcept method on a Classification and define a reference to a Concept

1203 instance from the Classification instance in order for that Classification to use an

1204 internal taxonomy. It is not necessary for the client to call

1205 setClassificationScheme for internal Classifications, because the classifying

1206 Concept already knows its root ClassificationScheme. For an internal

1207 classification, Classification.getName() must return the same value as

1208 Classification.getConcept().getName().

Sun Microsystems Page 44

JavaTM API for XML Registries April 10, 2002

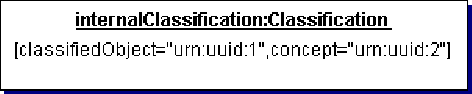
1209 **5.5.1 An Example of Internal Classification**

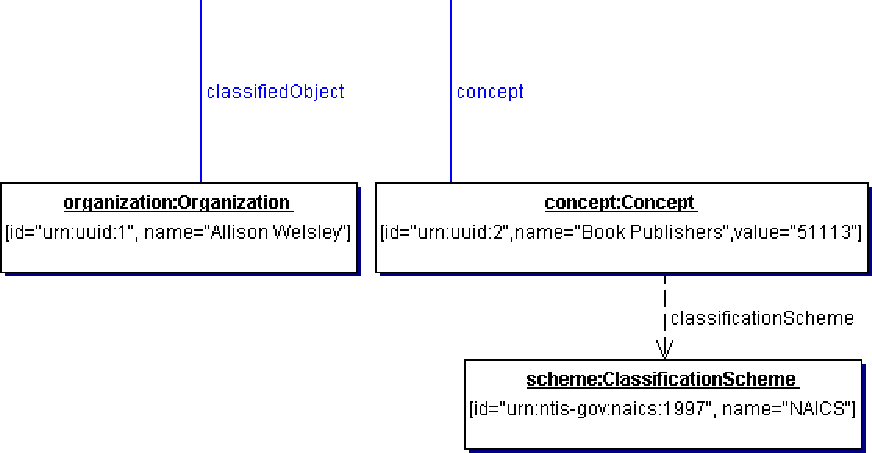
1210 Figure 16 shows an example of internal classification using a Concept to

1211 represent a taxonomy element. The example classifies an Organization instance

1212 as a Book Publisher using the NAICS standard taxonomy available as an internal 1213 taxonomy.

1214 To save space, Figure 16 does not show all the Concepts between the “Book 1215 Publishers” node and the NAICS ClassificationScheme. Had they been there, 1216 they would have been linked together by the parent attribute of each Concept.





1217

1218 **Figure 16: An Example of Internal Classification**

1219 **5.6 External Classification**

1220 A Classification instance that is used to classify a RegistryObject using an

1221 external taxonomy is referred to as an external Classification. A client may call 1222 the setValue method on a Classification and define a unique value that logically 1223 represents a taxonomy element within the taxonomy whose structure is defined 1224 externally. A client may call the setClassificationScheme method for external 1225 Classifications to define the ClassificationScheme that represents the external 1226 taxonomy.

Sun Microsystems Page 45

JavaTM API for XML Registries April 10, 2002

1227 **5.6.1 An Example of External Classification**



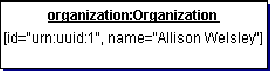
1235

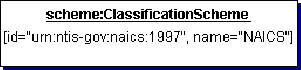
1228 Figure 17 shows an example of external classification. The example uses the 1229 same scenario, where a Classification classifies an Organization instance as a 1230 Book Publisher using the NAICS standard taxonomy. However, this time the

1231 structure of the NAICS taxonomy is not available internally to the registry, and 1232 consequently there is no Concept instance. Instead, the name and value

1233 attributes of the Classification are used to identify the Book Publishers taxonomy 1234 element. Note that name is optional but value is required.







1236 **Figure 17: An Example of External Classification** 1237

1238 **5.7 An Example of Multiple Classifications**

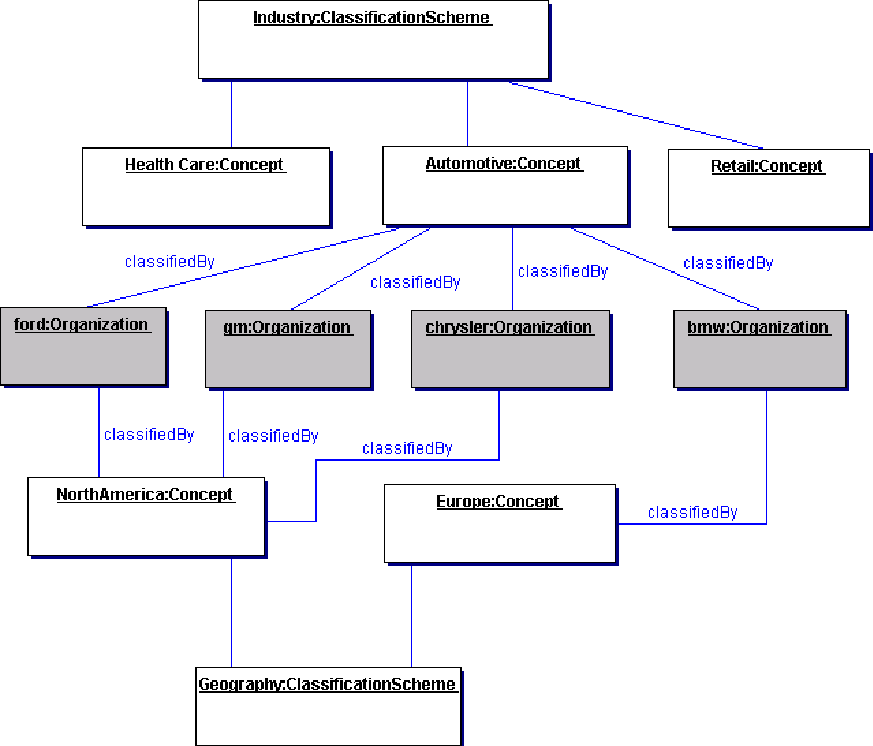
1239 The next example shows how a RegistryObject may be classified by multiple 1240 classification schemes. In this example, two internal ClassificationSchemes

1241 named Industry and Geography are used to classify several Organization 1242 RegistryObjects by their industry and geography.

1243 In Figure 18, in order to save space and improve readability, the Classification 1244 instances are not explicitly shown but are implied as associations between the 1245 RegistryObjects (shaded leaf node) and the associated Concepts.

Sun Microsystems Page 46

JavaTM API for XML Registries April 10, 2002



1246

1247 **Figure 18: An Example of Multiple Classifications**

1248 [Note] It is important to point out that the dark

1249 nodes are not part of the Concept tree. The

1250 leaf nodes of the Concept tree are *Health Care,*

1251 *Automotive, Retail, NorthAmerica and Europe*.

1252 The dark nodes are associated with the Concept

1253 tree via a Classification instance that is not

1254 shown in the figure.

1255 **5.8 Context-sensitive Classification**

1256 [Note] The contents of this section are for

1257 illustrative purposes only.

Sun Microsystems Page 47

JavaTM API for XML Registries April 10, 2002

1258 Consider the case depicted in Figure 19, where a Collaboration Protocol Profile 1259 for ACME Inc. is classified by the *Japan* Concept under the Geography

1260 classification scheme. In the absence of the context for this classification, its

1261 meaning is ambiguous. Does it mean that ACME is located in Japan, or does it 1262 mean that ACME ships products to Japan, or does it have some other meaning? 1263 To address this ambiguity, a Classification may optionally be classified by 1264 another Concept (in this example named isLocatedIn) that provides the missing

1265 context for the Classification. Another Collaboration Protocol Profile for 1266 MyParcelService may be classified by the same *Japan* Concept, where this 1267 Classification is associated with a different Concept (in this example named 1268 shipsTo) to indicate a different context from the one used by ACME Inc.

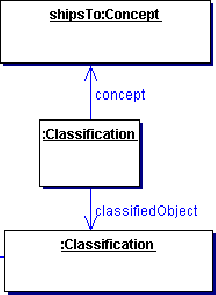
1270 **Figure 19: Context Sensitive Classification**

1269







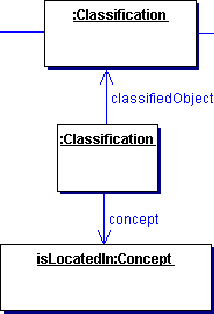












1271 Thus, in order to support the possibility of Classification within multiple contexts,

1272 a Classification may itself be classified by any number of Classifications that bind 1273 the first Classification to Concepts that provide the missing contexts.

1274 In summary, the generalized support for classification schemes in the information

1275 model allows a submitting organization to:

Sun Microsystems Page 48

JavaTM API for XML Registries April 10, 2002

1276 1. Classify a RegistryObject by submitting an internal or external

1277 Classification.

1278 2. Classify a RegistryObject along multiple facets by submitting multiple

1279 classifications.

1280 3. Qualify a classification submitted for a RegistryObject by the contexts in

1281 which it is being classified.

1282

1283

Sun Microsystems Page 49

JavaTM API for XML Registries April 10, 2002

1283 **6 Association of Registry Objects**

1284 A RegistryObject instance may be *associated* with zero or more RegistryObject 1285 instances. The information model defines an Association interface, an instance of 1286 which may be used to associate any two RegistryObject instances.

1287 **6.1 Example of an Association**

1288 One example of such an association is between two ClassificationScheme

1289 instances, where one ClassificationScheme supersedes the other

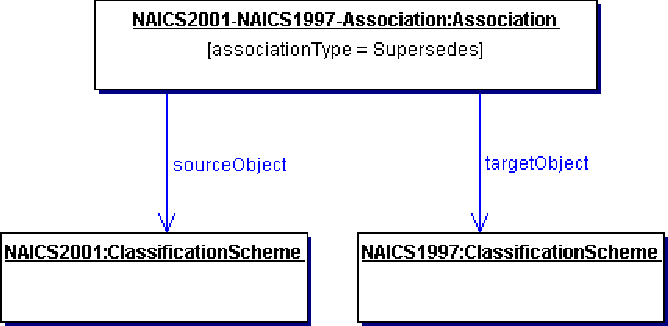
1290 ClassificationScheme, as shown in Figure 20. This may be the case when a new

1291 version of a ClassificationScheme is submitted.

1292 In Figure 20, we see how an Association is defined between a new version of the

1293 NAICS ClassificationScheme and an older version of the NAICS

1294 ClassificationScheme.



1295

1296 **Figure 20: Example of RegistryObject Association**

1297 **6.2 Source and Target Objects**

1298 An Association instance represents an association between a *source*

1299 RegistryObject and a *target* RegistryObject. These are referred to as the

1300 *sourceObject* and *targetObject* for the Association instance. It is important which

1301 object is the sourceObject and which is the targetObject, because it determines 1302 the directional semantics of an Association.

Sun Microsystems Page 50

JavaTM API for XML Registries April 10, 2002

1303 In the example in Figure 20, it is important to make the newer version of NAICS 1304 ClassificationScheme be the sourceObject and the older version of NAICS be the 1305 targetObject because the associationType implies that the sourceObject

1306 supersedes the targetObject (and not the other way around).

1307 **6.3 Association Types**

1308 Each Association must have an associationType attribute that identifies the type

1309 of that association. The associationType attribute is a reference to an

1310 enumeration Concept as defined by the extensible ClassificationScheme defined

1311 in A.4. Our example uses the predefined associationType Concept named

1312 *Supersedes*.

1313 **6.4 Intramural Associations**

1314 A common use case for the Association interface is when a User “u” creates an 1315 Association “a” between two RegistryObjects “o1” and “o2”, where association “a” 1316 and RegistryObjects “o1” and “o2” are objects that were created by the same 1317 User “u”. This is the simplest use case, where the association is between two 1318 objects that are owned by same User that is defining the Association. Such 1319 associations are referred to as *intramural associations*.

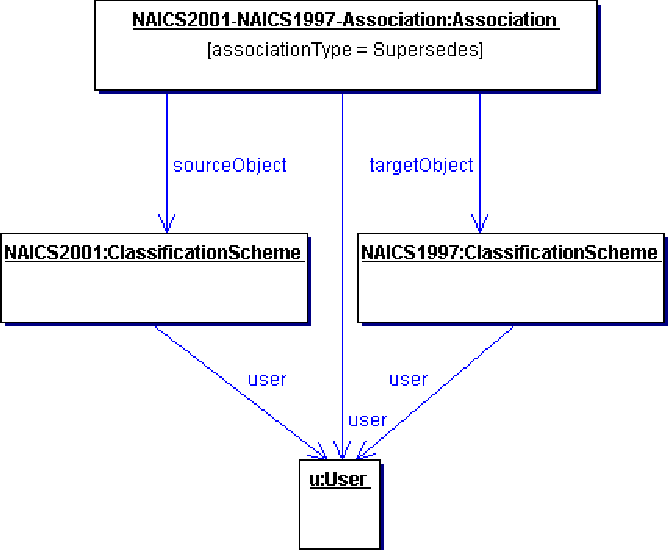
1320 Figure 21 below extends the previous example in Figure 20 for the intramural

1321 association case.

1322

Sun Microsystems Page 51

JavaTM API for XML Registries April 10, 2002



1323

1324 **Figure 21: Example of Intramural Association**

1325 **6.5 Extramural Association**

1326 The information model also allows a more sophisticated use case, where a User 1327 “u1” creates an Association “a” between two RegistryObjects “o1” and “o2”, 1328 where association “a” is owned by User “u1”, but RegistryObjects “o1” and “o2” 1329 are owned by User “u2” and User “u3” respectively.

1330 In this use case the Association is being defined where either or both objects that 1331 are being associated are owned by a User different from the User defining the 1332 Association. Such associations are referred to as *extramural associations*. The 1333 Association interface provides a convenience method called isExtramural that

1334 returns true if the Association instance is an extramural Association.

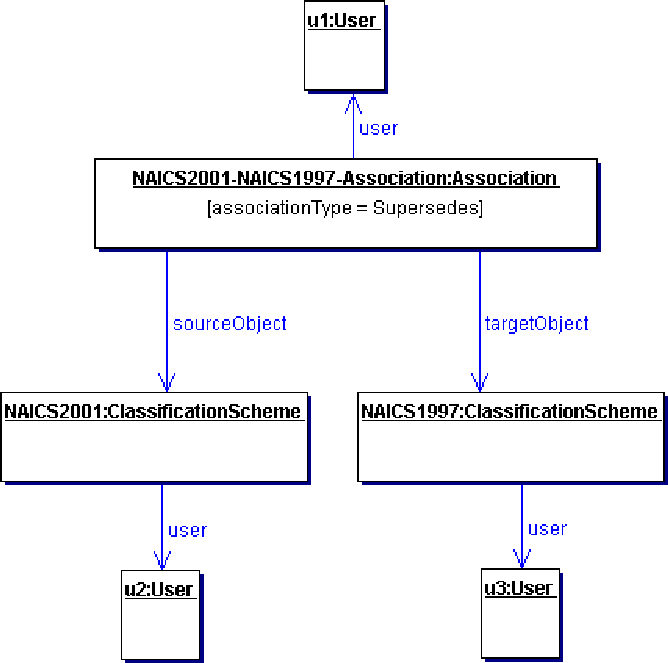
1335 Figure 22 extends the example in Figure 20 for the extramural association case.

1336 Note that it is possible for an extramural association to have two distinct Users 1337 rather than three distinct Users as shown in Figure 22. In such a case, one of the 1338 two users owns two of the three objects involved (Association, sourceObject, and

1339 targetObject).

Sun Microsystems Page 52

JavaTM API for XML Registries April 10, 2002



1340

1341 **Figure 22: Example of Extramural Association**

1342 **6.6 Confirmation of an Association**

1343 An association may need to be confirmed by the parties whose objects are

1344 involved in that Association as the sourceObject or targetObject. This section

1345 describes the semantics of confirmation of an association by the parties involved.

1346 **6.6.1 Confirmation of Intramural Associations**

1347 Intramural associations may be viewed as declarations of truth and do not

1348 require any explicit steps to confirm that Association as being true. In other 1349 words, intramural associations are implicitly considered confirmed.

Sun Microsystems Page 53

JavaTM API for XML Registries April 10, 2002 1350 **6.6.2 Confirmation of Extramural Associations**

1351 Extramural associations may be viewed as a unilateral assertion that may not be

1352 viewed as truth until it has been confirmed by the other (extramural) parties

1353 (Users “u2” and “u3” in example in section 6.5). The confirmAssociation

1354 method of the BusinessLifeCycleManager interface may be called by the

1355 extramural parties that own the sourceObject or targetObject to confirm an

1356 Association.

1357 **6.6.3 Undoing Confirmation of Extramural Associations**

1358 The unConfirmAssociation method on the BusinessLifeCycleManager 1359 interface may be called by the extramural parties that own the sourceObject or 1360 targetObject to undo a previous confirm on an Association.

1361 **6.7 Visibility of Unconfirmed Associations**

1362 Extramural associations require each extramural party to confirm the assertion 1363 being made by the extramural Association before the Association is visible to 3rd

1364 parties that are not involved in the Association. This ensures that unconfirmed

1365 Associations are not visible to 3rd party registry clients.

1366 In order for a caller to find the Associations that it needs to confirm, it can use the 1367 findCallerAssociations method of the BusinessQueryManager interface.

1368 **6.8 Possible Confirmation States**

1369 Assume the most general case where there are three distinct User instances for 1370 an extramural Association. This case is illustrated in Figure 22. The extramural 1371 Association needs to be confirmed by both the other (extramural) parties (Users 1372 “u2” and “u3” in example) in order to be fully confirmed. The methods

1373 isConfirmedBySourceOwner and isConfirmedByTargetOwner in the 1374 Association interface provide access to the confirmation state for both the 1375 sourceObject and targetObject. A third convenience method, called

1376 isConfirmed provides a way to determine whether the Association is fully

1377 confirmed or not. So there are the following four possibilities related to the 1378 confirmation state of an extramural Association:

1379

|  |  |  |
| --- | --- | --- |
| **Confirmed By Owner of Source Object** | **Confirmed By Owner of Target Object** | **Comments** |
| No | No | Unconfirmed |
| No | Yes | Unconfirmed (confirmed |

Sun Microsystems Page 54

JavaTM API for XML Registries April 10, 2002

by target owner only).

Yes

No

Unconfirmed (confirmed

by source owner only).

Yes

Yes

Confirmed.

1380

1381

Sun Microsystems Page 55

JavaTM API for XML Registries April 10, 2002

1381 **7 Connection Management**

1382 This chapter specifies how a JAXR provider manages JAXR Connections. It 1383 provides details on the ConnectionFactory interface and the Connection

1384 interface.

1385 **7.1 Looking Up a ConnectionFactory**

1386 A JAXR ConnectionFactory object is configured in a provider-specific way to 1387 create connections with registry providers.

1388 **7.1.1 Looking Up a ConnectionFactory Using the JNDI API**

1389 The preferred way for a client to look up a JAXR ConnectionFactory is within the 1390 Java Naming and Directory InterfaceTM (JNDI) API.

1391 A ConnectionFactory object is registered with a naming service in a provider 1392 specific way, such as one based on the JNDI API. This registration associates 1393 the ConnectionFactory object with a logical name. When an application 1394 wants to establish a connection with the provider associated with that

1395 ConnectionFactory object, it does a lookup, providing the logical name. The 1396 application can then use the ConnectionFactory object that is returned to 1397 create a connection to the messaging provider.

1398 **7.1.2 Looking Up a ConnectionFactory Without Using the JNDI API**

1399 The JAXR API provides an alternative way to look up a JAXR ConnectionFactory 1400 that does not require the use of the JNDI API. This is done using the

1401 newInstance static method on the abstract class ConnectionFactory provided in 1402 the JAXR API. The newInstance method returns a JAXR ConnectionFactory. The 1403 client may indicate which factory class should be instantiated by the newInstance 1404 method by defining the system property

1405 javax.xml.registry.ConnectionFactoryClass.

1406 If this property is not set, the JAXR provider must return a default 1407 ConnectionFactory instance.

1408 **7.2 Setting Connection Properties on ConnectionFactory**

1409 Once a ConnectionFactory is available to the client, the client may configure the 1410 ConnectionFactory with a Properties object by calling the setProperties method

1411 on ConnectionFactory.

Sun Microsystems Page 56

JavaTM API for XML Registries April 10, 2002

1412 The properties specified may be either *standard properties* or *provider-specific* 1413 *properties*.

1414 1. Standard properties are defined by the JAXR API.

1415 2. Provider-specific properties are defined by a specific JAXR provider.

1416 Connection configuration properties must be qualified by a package name. 1417 Standard connection configuration properties use the package

1418 javax.xml.registry, while provider-specific properties use the top-level 1419 package name for the provider (e.g. com.sun.xml.registry.ebxml).

1420 **7.2.1.1 Standard Connection Properties**

1421 The following table describes those connection properties that are standardized

1422 by this specification. JAXR providers may define additional properties specific to

1423 that provider.

1424

1425

**Property**

Data

**type**

**Description**

javax.xml.registry. queryManagerURL

String

URL to the query manager service within the target registry provider.

javax.xml.registry.

lifeCycleManagerURL

String

URL to the life cycle manager service within the target registry provider. If un­specified, must default to value of the queryManagerURL described above.

javax.xml.registry.

semanticEquivalences

String

Allows specification of semantic equivalences as described in Appendix B.

javax.xml.registry. security.

authenticationMethod

String

Provides a hint to the JAXR provider on the authentication method to be used when authenticating with the registry provider.

javax.xml.registry.uddi .maxRows

Integer

Specifies the maximum number of rows to be returned for find operations. This property is specific for UDDI providers.

javax.xml.registry. postalAddressScheme

String

Specifies the id of a ClassificationScheme that is used as the default postal address scheme for this connection. See D.6.1.3 for details.

Sun Microsystems Page 57

JavaTM API for XML Registries April 10, 2002

1431

1432

1426

1427 **7.3 Creating a JAXR Connection**

1428 To create a Connection to a registry provider, a client uses the

1429 createConnection method of a JAXR provider’s ConnectionFactory.

1430

|  |
| --- |
| public Connection **createConnection**()  throws JAXRException |

1433 The createConnection method must check that the

1434 javax.xml.registry.queryManagerURL described above is defined. If it is

1435 not defined, then the method must throw an InvalidRequestException. The

1436 createConnection method may also check if the URL specified is a valid URL.

1437 **7.4 Synchronous Connections**

1438 The JAXR client uses the setSynchronous method on a Connection to

1439 dynamically alter whether it receives responses and exceptions from the JAXR

1440 provider synchronously or not. The JAXR provider must use this communication

1441 preference when processing requests on behalf of that client.

1442 If the communication preference is synchronous, the JAXR provider must 1443 process each request method call completely in a synchronous manner before it 1444 returns a non-null JAXRResponse (or a sub-interface) instance that contains the 1445 response to the request. The client thread must block until the JAXR provider has 1446 synchronously processed the request. The processing usually involves a round-

1447 trip interaction with a registry provider.

1448 **7.5 Asynchronous Connections**

1449 The JAXR client may indicate an asynchronous communication mode by calling 1450 the setSynchronous method on a Connection with a false parameter.

1451 If the communication preference is asynchronous, each request method call 1452 returns a non-null JAXRResponse (or a sub-interface) immediately. The JAXR 1453 provider may spawn a separate thread to process the client request

1454 asynchronously.

1455 [Note] In this version of the specification,

1456 asynchronous communication mode is not required

1457 within a J2EE container environment. The reason

1458 is that asynchronous mode support typically

1459 requires threads, sockets and so on, which are

1460 not allowed within a J2EE component.

Sun Microsystems Page 58

JavaTM API for XML Registries April 10, 2002 1461 **7.5.1 JAXRResponse and Futures Design Pattern**

1462 The JAXRResponse returned immediately by the JAXR provider may not be

1463 immediately available. It uses a “futures” design pattern [Futures1, Futures2].

1464 For an asynchronous request, a JAXR Provider will immediately give the client 1465 back a JAXRResponse instance even though its value is not available. Internally, 1466 the JAXR Provider will allocate a globally unique request ID. It is suggested that 1467 this ID be a DCE 128 bit UUID. This request ID is available to the client via the 1468 getRequestId method on JAXRResponse. The JAXR Provider must maintain 1469 the mapping between request IDs and JAXRResponse instances so that when a

1470 reply arrives from the registry provider at some time in the future, it can find the 1471 corresponding JAXRResponse instance and deliver the reply to the instance. 1472 This causes the value in the JAXRResponse instance to become available.

1473 If a client attempts to read a value from a JAXRResponse that is not yet

1474 available, the JAXR provider must cause the caller to be blocked. Alternatively, a

1475 caller may examine a JAXRReponse for the availability of its value before

1476 attempting to read its value (and potentially blocking). The availability of the

1477 value may be polled by getStatus method, which must return

1478 STATUS\_UNAVAILABLE when invoked on a JAXRResponse with a undefined

1479 value. In addition an isAvailable method on JAXRResponse is also provided

1480 as a convenience. The isAvailable method on JAXRResponse returns true or

1481 false depending upon whether the value is available or not.

1482 Having a reference to a JAXRResponse does not block the client thread. The 1483 client thread is blocked only when it tries to access the reply contained in a 1484 JAXRResponse and the reply is not available.

1485 **7.6 Security Credentials Specification**

1486 The JAXR client uses the setCredentials method to dynamically alter its

1487 security credentials. These credentials provide details on the security-related 1488 identity associated with the client. An example of a credential is a username and 1489 password combination. The JAXR provider must use the credentials defined in 1490 the Connection instance at any given time when processing client requests. This

1491 may require having to re-authenticate with the registry provider in response to the 1492 setCredentials call when appropriate. The setCredentials method is 1493 described in more detail in section 10.5.1 within the context of JAXR security

1494 features.

Sun Microsystems Page 59

JavaTM API for XML Registries April 10, 2002 1495 **7.7 Federated Connections**

1496 The JAXR API defines an interface named FederatedConnection. The

1497 FederatedConnection interface is a specialized sub-interface of the Connection 1498 interface. The FederatedConnection interface defines a single logical connection 1499 to multiple registry providers. A FederatedConnection is used in performing

1500 distributed or federated queries against target registry providers while treating

1501 them as a single logical registry provider. Federated queries are described in

1502 Section 9.7. The federated connection capability is optional in this version of the

1503 specification.

1504 **7.7.1 Creating a FederatedConnection**

1505 To create a federated connection to multiple registry providers, a client uses the

1506 createFederatedConnection method of a JAXR provider’s

1507 ConnectionFactory.

1508

public FederatedConnection **createFederatedConnection**(Collection connections) throws JAXRException

1511 **7.7.1.1 Connection Configuration**

1509

1510

1512 The client must specify a Collection of Connection instances to the

1513 createFederatedConnection method. These connections may be primitive

1514 connections or federated connections.

1515 **7.8 Concurrent Connections**

1516 A single JAXR client may concurrently maintain multiple connections. Each non-1517 federated connection uses a single JAXR provider to access a single registry

1518 provider.

1519 Collectively, these connections may concurrently access multiple registry

1520 providers.

1521 Connection implementations must be thread-safe implementations. 1522 **7.9 Using a Connection to Access the Registry**

1523 Once a JAXR client has created a Connection using a ConnectionFactory, it can 1524 then use the Connection to access various capability specific interfaces. For 1525 example, it may use the Connection to:

1526 a. Access the lifecycle management functionality of the JAXR provider to

1527 create, update and delete objects in the target registry provider.

Sun Microsystems Page 60

JavaTM API for XML Registries April 10, 2002

1528 b. Access the query management functionality of the JAXR provider to find

1529 and retrieve objects from the target registry provider.

1530 The client must first get access to a RegistryService interface by calling the 1531 getRegistryService method on the Connection. The client may then call 1532 appropriate methods on the RegistryService interface to get references to

1533 various capability specific interfaces. For example, it may call the

1534 getBusinessLifeCycleManager method to get the

1535 BusinessLifeCycleManager interface for that Connection.

1536 **7.10 Closing a Connection**

1537 A JAXR provider typically allocates significant resources outside the JVM on 1538 behalf of a Connection. These resources include a network connection between 1539 the JAXR provider and the target registry provider shown at the bottom of Figure 1540 4. The network connection between a JAXR provider and a target registry 1541 provider is represented in Figure 4 by the block arrows marked ebXML/SOAP 1542 etc. The technical details of such network connections are registry provider-

1543 specific and therefore outside the scope of this specification.

1544 In order to conserve system resources, clients should close Connections when 1545 they are no longer needed. A client closes a Connection by calling the close

1546 method on it.

1547 **7.11 Connection Setup Sequence**

1548 Figure 23 illustrates the sequence of events during a typical JAXR Connection

1549 establishment.

Sun Microsystems Page 61

JavaTM API for XML Registries April 10, 2002

L

Lri

connection Connection

T ConnectionFactoty:=lookup(java.lang.Pingjava.lang.Objegt

reuClient

clientiknolication

ndiContext

Context

selPropertiespropertiesyvoid

factory ConnectionFactory

Connection:=createConnectionCEConnection

setCredentials(Set):void

setSynchronous(boolean):void

RegistrySetvice:=gelRegisttyService0:1TegistlySetvice

y



**Figure 23: Connection Setup Sequence**

1550

1551

Sun Microsystems Page 62

1564

1577

1576

1575

1574

1573

1572

1571

1570

1569

1568

1567

1566

1565

1578

1563

1562

1561

1560

1559

1558

1554

1555

1556

1557

1553

Sun Microsystems Page 63

1552 **7.11.1 Connection Creation Code Sample**

JavaTM API for XML Registries April 10, 2002



//Define connection configuration properties

BusinessLifeCycleManager lcm = rs.getBusinessLifeCycleManager();

//Now get one or more capability specific interfaces

RegistryService rs = connection.getRegistryService();

connection.setSynchronous(false);

connection.setCredentials(credentials);

....

Set credentials = new HashSet();

Connection connection = factory.createConnection();

factory.setProperties(props);

//Create the connection passing it the configuration properties

props.put("javax.xml.registry.lifeCycleManagerURL", "<http://java.sun.com/uddi/publish>");

props.put("javax.xml.registry.queryManagerURL", "<http://java.sun.com/uddi/inquiry>");

Properties props = new Properties();

ConnectionFactory factory = ConnectionFactory.newInstance();

//method.

//Create ConnectionFactory using class specified in System property and static newInstance

"com.sun.xml.registry.uddi.ConnectionFactoryImpl");

System.setProperty("javax.xml.registry.ConnectionFactoryClass",

//Add system property to define which provider specific ConnectionFactory to use

import javax.xml.registry.\*;

1583

1582

1581

1580

1579

JavaTM API for XML Registries April 10, 2002

1583 **8 Life Cycle Management**

1584 This chapter specifies those aspects of the JAXR API that deal with managing 1585 the life cycle of registry objects. Examples of life cycle management operations

1586 include the ability to create, update and deleted registry objects.

1587 Life cycle management interfaces are designed for use by the *Submitting* 1588 *Organization* or the *publisher* of registry metadata and content.

1589 [Note] Some life cycle management requests may be

1590 privileged operations and require

1591 authentication and authorization. Security

1592 aspects of life cycle management is discussed

1593 in Chapter 10.

1594 The JAXR API provides the following interfaces for managing life cycle of registry

1595 objects:

1596 o Interface LifeCycleManager provides complete support for all life cycle

1597 management needs using a generic API.

1598 o Interface BusinessLifeCycleManager extends the LifeCycleManager

1599 interface and provides the most common life cycle management

1600 capabilities for the key business objects in the information model in an

1601 explicit API.

1602 **8.1 Unique Key Assignment**

1603 As specified in the information model, every RegistryObject in the registry has a 1604 unique key. This key is usually generated by the registry provider. Some registry 1605 providers, such as ebXML Registry providers, optionally allow the submitter to

1606 specify this unique key.

1607 The JAXR API allows a client to specify a key for a RegistryObject when

1608 submitting it to the registry. The client-supplied key must be used as the key for

1609 the object within the target registry provider, if all of the following conditions are

1610 true:

1611 o The client supplies a key.

1612 o The registry provider supports client supplied keys

1613 o The client-supplied key is in a format acceptable by the specification

1614 governing the target registry. For example ebXML Registry specification

1615 requires client-supplied keys to be UUID based URNs.

Sun Microsystems Page 64

JavaTM API for XML Registries April 10, 2002

1616 A JAXR provider must ignore client-supplied keys if any of the above conditions 1617 are not met. In this case, the registry provider is expected to generate the unique

1618 key.

1619 **8.2 Interface *LifeCycleManager***

1620 This interface provides complete support for all life cycle management needs

1621 using a generic API.

1622 **8.2.1 Requests, Responses and Exception Handling**

1623 The LifeCycleManager interface provides several methods that allow clients to

1624 make lifecycle requests with the target registry. Examples include:

1625 o deleteObjects

1626 o deprecateObjects

1627 o saveObjects

1628 o unDeprecateObjects

1629 Each of these methods returns a BulkResponse object. The BulkResponse 1630 contains the Collection of response objects and may contain a Collection of

1631 RegistryException instances in case the request resulted in errors within the

1632 target registry provider.

1633 If the JAXR provider detects an error in a client request prior to contacting the 1634 target registry provider, then the JAXR provider must throw a JAXRException. 1635 The JAXR provider must not catch runtime exceptions such as

1636 NullPointerException, since they are indicative of a programming error in the 1637 JAXR provider or the JAXR client. Instead, the JAXR provider should catch

1638 errors in a client request before they lead to potential runtime exceptions (e.g.

1639 NullPointerException). In such cases, the JAXR provider must throw a

1640 JAXRException that provides a clear indication as to the error in the client

1641 request and how to fix the error.

1642 However, if the error is detected by the target registry provider, then the

1643 RegistryException is included within the BulkResponse instance’s Collection of 1644 RegistryExceptions.

1645 **8.2.2 Creating Objects Using Factory Methods**

1646 The LifeCycleManager interface has several factory methods that follow the 1647 naming pattern create<interface> where <interface> represents the name

1648 of an interface in the javax.xml.registry.infomodel package.

Sun Microsystems Page 65

JavaTM API for XML Registries April 10, 2002

1659

1660

1649 These factory methods are used by clients to create different types of objects 1650 defined by the information model.

1651 There is also a generic factory method called createObject, which allows 1652 clients to create any type of information model object. This method is useful for 1653 tool vendors who create tools supporting the JAXR API.

1654 **8.2.3 Saving Objects**

1655 An object created using a factory method initially exists only in memory. It is not

1656 saved in a registry provider until it is saved explicitly. The LifeCycleManager 1657 interface provides a saveObjects method for this purpose.

1658

|  |
| --- |
| public BulkResponse **saveObjects** (java.util.Collection objects) throws JAXRException |

1661 The saveObjects method is used to create or update metadata and content.

1662 This method saves a heterogeneous Collection of objects that are instances of

1663 RegistryObject sub-interfaces. Each such object in the collection is stored into 1664 the registry. Implementations must traverse object references from the object

1665 being saved and save them implicitly.

1666 ***8.2.3.1* Interface *BulkResponse***

1667 The BulkResponse interface is returned by many methods in the API where the 1668 response needs to include a Collection of objects. The BulkResponse interface is 1669 described here in the context of save methods. It behaves similarly in other 1670 usage contexts.

1671 Each save method returns a BulkResponse instance. The BulkResponse

1672 instance contains a Collection of keys that are accessed via the

1673 getCollection method. These keys are for those objects that were saved

1674 successfully. The BulkResponse may also contain a Collection of

1675 SaveException instances as described next.

1676 **8.2.3.2 Interface SaveException**

1677 In event of a partial success where only a subset of objects was saved

1678 successfully, the getStatus method of the BulkResponse must return

1679 JAXRResponse.STATUS\_WARNING. In this case, a Collection of

1680 SaveException instances is included in the BulkResponse instance. The

1681 SaveExceptions provide information on each error that prevented some

1682 objects in the save method Collection from being saved successfully.

Sun Microsystems Page 66

JavaTM API for XML Registries April 10, 2002

1704

1705

1683 Note that the SaveExceptions are returned as part of the BulkResponse 1684 instead of being thrown, in order to allow the BulkResponse to be returned 1685 despite the exception.

1686

1687 **8.2.3.3 Implicit Saving of Objects**

1688 A JAXR provider must traverse object references from the object being saved 1689 and save them implicitly. For example, if the client saves an Organization

1690 explicitly, then the JAXR provider must implicitly save any Classifications,

1691 Associations, ExternalIdentifiers, Services, ServiceBindings etc., that are 1692 reachable from the Organization being saved.

1693 **8.2.4 Updating Objects**

1694 An object that is created and subsequently saved to the target registry provider 1695 can be updated by modifying the object in memory. Such updated objects must 1696 be saved using save methods such as saveObjects in order to be updated in the 1697 registry. The LifeCycleManager interface does not provide update methods that 1698 are distinct and separate from save methods.

1699 **8.2.5 Deleting Objects**

1700 An object that is created and subsequently saved to the target registry provider

1701 may be deleted from the target registry using a delete method of 1702 LifeCycleManager such as deleteObjects.

1703

|  |
| --- |
| public BulkResponse **deleteObjects** (java.util.Collection keys) throws JAXRException |

1706 The deleteObjects method is used to delete previously submitted RegistryObject

1707 instances.

1708 This method specifies a uniform Collection of Key instances identifying pre-

1709 existing objects in the registry.

1710 An attempt to remove a RegistryObject while it is still the target of references

1711 may result in an InvalidRequestException that is returned within the

1712 BulkResponse, if the registry provider enforces such deletion constraints.

Sun Microsystems Page 67

JavaTM API for XML Registries April 10, 2002

1713 **8.2.5.1 Interface DeleteException**

1714 This exception is used in the event of failure or partial success during a delete

1715 operation. A Collection of instances of this exception is returned in the 1716 BulkResponse for a delete method invocation if delete exceptions are 1717 encountered.

1718 **8.2.6 Deprecating Objects**

1719 Deprecating an object is an alternative to deleting it. Deprecating an object marks

1720 it as obsolete and likely to be deleted sometime in the future.

1721 An object that is created and subsequently saved to the target registry provider

1722 may be deprecated from the target registry using a deprecate method of

1723 LifeCycleManager such as deprecateObjects.

1724

|  |
| --- |
| public BulkResponse **deprecateObjects** (java.util.Collection keys) throws JAXRException |

1727

1725

1726

1728 The deprecateObjects method is used to deprecate previously submitted 1729 RegistryObject instances.

1730 This method specifies a uniform Collection of Key instances identifying pre-

1731 existing objects in the registry. Deprecating an object marks it as becoming

1732 obsolete. A deprecated object may remain in the registry for some time before it

1733 is deleted.

1734 Once an object is deprecated, the JAXR provider must not allow any new

1735 references (e.g. new Associations, Classifications and ExternalLinks) to that 1736 object to be submitted. If a client makes an API call that results in a new 1737 reference to a deprecated object, the JAXR provider must throw a

1738 java.lang.IllegalStateException within a JAXRException. However, existing 1739 references to a deprecated object continue to function normally.

1740 **8.2.7 Undeprecating Objects**

1741 A deprecated object may be undeprecated using the unDeprecateObjects 1742 method of LifeCycleManager.

1743

1744

1745

|  |
| --- |
| public BulkResponse unD**eprecateObjects**(java.util.Collection keys) throws JAXRException |

1746

Sun Microsystems Page 68

JavaTM API for XML Registries April 10, 2002

1747 The unDeprecateObjects method is used to un-deprecate previously deprecated 1748 RegistryObject instances.

1749 This method specifies a uniform Collection of Key instances identifying pre-

1750 existing deprecated objects in the registry.

1751 Once an object is undeprecated, the JAXR provider must again allow new

1752 references (e.g. new Associations, Classifications and ExternalLinks) to that 1753 object to be submitted.

1754 **8.3 Interface *BusinessLifeCycleManager***

1755 Interface BusinessLifeCycleManager defines a simple business-level API

1756 for life cycle management of some important high-level interfaces in the

1757 information model. This interface provides no new functionality beyond that of

1758 LifeCycleManager. The goal of defining this interface is to provide an API similar 1759 to that of the publisher’s API in UDDI. The intent is to provide a familiar API to 1760 UDDI developers.

1761 The BusinessLifeCycleManager interface provides the ability to explicitly submit,

1762 update or delete instances of the most important high-level interfaces in the

1763 information model. These high-level interfaces include:

1764 o Interface Organization

1765 o Interface Service

1766 o Interface ServiceBinding

1767 o Interface Concept

1768

1769

|  |  |
| --- | --- |
| **Method Summary** | |
| void | **confirmAssociation**(Association assoc) |
| Confirms this Association by the User associated with the caller. |
| BulkResponse | **deleteAssociations**(java.util.Collection associationKeys) |
|  | Deletes the Associations corresponding to the specified Keys. |
| BulkResponse | **deleteClassificationSchemes**(java.util.Collection schemeKeys) |
|  | Deletes the ClassificationSchemes corresponding to the specified Keys. |

Sun Microsystems Page 69

JavaTM API for XML Registries April 10, 2002

1770

|  |  |
| --- | --- |
| BulkResponse | **deleteConcepts**(java.util.Collection conceptKeys) |
|  | Deletes the Concepts corresponding to the specified Keys. |
| BulkResponse | **deleteOrganizations**(java.util.Collection organizationKeys) |
|  | Deletes the organizations corresponding to the specified Keys. |
| BulkResponse | **deleteServiceBindings**(java.util.Collection bindingKeys) |
|  | Deletes the ServiceBindings corresponding to the specified Keys. |
| BulkResponse | **deleteServices**(java.util.Collection serviceKeys) |
|  | Deletes the services corresponding to the specified Keys. |
| BulkResponse | **saveAssociations**(java.util.Collection associations, |
|  | boolean replace)  Saves the specified Association instances. |
| BulkResponse | **saveClassificationSchemes**(java.util.Collection schemes) |
|  | Saves the specified ClassificationScheme instances. |
| BulkResponse | **saveConcepts**(java.util.Collection concepts) |
|  | Saves the specified Concepts. |
| BulkResponse | **saveOrganizations**(java.util.Collection organizations) |
|  | Saves the specified Organizations. |
| BulkResponse | **saveServiceBindings**(java.util.Collection bindings) |
|  | Saves the specified ServiceBindings. |
| BulkResponse | **saveServices**(java.util.Collection services) |
|  | Saves the specified Services. |
| void | **unConfirmAssociation**(Association assoc) |
| Undoes a previous confirmation of this Association by the User associated with the caller. |

Sun Microsystems Page 70

JavaTM API for XML Registries April 10, 2002 1771 **8.3.1 Save Methods**

1772 The BusinessLifeCycleManager interface defines a set of save methods, 1773 one for each key interface (e.g. saveOrganizations). Each save method takes a 1774 Collection as parameter so it can save multiple objects of the type associated 1775 with the save method. That Collection contains objects that are instances of the 1776 type associated with the save method. For example, the saveOrganizations

1777 method accepts a Collection of Organization instances. If the Collection contains 1778 an object whose type does not match the save method, the implementation must 1779 throw an UnexpectedObjectException.

1780 Note that a client must save an object using the appropriate save method when a

1781 setter method on the object is called by the client. Calling the setter method 1782 without saving the object will not save the modified object to the target registry.

1783 The save methods of BusinessQueryManager are a convenience for those who 1784 prefer an explicit API. Calling a save method of BusinessQueryManager should 1785 have the same effect as calling the saveObjects method in LifeCycleManager.

1786 **8.3.2 Delete Methods**

1787 The BusinessLifeCycleManager interface defines a set of delete methods, 1788 one for each key interface (e.g. deleteOrganizations etc.). Each delete method 1789 takes a Collection as parameter so it can delete multiple objects of the type 1790 associated with the delete method. The Collection is homogeneous and

1791 contains keys to objects that are being deleted. For example, the

1792 deleteOrganizations method accepts a Collection of Key instances where 1793 each Key is the Key for an Organization instance. If the Collection contains a key 1794 whose object type does not match the delete method, the implementation must 1795 throw an UnexpectedObjectException.

1796 The delete methods of BusinessQueryManager are a convenience for those who 1797 prefer an explicit API. Calling a delete method of BusinessQueryManager should 1798 have the same effect as calling the deleteObjects method in LifeCycleManager.

1799 **8.4 Life Cycle Management and Federated Connections**

1800 Life cycle management operations are not supported by federated connections 1801 as represented by a FederatedConnection. The getLifeCycleManager and 1802 getBusinessLifeCycleManager methods of RegistryService from a 1803 FederatedConnection must throw UnsupportedCapabilityException.

1804

Sun Microsystems Page 71

JavaTM API for XML Registries April 10, 2002

1804 **9 Query Management**

1805 This section specifies those aspects of the JAXR API that deal with querying the

1806 registry for registry object (metadata) and repository item (content) instances. 1807 Query management interfaces are designed for use by any non-privileged

1808 registry user, typically through a specialized JAXR client, such as a Registry

1809 Browser tool or an intelligent web agent (digital assistant).

1810 [Note] Security aspects of query management are

1811 discussed in Chapter 10.

1812

1813 The API provides a common QueryManager interface as well as two specialized

1814 sub-interfaces:

1815 o Interface BusinessQueryManager

1816 o Interface DeclarativeQueryManager

1817 ***9.1* Interface *QueryManager***

1818 Interface QueryManager provides a common base class for all other specialized 1819 QueryManager sub-classes in the API. It has the following methods:

1820

|  |  |
| --- | --- |
| **Method Summary** | |
| RegistryObject | **getRegistryObject**(java.lang.String id) |
|  | Gets the RegistryObject specified by the Id. |
| RegistryObject | **getRegistryObject**(java.lang.String id, |
|  | java.lang.String objectType)  Gets the RegistryObject specified by the Id and type of object. |
| BulkResponse | **getRegistryObjects**() |
|  | Gets the RegistryObjects owned by the caller. |
| BulkResponse | **getRegistryObjects**(java.util.Collection objectKeys) |
|  | Gets the specified RegistryObjects. |
| BulkResponse | **getRegistryObjects**(java.util.Collection objectKeys, |
|  | java.lang.String objectTypes)  Gets the specified RegistryObjects. |

Sun Microsystems Page 72

JavaTM API for XML Registries April 10, 2002

|  |  |
| --- | --- |
| BulkResponse | **getRegistryObjects**(java.lang.String objectType) |
|  | Gets the RegistryObjects owned by the caller, that are of the specified type. |
| RegistryService | **getRegistryService**() |
|  | Returns the parent RegistryService that created this object. |

1821

1822 **9.2 Interface *BusinessQueryManager***

1823 Interface BusinessQueryManager provides a simple business-level API that

1824 provides the ability to query for the most important high-level interfaces in the

1825 information model.

1826 Many of the methods in this interface take similar arguments and have the same

1827 return type:

1828 **findQualifiers** - a Collection of find qualifiers as defined by the FindQualifier

1829 interface. It specifies qualifiers that effect string matching, sorting, and boolean

1830 predicate logic and so on.

1831 **namePatterns** - a Collection that may consist of either String or LocalizedString

1832 objects. Each String, or value within a LocalizedString, is a partial or full name 1833 pattern with wildcard searching as specified by the SQL -92 LIKE specification.

1834 Unless otherwise specified in findQualifiers, this is a logical OR and a match on

1835 any name qualifies as a match for this criteria.

1836 **classifications** - a Collection of Classifications that classify the object. It is

1837 analogous to a catgegoryBag in UDDI. Unless otherwise specified in

1838 findQualifiers, this is a logical AND and requires a match on ALL specified

1839 Classifications to qualify as a match for this criteria. A transient Classification 1840 may be created by the programmer using LifeCycleManager.createClassification

1841 to use in this Collection.

1842 **specifications** - a Collection of RegistryObjects that represent (proxy) a

1843 technical specification. It is analogous to a tModelBag in UDDI. Unless otherwise

1844 specified in findQualifiers, this is a logical AND and requires a match on ALL

1845 specified Specifications to qualify as a match for this criteria.

1846 **externalIdentifiers** - a Collection of ExternalIdentifiers that provide an external

1847 identifier for the object using an identification scheme such as DUNS. It is

1848 analogous to an identifierBag in UDDI. Unless otherwise specified in

1849 findQualifiers, this is a logical AND and requires a match on ALL specified

Sun Microsystems Page 73

JavaTM API for XML Registries April 10, 2002

1850 Classifications to qualify as a match for this criteria. A transient ExternalIdentifier 1851 may be created by the programmer using

1852 LifeCycleManager.createExternalIdentifier to use in this Collection.

1853 **externalLinks** - Ia Collection of ExternalLinks that link the object to content

1854 outside the registry. It is analogous to an overviewDoc in UDDI. Unless otherwise

1855 specified in findQualifiers, this is a logical AND and requires a match on ALL

1856 specified ExternalLinks to qualify as a match for this criteria.

1857 **BulkResponse** - Contains Collection of objects returned by the find methods.

1858

1859

**Method Summary**

BulkResponse

**findAssociations**(java.util.Collection findQualifiers,

java.lang.String sourceObjectId,

java.lang.String targetObjectId,

java.util.Collection associationTypes)

Finds all Associations that match ALL of the criteria specified by the parameters of this call.

BulkResponse

**findCallerAssociations**(java.util.Collection findQualif

iers, java.lang.Boolean confirmedByCaller, java.lang.Boolean confirmedByOtherParty, java.util.Collection associationTypes)

Finds all Associations owned by the caller that match ALL of the criteria specified by the parameters of this call.

ClassificationScheme

**findClassificationSchemeByName**(java.util.Collection fi

ndQualifiers, java.lang.String namePattern)

Find a ClassificationScheme by name based on the specified name pattern.

BulkResponse

**findClassificationSchemes**(java.util.Collection findQua

lifiers, java.util.Collection namePatterns, java.util.Collection classifications, java.util.Collection externalLinks)

Finds all ClassificationSchemes that match ALL of the criteria specified by the parameters of this call.

Concept

**findConceptByPath**(java.lang.String path)

Find a Concept based on the path specified.

BulkResponse

**findConcepts**(java.util.Collection findQualifiers,

Sun Microsystems Page 74

JavaTM API for XML Registries April 10, 2002

java.util.Collection namePatterns,

java.util.Collection classifications,

java.util.Collection externalIdentifiers,

java.util.Collection externalLinks)

Finds all Concepts that match ALL of the criteria specified by the parameters of this call.

BulkResponse

**findOrganizations**(java.util.Collection findQualifiers,

java.util.Collection namePatterns,

java.util.Collection classifications,

java.util.Collection specifications,

java.util.Collection externalIdentifiers,

java.util.Collection externalLinks)

Finds all Organizations that match ALL of the criteria specified by the parameters of this call.

BulkResponse

**findRegistryPackages**(java.util.Collection findQualifie

rs, java.util.Collection namePatterns,

java.util.Collection classifications,

java.util.Collection externalLinks)

Finds all RegistryPackages that match ALL of the criteria specified by the parameters of this call.

BulkResponse

**findServiceBindings**(Key serviceKey,

java.util.Collection findQualifiers,

java.util.Collection classifications,

java.util.Collection specifications)

Finds all ServiceBindings that match ALL of the criteria specified by the parameters of this call.

BulkResponse

**findServices**(Key orgKey,

java.util.Collection findQualifiers,

java.util.Collection namePatterns,

java.util.Collection classifications,

java.util.Collection specifications)

Finds all Services that match ALL of the criteria specified by the parameters of this call.

1860

1861 **9.2.1 Find Methods**

1862 The BusinessQueryManager interface defines a set of find methods for each

1863 key interface. Most find methods can return multiple objects of the type

1864 associated with the find method that match the specified search criteria.

Sun Microsystems Page 75

JavaTM API for XML Registries April 10, 2002

1884

1885

1886

1887

1865 **9.2.1.1 Collection Parameters**

1866 Any Collection parameter that is a Collection of objects is a homogeneous

1867 collection of objects of the appropriate type. If the Collection contains an object

1868 whose type is unexpected, the implementation must throw an

1869 UnexpectedObjectException.

1870 Unless noted otherwise, all Collection parameters in the JAXR API have the

1871 following usage semantics. If the Collection is null, the JAXR provider must treat 1872 it as if it were an empty Collection. A JAXR provider may use null Collection 1873 values as a hint to optimize the processing of the Collection.

1874 **9.2.1.2 Interface FindException**

1875 This exception is used in the event of failure or partial success during a find

1876 operation. A Collection of instances of this exception is returned in the

1877 BulkResponse for a find method invocation if FindExceptions are encountered.

1878 **9.2.2 Canonical Paths Syntax for Concepts**

1879 In the findConceptByPath method, the desired Concept is indicated via a

1880 canonical representation that uniquely identifies the absolute path leading from

1881 the ClassificationScheme to that Concept.

1882 The canonical path representation is defined by the following BNF grammar: 1883

|  |
| --- |
| canonicalPath ::= '/' schemeId conceptPath  conceptPath ::= '/' conceptValue  | '/' conceptValue ( conceptPath )? |

1888 In the above grammar, schemeId is the id attribute of the ClassificationScheme 1889 instance, and conceptValue is defined by NCName production as defined by

1890 [http://www.w3.org/TR/REC-xml-names/#NT-NCName](http://www.w3.org/TR/REC-xml-names/" \l "NT-NCName.).

1891 **9.2.2.1 Example of Canonical Path Representation**

1892 The following canonical path represents the Concept with value ‘UnitedStates’ 1893 with a parent Concept with value ‘NorthAmerica’ under a ClassificationScheme

1894 with id ‘Geography-id’.

1895

1896 /Geography-id/NorthAmerica/UnitedStates

Sun Microsystems Page 76

JavaTM API for XML Registries April 10, 2002

1897 **9.3 Interface *DeclarativeQueryManager***

1898 Interface DeclarativeQueryManager provides a more flexible generic API 1899 that provides the ability to perform ad hoc queries using a declarative query 1900 language syntax. Currently the only declarative syntaxes supported are SQL-92 1901 and OASIS ebXML Registry Filter Queries. Note that support of SQL queries is 1902 optional for OASIS ebXML Registries. If the target registry does not support SQL 1903 queries then methods calls on DeclarativeQueryManager should throw

1904 UnsupportedCapabilityException.

1905

1906

1907

|  |  |
| --- | --- |
| **Method Summary** | |
| Query | **createQuery**(int queryType, java.lang.String queryString) |
|  | Creates a Query object given a queryType (for example, QUERY\_TYPE\_SQL) and a String that represents a query in the syntax appropriate for queryType. |
| BulkResponse | **executeQuery**(Query query) |
|  | Executes a query as specified by query paramater. |

1908

1909 **9.3.1 Interface Query**

1910 The Query interface encapsulates a query in a declarative query language. 1911 Currently a Query can only be defined using an SQL-92 syntax or the OASIS 1912 ebXML Registry query syntax. In future support for other query languages such 1913 as XQuery may be added.

1914

1915

|  |  |
| --- | --- |
| **Method Summary** | |
| int | **getType**() |
| Gets the type of Query (e.g. SQL). |
| java.lang.String | **toString**() |
| Must print the String representing the query. |

Sun Microsystems Page 77

JavaTM API for XML Registries April 10, 2002

1916

1917 **9.3.2 Creating a Query**

1918 A JAXR client must first create a Query object to encapsulate its query in a 1919 supported declarative query syntax such as SQL. This is done by calling the 1920 createQuery factory method on the DeclarativeQueryManager. A JAXR

1921 provider may optionally perform client-side validation of the SQL query syntax 1922 and throw an InvalidRequestException when an invalid query is specified.

1923 In the absence of such validation, it is expected that the registry provider will 1924 detect the error, in which case a RegistryException will be returned in the 1925 BulkResponse.

1926 **9.3.3 Executing a Query**

1927 A JAXR client executes a declarative query encapsulated by a Query instance 1928 by invoking the executeQuery method of the DeclarativeQueryManager, 1929 giving it the Query object as parameter.

1930 **9.4 SQL Query Syntax**

1931 The syntax for the SQL Query is defined by a stylized use of a proper subset of 1932 the “SELECT” statement of Entry Level SQL defined by ISO/IEC 9075:1992, 1933 Database Language SQL [SQL], extended to include sql invoked routines 1934 (also known as stored procedures) as specified in ISO/IEC 9075-4 [SQL-PSM].

1935 **9.4.1 SQL Query Syntax Binding To Information Model**

1936 The SQL Queries are defined based upon the query syntax defined in [RIM] and 1937 a relational schema that is an algorithmic binding to the information model as 1938 described in the section titled “SQL Query Syntax Binding To [RIM]” in [RIM].

1939 **9.5 OASIS ebXML Registry Filter Query Syntax**

1940 The [ebRS] specification defines the filter query syntax for the OASIS ebXML 1941 Registry. This syntax is an XML syntax defined by an XML Schema.

1942 **9.6 Query Result**

1943 The executeQuery method returns a BulkResponse that contains a

1944 homogeneous collection of objects. The type of objects is defined by the FROM

1945 clause of the query. For example, SELECT from Organization WHERE ...

1946 returns a Collection of Organization instances.

Sun Microsystems Page 78

JavaTM API for XML Registries April 10, 2002

1947 **9.7 Federated Queries**

1948 A client may issue a federated query against multiple registry providers as if they 1949 were a single logical registry provider. A federated query is issued in a manner 1950 similar to a non-federated query by calling a method on either the

1951 BusinessQueryManager or DelarativeQueryManager interfaces. The only 1952 difference is that the client must use the BusinessQueryManager or

1953 DelarativeQueryManager interface that was obtained from a RegistryService 1954 of a FederatedConnection instance rather than of a primitive Connection.

1955 Federated query capability is an optional feature of a JAXR provider for version

1956 1.0.

1957

Sun Microsystems Page 79

JavaTM API for XML Registries April 10, 2002

1957 **10 Security Architecture**

1958 The JAXR API does not specify its own security mechanisms. Instead, the API 1959 defines some minimal methods that allow implementations to choose their 1960 underlying security mechanism.

1961 The minimal security-related methods in the JAXR API are aligned with the Java

1962 Authentication and Authorization Service (JAAS) and Java Secure Socket

1963 Extension (JSSE) specifications. These are a standard part of the Java 2

1964 Platform, Standard Edition (J2SETM) version 1.4 software. Support for earlier 1965 versions of the J2SE software is available through stand-alone versions of these 1966 packages.

1967 **10.1 Integrity**

1968 To ensure the integrity of a JAXR request to the target registry provider, the

1969 JAXR API provides the ability for the request to the registry provider to be signed

1970 using a digital certificate. All Level 1 JAXR providers must be capable of sending

1971 signed requests to the registry provider and receiving signed responses from the

1972 registry provider.

1973 The JAXR client does not directly sign requests, nor does it validate signed

1974 responses. Instead, this functionality is delegated to the Level 1 JAXR provider.

1975 **10.2 Confidentiality**

1976 To ensure the confidentiality of a JAXR request to the target registry provider, all 1977 JAXR providers (Level 0 and above) must be able to use SSL to communicate 1978 with a registry that is accessible over the HTTPS protocol. Use of the HTTPS 1979 protocol is transparent to the JAXR client except for the fact that the URL defines

1980 ‘https’ as the protocol.

1981 Level 1 JAXR providers may be capable of sending encrypted requests to the 1982 registry provider and receiving encrypted responses from the registry provider.

1983 The JAXR client does not directly encrypt requests, nor does it decrypt incoming

1984 encrypted responses from the registry. Instead, this functionality is delegated to 1985 the Level 1 JAXR provider.

Sun Microsystems Page 80

JavaTM API for XML Registries April 10, 2002 1986 **10.3 Authentication**

1987 From the perspective of the target registry provider, it is the JAXR provider, not 1988 the JAXR client, that is the registry provider’s client. The JAXR provider must 1989 authenticate with the registry provider as specified by the specification governing 1990 the registry provider (e.g. ebXML Registry, UDDI). Typically, such authentication

1991 occurs on certain privileged requests. For example, in UDDI, authentication is 1992 needed only on requests that use the UDDI publishing API to submit, update, or

1993 delete content.

1994 In all cases, the JAXR provider initiates the authentication requests, while the

1995 registry provider performs the actual authentication.

1996 The JAXR client does not directly initiate authentication. It does not need to know 1997 when authentication with the target registry is necessary nor how it must be

1998 done. Instead, this functionality is delegated to the JAXR provider.

1999 **10.3.1 Authentication Methods**

2000 A JAXR provider may support multiple authentication methods. A JAXR client

2001 may specify a particular authentication method using the

2002 javax.xml.registry.security.authenticationMethod connection 2003 property. If the provider does not support the specified authentication method 2004 then it must throw UnsupportedCapabilityException during the

2005 ConnectionFactory.createConnection call. The following authentication 2006 methods have been defined as normative authentication methods:

2007 o UDDI\_GET\_AUTHTOKEN is the get\_AuthToken protocol defined by

2008 [UDDI\_API2].

2009 o HTTP\_ BASIC is the HTTP basic authentication as specified in RFC2068.

2010 o CLIENT\_CERTIFICATE

2011 o MS\_ PASSPORT is the Microsoft Passport authentication mechanism.

2012 A provider may support one or more of these methods as well as additional

2013 provider-specific methods.

2014 **10.4 Authorization**

2015 The JAXR provider does not perform any authorization decisions. All requests

2016 from the client are relayed to the registry provider, along with any required

2017 authentication tokens. The registry provider may perform authorization checks in

2018 a registry provider-specific manner and report any authorization errors. The

2019 JAXR provider must map any such registry provider-specific errors to a

2020 JAXRException and deliver it to the client.

Sun Microsystems Page 81

JavaTM API for XML Registries April 10, 2002

2021 **10.5 Security Support in JAXR API**

2022 The following section describes the support for security features in the JAXR API. 2023 **10.5.1 User Registration**

2024 The JAXR API does not provide any support for registration of User credentials 2025 with a registry provider. Such user registration is a one-time activity that must be 2026 performed out-of-band with respect to the JAXR API.

2027 **10.5.2 Method Connection.setCredentials**

2028 public void **setCredentials**(Set credentials)

2029 throws JAXRException

2030 Allows the client to set the security credentials for the user that is currently

2031 associated with the client. The term credential has the meaning defined by the 2032 JAAS API. A credential may be any java.lang.Object instance that is supported 2033 as a credential type by the JAXR provider.

2034 A JAXR provider must support credentials in the form of either a digital certificate 2035 and private key or a username and password combination. A JAXR provider may 2036 support other forms of security credentials. A JAXR provider is not required to 2037 support certain forms of credentials if they are not supported by the target

2038 registry provider. For example, if digital certificates are not supported by UDDI, 2039 then a JAXR UDDI provider need not support digital certificate credentials.

2040 When a JAXR client specifies a digital certificate as a credential, it must do so

2041 using an instance of the javax.security.auth.x500.X500PrivateCredential class as 2042 defined by the JAAS API.

2043 When a JAXR client specifies a username and password combination as a 2044 credential, it must do so using an instance of the

2045 [java.net](http://java.net/).PasswordAuthentication class. This class is a simple container for a 2046 username and password.

2047 A JAXR provider must be able to use the appropriate credential from the

2048 credentials set for the Connection by the last setCredentials call and authenticate

2049 with the registry provider in a provider-specific manner.

2050 If a client dynamically changes its credentials, the change has no impact on the

2051 pre-existing RegistryService instance within that Connection. Nor does it have 2052 any impact on any information model objects created within that Connection.

Sun Microsystems Page 82

JavaTM API for XML Registries April 10, 2002

2053 **Appendix A Pre-defined Enumerations**

2054 This section defines the pre-defined enumerations used by the JAXR API. These 2055 enumerations are defined as Concept hierarchies (a ClassificationScheme and a 2056 set of child Concepts).

2057 The enumerations are listed using the following notational convention. Each

2058 enumeration is a separate section within this appendix. The name of the

2059 ClassificationScheme of each enumeration is the name of the enumeration and is

2060 used as the section title.

2061 **A.1 Identification of Pre-defined Enumerations**

2062 A client may identify the ClassificationScheme for a pre-defined enumeration 2063 using the name of the ClassificationScheme in the en\_US locale. Consequently, 2064 the ClassificationScheme for a pre-defined enumeration must always have a 2065 name defined in en\_US locale. So to identify a Concept with value of “Service” 2066 within the pre-defined enumeration ObjectType in a findConceptByPath call, a

2067 client writes the following code:

2068

2069 Concept serviceConcept = bqm.findConceptByPath(‘/ObjectType/Service’);

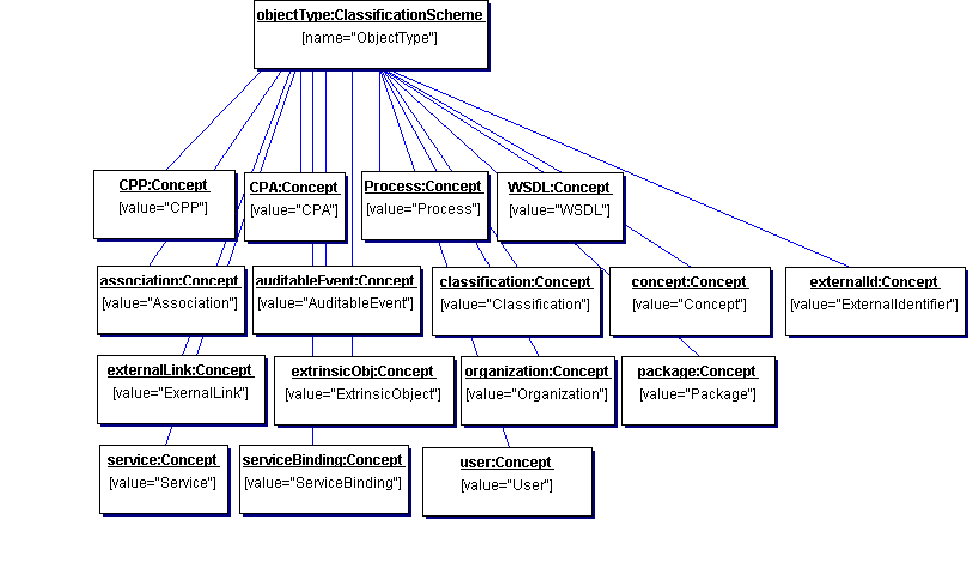
2070

2071 **A.2 Enumeration ObjectType**

2072 The ObjectType e numeration is used in the getObjectType method of 2073 RegistryObject.

Sun Microsystems Page 83

JavaTM API for XML Registries April 10, 2002

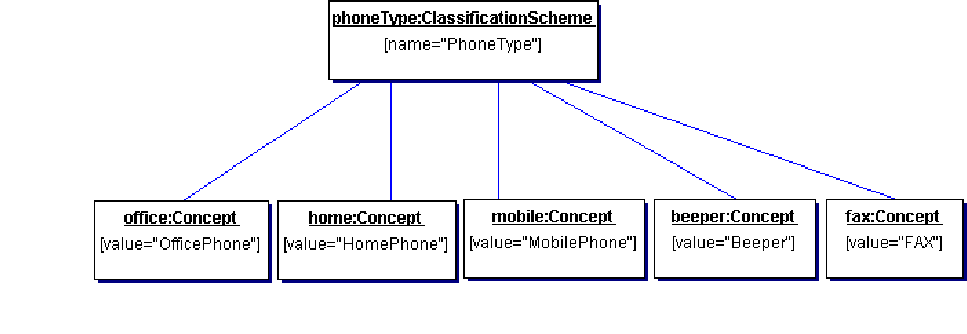


2074

2075 **Figure 24: Pre-defined enumeration ObjectType**

2076 **A.3 Enumeration PhoneType**

2077 This enumeration is used in the getPhoneType method of TelephoneNumber.



2078

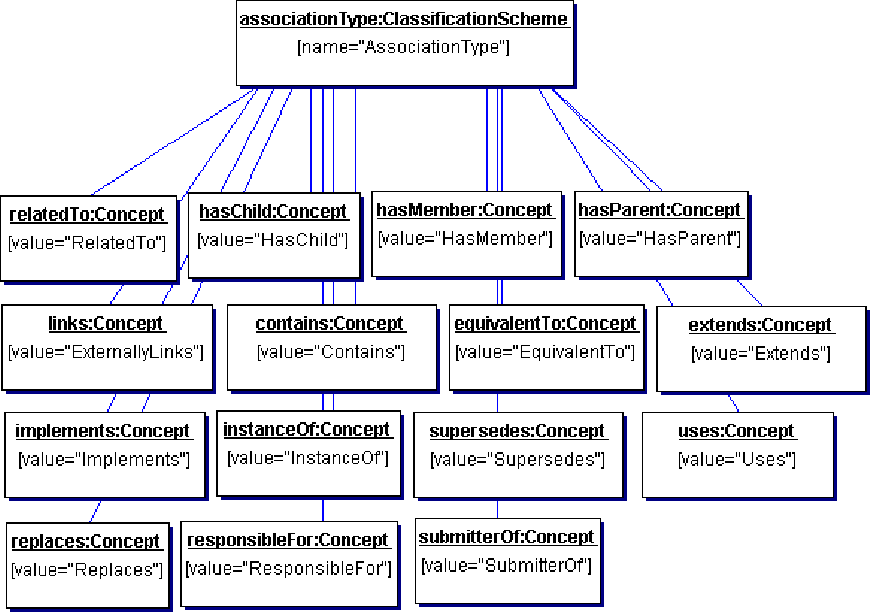
2079 **Figure 25: Pre-defined enumeration PhoneType**

2080 **A.4 Enumeration AssociationType**

2081 This enumeration is used in the getAssociationType method of Association.

Sun Microsystems Page 84

JavaTM API for XML Registries April 10, 2002



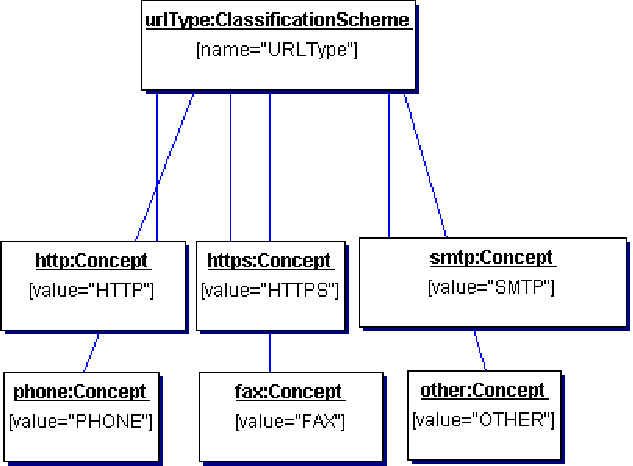
2082

2083 **Figure 26: Pre-defined enumeration AssociationType** 2084 **A.5 Enumeration URLType**

2085 This enumeration is used in classifying a ServiceBinding according to the type of 2086 access point it supports.

Sun Microsystems Page 85

JavaTM API for XML Registries April 10, 2002



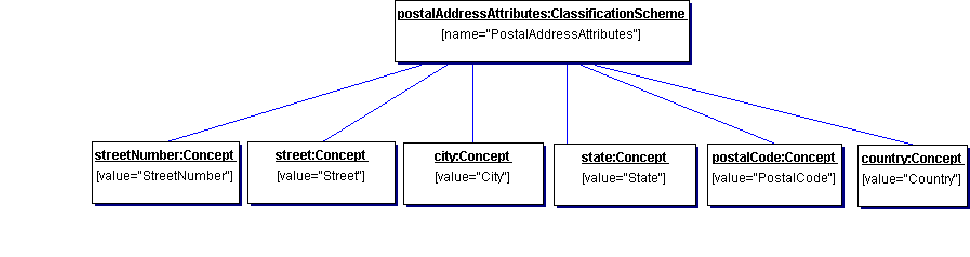
2087

2088 **Figure 27: Pre-defined enumeration URLType**

2089 **A.6 Enumeration PostalAddressAttributes**

2090 This enumeration is used to provide a mapping from the attributes of the 2091 PostalAddress class to any user-defined taxonomy.

2094 **Appendix B Semantic Equivalence of JAXR Concepts**



2092

2093 **Figure 28: Pre-defined enumerations for PostalAddressAttributes**

2095 This appendix describes those aspects of the JAXR APIthat allow the definition

2096 of semantic equivalence between two Concepts in potentially two different

2097 ClassificationSchemes.

Sun Microsystems Page 86

JavaTM API for XML Registries April 10, 2002

2098 When two Concepts are semantically equivalent, they represent the same notion 2099 or concept. It does not matter which node is the source and which is the target,

2100 since the semantics of this association implicitly apply bi-directionally.

2101 An example of the use of semantic equivalence is in the PostalAddress mapping 2102 for UDDI providers.

2103 A JAXR provider must allow a client to define semantic equivalences on a per-2104 connection basis using the javax.xml.registry.semanticEquivalences 2105 connection property. A JAXR provider may provide the ability to define the 2106 semantic equivalences in a provider-specific manner as global defaults for the

2107 provider.

2108 Each semantic equivalence is specified as a separate tuple in the single

2109 javax.xml.registry.semanticEquivalences connection property. Each

2110 individual semantic equivalence tuple consists of the id of two equivalent

2111 concepts separated by a comma ‘,’ separator. Within the property value,

2112 individual semantic equivalence tuples are separated by the ‘|’ character.

2113 The format is described below:

2114

2115 javax.xml.registry.semanticEquivalences=<id1>, <id2> | <id3>, <id4> | ....

2116 Spaces are allowed between tokens. The backslash ‘\’ character may be used as

2117 a continuation indicator, as allowed by Java property file syntax.

2118 An example follows below. Each property is on the same line but is shown 2119 wrapped due to the formatting limits of this document.

2120 A JAXR provider must ignore a semanticEquivalence property that is invalid for

2121 some reason (for example, the id specified was not that of a Concept). In such 2122 cases, the JAXR provider should emit a suitable warning to the user.

2123

|  |
| --- |
| javax.xml.registry.semanticEquivalences= \ urn:uuid:0a1324f7-6d4a-4d73-a088-9ab1d00c9a91, \ urn:uuid:23a5feac-26b9-4525-82fc-997885a0e6a2 | \ urn:uuid:1acf6ed2-cd6e-4797-aad8-8937a3cff88b, \ urn:uuid:152d6f28-cb56-4e5d-9f55-96b132def0e4 |

Sun Microsystems Page 87

2124 2125 2126 2127 2128

2129

JavaTM API for XML Registries April 10, 2002

2130 **Appendix C JAXR Mapping to ebXML Registry**

2131 This appendix describes how the JAXR information model maps to the ebXML 2132 Registry information model defined in [ebXML-RIM]. Note that the JAXR 2133 information model used [ebXML-RIM] as its starting point. Consequently, the 2134 mapping from JAXR to ebXML Registry is often direct.

2135 **C.1.1 Mapping of Interfaces**

2136 Table 1 describes the mapping between the interfaces in the ebXML and JAXR 2137 information models respectively. Both models use UML interfaces and classes.

2138

2139 **Table 1: Mapping of ebXML Interfaces to JAXR Interfaces**

|  |  |  |
| --- | --- | --- |
| **EbXML** | **JAXR** | **Description** |
| Association | Association | Identical definition |
| AuditableEvent | AuditableEvent | Identical definition |
| Classification | Classification | Identical definition |
| ClassificationNode | Concept | Name change only |
| EmailAddress | EmailAddress | Identical definition |
| ExternalIdentifier | ExternalIdentifier | Identical definition |
| ExternalLink | ExternalLink | Identical definition |
| ExtrinsicObject | ExtrinsicObject | Identical definition |
| Organization | Organization | Identical definition |
| RegistryPackage | RegistryPackage | Identical definition |
| PersonName | PersonName | Identical definition |
| PostalAddress | PostalAddress | Identical definition |
| RegistryEntry | RegistryEntry | Factored slots-related methods into ExtensibleObject |
| RegistryObject | RegistryObject | Change name of Id attribute to Key |
| Service | Service | Identical definition |
| ServiceBinding | ServiceBinding | Identical definition |
| Slot | Slot | Identical definition |

Sun Microsystems Page 88

JavaTM API for XML Registries April 10, 2002

|  |  |  |
| --- | --- | --- |
| SpecificationLink | SpecificationLink | Identical definition |
| TelephoneNumber | TelephoneNumber | Identical definition |
| User | User | Identical definition |
| Versionable | Versionable | Identical definition |

2140 **C.1.2 Mapping of New Classes In JAXR To ebXML**

|  |  |
| --- | --- |
| **JAXR** | **Description** |
| ExtensibleObject | Factored slots-related methods from RegistryObject into ExtensibleObject. No impact on mapping. |
| Key | Maps to an id of type String. No real impact on mapping. |

2141

2142 **C.1.3 ebXML Functionality Not Supported By JAXR**

2143 The following table declares all ebXML functionality that is not accessible via 2144 JAXR API. Any potential omissions from this list are specification errors and 2145 should be reported.

2146

ebXML

|  |  |  |
| --- | --- | --- |
| **Feature** | **Disposition** | **Description** |
|  |  | Currently, all functionality of OASIS ebXML Registry is supported. |

2147 **Appendix D JAXR Mapping To UDDI**

2148 This appendix describes how the JAXR information model maps to the UDDI 2149 XML data structure as defined in version 2.0 of the UDDI specification [UDDI-2150 DS]. UDDI data structures are described in an XML format.

2151 **D.1 Mapping of UDDI Inquiry API Calls To JAXR**

2152 The following table shows the mapping from UDDI Inquiry API methods to JAXR

2153 methods. Unless otherwise qualified, the JAXR interface is

2154 BusinessQueryManager.

2155

2156 Sun Microsystems Page 89

JavaTM API for XML Registries April 10, 2002

**UDDI Method**

Business-

QueryManager

**Method**

**Comments**

find\_binding

findServiceBindings

No comments

find\_business

findOrganizations

No comments

find\_related\_business

findAssociations

Will require traversing the association to get the related business in separate API call

find\_service

findServices

No comments

find\_tModel

findConcepts, findClassification-Schemes

No comments

get\_bindingDetail

Not needed

Handled transparently by JAXR provider

get\_businessDetail

Not needed

Handled transparently by JAXR provider

get\_businessDetailExt

Unsupported

Use RegistryService.

makeRegistrySpecificRequest

get \_serviceDetail

Not needed

Handled transparently by JAXR provider

get\_tModelDetail

Not needed

Handled transparently by JAXR provider

2156

2157 **D.2 Mapping of UDDI Publisher API Calls to JAXR**

2158 The following table shows the mapping from UDDI Publisher API methods to

2159 JAXR methods. Unless otherwise qualified, the JAXR interface is

2160 BusinessLifeCycleManager.

2161

**UDDI Method**

BusinessLifeCycle-

**Manager Method**

**Comments**

add\_publisherAssertions

saveAssociations, confirmAssociation

No comments

delete\_binding

deleteServiceBindings

No comments

Sun Microsystems Page 90

JavaTM API for XML Registries April 10, 2002

2162

delete\_business

deleteOrganizations

No comments

delete\_publisherAssertions

deleteAssociations

No comments

delete\_service

deleteServices

No comments

delete\_tModel

deleteClassification- Schemes and deleteConcepts

In UDDI delete\_tModel does not delete the tModel. It simply hides it from find\_tModel calls. The QueryManager.getRegistry-Object calls will still return the deleted tModel after a deleteConcepts or deleteClassificationSchemes call.

discard\_authToken

Not needed

Handled transparently by JAXR provider

get\_assertionStatusReport

BusinessQuery- Manager.-

findCallerAssociations

No comments

get\_authToken

Not needed

Handled transparently by JAXR provider

get\_publisherAssertions

BusinessQuery- Manager.-

findCallerAssociations

JAXR provider must transparently authenticate with UDDI provider

get\_registeredInfo

QueryManager. getRegistryObjects( objectType)

JAXR provider must transparently authenticate with UDDI provider

save\_binding

saveServiceBindings

No comments

save\_business

saveOrganizations

No comments

save\_service

SaveServices

No comments

save\_tModel

saveClassificationSche

mes and saveConcepts

No comments

set\_publisherAssertions

saveAssociations

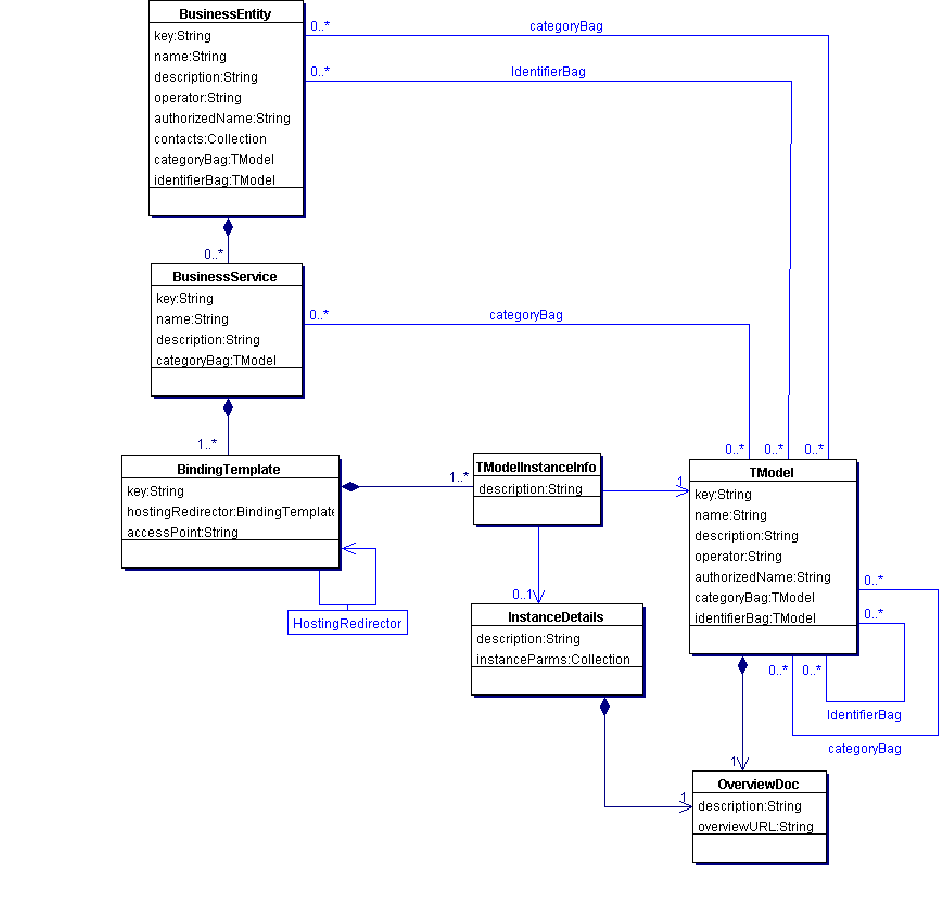
No comments

Sun Microsystems Page 91

JavaTM API for XML Registries April 10, 2002

2163 **D.3 Simplified UML Model For UDDI Information Model**

2166 Figure 29 shows a simplified UML model representing the UDDI information 2167 model. Note that the model is not an exact rendering of the UDDI information



2164

2165 **Figure 29: Simplified UML Model for UDDI Information Model**

2168 model but has been simplified to aid the reader’s understanding.

Sun Microsystems Page 92

JavaTM API for XML Registries April 10, 2002

2169 **D.4 Mapping of JAXR Attributes to UDDI**

2170 The UDDI data types are not extensible. It is therefore not always possible to 2171 map an attribute from a JAXR interface to UDDI. For example, UDDI does not 2172 support majorVersion and minorVersion. A JAXR provider for UDDI must throw 2173 an UnsupportedCapabilityException when a client attempts to call a setter 2174 method for an attribute that has no mapping in UDDI (e.g.

2175 RegistryEntry.setMajorVersion). Note that such methods are designated as level 2176 1 methods for the convenience of the JAXR client programmer.

2177 Similarly, not all JAXR interfaces have a mapping to UDDI. For example, the 2178 JAXR RegistryPackage interface has no mapping to UDDI. The

2179 LifeCycleManager.createObject method and any other related factory methods 2180 must throw an UnsupportedCapabilityException when a client attempts to create 2181 an object that cannot be mapped to UDDI.

2182 **D.5 Mapping of UDDI Attributes to JAXR**

2183 The JAXR specification used the following approaches, listed in order of

2184 preferences, when specifying the mapping between UML attributes of the JAXR

2185 model and UDDI:

2186 1. Map UDDI attribute to a statically defined (non-Slot) JAXR attribute within

2187 a JAXR interface

2188 2. Map UDDI attribute to a dynamically defined Slot attribute within an

2189 instance of a JAXR interface

2190

2191 **D.6 Mapping of Interfaces**

2192 This section provides the mapping of the highest-level UDDI data structures to 2193 the interfaces defined by the JAXR information model. The table provides an 2194 entity-level mapping and subsections discuss element/attribute-level mapping for 2195 each key concept. Since JAXR defines its information model in terms of 2196 interfaces, most of the UDDI entity attributes are mapped to operations on the 2197 JAXR information model objects.

2198

|  |  |  |  |
| --- | --- | --- | --- |
| **UDDI JAXR** | | | **Description** |
| businessEntity | Organization |  | |
| businessService | Service |  | |
| bindingTemplate | ServiceBinding | See D.6.3 for details. | |

Sun Microsystems Page 93

JavaTM API for XML Registries April 10, 2002

|  |  |  |
| --- | --- | --- |
| tModel (fingerprint) | Concept | See D.6.4 for details. |
| tModel (namespace) | ClassificationScheme | See D.6.4 for details. |
| discoveryURL | ExternalLink | See D.6.1.1 for details. |
| contact | User |  |
| identifierBag | Collection of  ExternalIdentifier  instances | See D.7.2 for details. |
| categoryBag | Collection of  Classification instances | See D.7.3 for details. |
| address | PostalAddress | See D.6.1.3 for details. |
| overviewDoc | ExternalLink | See D.6.4.5 for details. |
| keyedReference (in categoryBag) | Classification | See D.7.1 for details. |
| keyedReference (in identityBag) | ExternalIdentifier | See D.7.1 for details. |

2199

2200 **D.6.1 UDDI businessEntity**

2201 businessEntity is one of the four core data structures in UDDI. The

2202 businessEntity maps to Organization in the JAXR information model. The

2203 following table shows the attribute level mapping between a UDDI businesEntity

2204 and a JAXR Organization.

2205

|  |  |  |
| --- | --- | --- |
| **businessEntity** | **Organization** | **Description** |
| businessKey | Organization.getKey |  |
| authorizedName | Organization.getSlot | Read-only Slot named *authorizedName* of type String. |
| operator | Organization.getSlot | Read-only Slot named *operator* of type String. |
| discoveryURL | Organization.getExternalLinks | businessEntity contains a list of discoveryURLs while Organization contains a collection of external links. |

Sun Microsystems Page 94

JavaTM API for XML Registries April 10, 2002

|  |  |  |
| --- | --- | --- |
| name | Organization.getName |  |
| description | Organization.getDescription |  |
| contact | Organization.getUsers | JAXR designates one contact as the primary contact while UDDI does not. The mapping will assume the first UDDI contact to be the primary contact in JAXR. |
| businessServices | Organization.getServices | businessService maps to Service interface. |
| identifierBag | Organization.  getExternalIdentifiers | See D.7.2. |
| categoryBag | Organization.  getClassifications | See D.7.3. |

2206 **D.6.1.1 UDDI discoveryURL**

2207 In UDDI, the default discoveryURL is assigned by the UDDI registry and is used 2208 to retrieve the XML document for the businessEntity and everything contained 2209 within it. Any additional discoveryURLs are assigned by the submitter, and

2210 provide links to external content that provides information about the

2211 businessEntity (referred to as discovery documents in UDDI).

2212 The default discoveryURL is identified by having a useType of either

2213 businessEntity or businessEntityExt. Any other useType value indicates an

2214 optional discoverURL.

2215 **D.6.1.1.1 Getting a discoveryURL from UDDI**

2216 When a businessEntity is retrieved from UDDI and mapped to a JAXR

2217 Organization, all discovery URLs are mapped to an ExternalLink. The first

2218 ExternalLink in the Collection returned by getExternalLinks method on

2219 Organization object must map to the default registry provider assigned

2220 discoveryURL. All other ExternalLinks must map to the optional discoveryURLs.

Sun Microsystems Page 95

JavaTM API for XML Registries April 10, 2002

2221 **D.6.1.1.2 Saving discoveryURL to UDDI**

2222 When a client saves an Organization to UDDI, the default discoveryURL is

2223 assigned to the corresponding businessEntity by the registry provider and is not

2224 provided by the JAXR client. All ExternalLink instances associated with the

2225 Organization are mapped to optional discoveryURL instances such that the name

2226 of the ExternalLink is mapped to the useType attribute of the discoveryURL.

2227

|  |  |  |
| --- | --- | --- |
| **DiscoveryURL** | **ExternalLink** | **Description** |
| useType | ExternalLink.getName |  |
| url value | ExternalLink.getExternalURI |  |

2228 **D.6.1.2 UDDI contact**

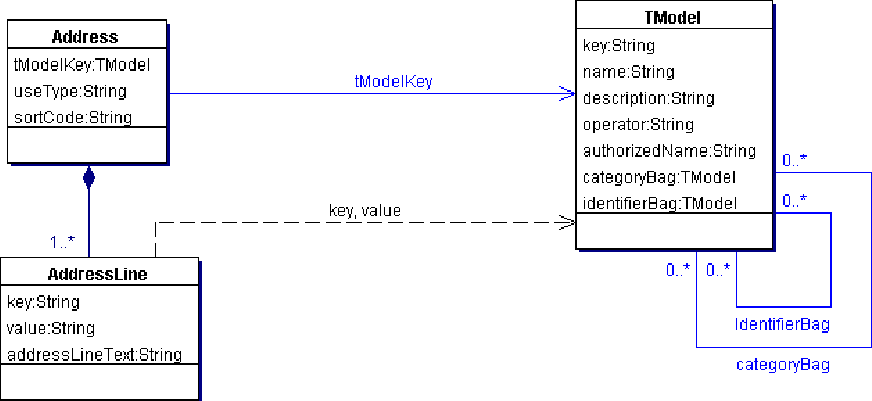
2229 The UDDI contact element maps to the interface User in JAXR as follows: 2230

|  |  |  |
| --- | --- | --- |
| **Contact** | **User** | **Description** |
| useType | User.getType |  |
| description | User.getDescription |  |
| personName | User.getPersonName.getFullName |  |
| phone | User.getTelephoneNumbers.getNumber |  |
| email | User.getEmailAddresses |  |
| address | User.getPostalAddress |  |

2231

Sun Microsystems Page 96

JavaTM API for XML Registries April 10, 2002 2232 **D.6.1.3 UDDI address**



2233

2234 **Figure 30: UDDI Information Model for Address**

2235 A UDDI address has an ordered Collection of addressLine instances. Each 2236 addressLine is a String. However in UDDI V2, each addressLine may be 2237 attached a meaning by linking it via a key-value pair of attributes to a taxonomy 2238 element defined under a taxonomy represented by a tModel. The tModel is 2239 referenced by a tModelKey within the address. Thus in UDDI an address is 2240 unstructured by default and can optionally be given meaning.

2241 In JAXR, PostalAddress is a structured interface with well-defined attributes for 2242 street, city, postal code, country etc. This brings about an issue of mapping 2243 between structured information in JAXR and unstructured information in UDDI. 2244 The solution is to use semantic equivalence mapping capabilities in JAXR API as 2245 described in Appendix B, as follows.

2246 1. The user or system administrator defines one or more user-defined

2247 ClassificationSchemes (a.k.a postal address schemes) representing UDDI

2248 tModels commonly used by address as a postal address scheme.

2249 2. The user or system administrator defines semantic equivalence between

2250 each Concept in the pre-defined PostalAddressAttributes

2251 ClassificationScheme and one or more Concepts in the user-defined

2252 postal scheme used by address.

Sun Microsystems Page 97

JavaTM API for XML Registries April 10, 2002

2253 3. The client sets a default postalAddressScheme (using the

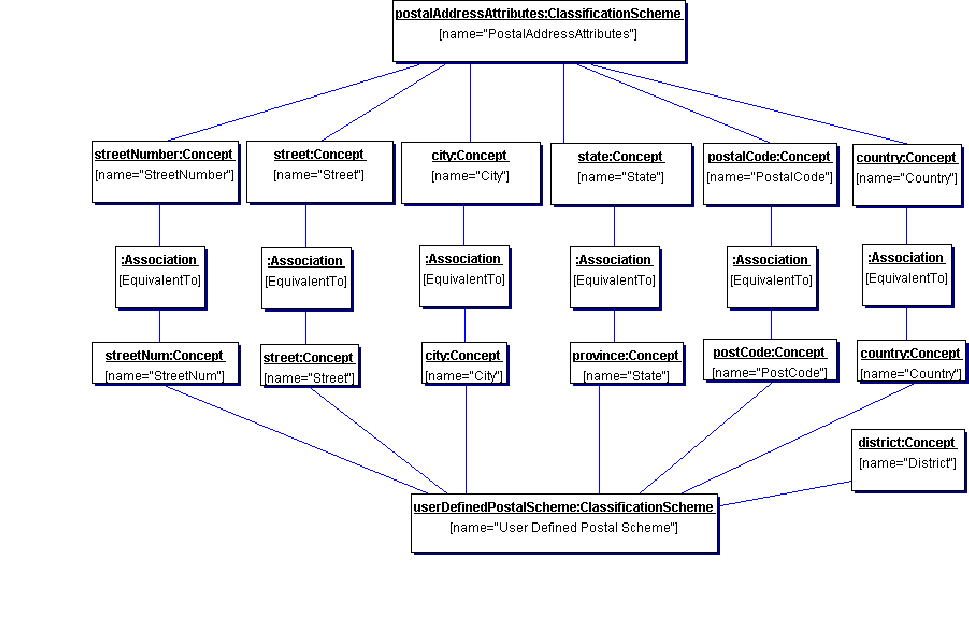
2254 javax.xml.registry.postalAddressScheme connection property) on the

2255 Connection instance and/or sets a postal address scheme on a specific

2256 PostalAddress. The postal address scheme for a PostalAddress over-

2257 rides the default postal scheme set on the Connection.

2261 A JAXR provider for UDDI must use the semantic equivalences defined with 2262 user-defined ClassificationScheme to validate and properly map PostalAddress 2263 attributes to addressLines with keyed references to the tModel corresponding to 2264 the postal scheme for the PostalAddress. This allows a JAXR client programmer 2265 to use the set/get methods defined in the PostalAddress class to conveniently set 2266 and get the values of the PostalAddress attributes such as city, stateOrProvince, 2267 country etc.



2258

2259 **Figure 31: Semantic Equivalence and Mapping of User Defined Postal Scheme to**

2260 **PostalAddress Attribute**

2268 **D.6.1.3.1 Mapping of PostalAddress During Save Operations**

2269 The JAXR UDDI provider must map the PostalAddress to UDDI during a Save 2270 Operation as follows:

2271 o The PostalAddress must map to an address element.

Sun Microsystems Page 98

JavaTM API for XML Registries April 10, 2002

2272 o If there is a postal scheme defined on the PostalAddress, or if a default

2273 postal scheme is defined on the RegistryService, then a tModelKey

2274 attribute must be set to the Id of the postal scheme. Otherwise, the

2275 tModelKey must not be specified.

2276 o Each attribute of PostalAddress must map to an addressLine element.

2277 o The order of addressLine elements must match the order of

2278 PostalAddressAttributes as shown left to right in Figure 31 above for the

2279 Concepts under postalAddressAttributes scheme (second row from top).

2280 o If a tModelKey has been set on the address element and there is a

2281 semantic equivalence found between a PostalAddress attribute and the

2282 postal scheme, then a name and value attribute pair is specified for the

2283 corresponding addressLine element. The name and value are the name

2284 and value of the Concept in the postal scheme that was equivalent to the

2285 PostalAddress attribute. If no semantic equivalence was found then the

2286 name attribute and the value attribute must be as the name and value of

2287 the corresponding Concept in PostalAddressScheme.

2288 o If no tModelKey has been set on the address element then each

2289 addressLine element should specify the name and value attribute as

2290 defined by the name and value of the corresponding Concept in

2291 PostalAddressScheme.

2292 **D.6.1.3.2 Mapping of UDDI address During Find Operations**

2293 The JAXR UDDI provider must map the UDDI address element to JAXR d uring a 2294 find operation as follows:

2295 o The address element must map to a PostalAddress instance.

2296 o If there is a tModelKey defined for the address element and it matches a

2297 postal scheme, the postalScheme must be set to the matching postal

2298 scheme.

2299 o An address line is mapped to a PostalAddress attribute if a match to a

2300 PostalAddress attribute is found. The match is found if the following

2301 conditions are true:

2302 o A postal scheme has been set and

2303 o The value attribute of the addressLine matches a value attribute of

2304 a Concept in the postal scheme and

2305 o That postal scheme concept with matching value has a semantic

2306 equivalence with a PostalAddress attribute

Sun Microsystems Page 99

JavaTM API for XML Registries April 10, 2002

2307 o An address line is mapped to a value in the Collection of values in a slot

2308 named addressLines defined for the PostalAddress if no match is found to

2309 a PostalAddress attribute.

2310

2311 The following table summarizes the attribute mappings between the UDDI 2312 address element and JAXR PostalAddress interface.

2313

|  |  |  |
| --- | --- | --- |
| **Address** | **PostalAddress** | **Description** |
| addressLines | PostalAddress.getSlot | Mapped to attributes of PostalAddress if semantic equivalence established. Otherwise mapped to a Slot named addressLines which has a Collection of values |
| useType | PostalAddress.getType |  |
| sortCode | PostalAddress.getSlot | Slot named sortCode |

2314

2315 **D.6.2 UDDI businessService**

2316 businessService in UDDI represents a logical group of services, which have

2317 common classifications. It is functionally minimal and really serves as a grouping 2318 of bindingTemplates.

2319 businessService maps directly to the Service interface in the JAXR information 2320 model.

2321

|  |  |  |
| --- | --- | --- |
| **businessService** | **Service** | **Description** |
| businessKey | Service.getProvidingOrganization.getKey |  |
| serviceKey | Service.getKey |  |
| name | Service.getName |  |
| description | Service.getDescription |  |
| bindingTemplates | Service.getServiceBindings |  |
| categoryBag | Service.getClassifications | See D.7.3. |

Sun Microsystems Page 100

JavaTM API for XML Registries April 10, 2002

2322

2323 **D.6.3 UDDI bindingTemplate**

2324 The bindingTemplate is another one of the core concepts in UDDI and serves not 2325 only to provide means of accessing (an entry point to) the desired service but

2326 also to carry with it the technical specification for the service. These technical 2327 specifications may be either in the form of a text-based document or in an 2328 interface definition language such as WSDL.

2329 The UDDI bindingTemplate element maps to the JAXR ServiceBinding interface

2330 as follows:

2331

|  |  |  |
| --- | --- | --- |
| **bindingTemplate** | **ServiceBinding** | **Description** |
| bindingKey | ServiceBinding.getKey |  |
| serviceKey | ServiceBinding.getService.getKey |  |
| description | ServiceBinding.getDescription |  |
| accessPoint | ServiceBinding.getAccessURI | URLType attribute in accessPoint is mapped by classifying the ServiceBinding with a sub-concept of URLType Concept (A.5). Default urlType is http. |
| hostingRedirector | ServiceBinding. getTargetBinding | There is only one element, bindingKey, in this structure and it maps to targetBinding attribute |
| tModelInstanceDetails | Not mapped explicitly as it is just a Collection |  |
| tModelInstanceInfo | Mapped to a SpecificationLink | See D.10 for mapping example. |
| instanceDetail | Mapped to a SpecificationLink | See D.10 for |

Sun Microsystems Page 101

JavaTM API for XML Registries April 10, 2002

mapping example.

2332

2333 **D.6.3.1 tModelInstanceInfo and instanceDetails**

2334 Both tModelInstanceInfo and instanceDetails are combined together and mapped

2335 to a single SpecificationLink instance.

2336 The registryObject attribute in the SpecificationLink instance is the Concept

2337 representing the tModel providing the specification fingerprint for the

2338 bindingTemplate.

2339 The description of the instanceDetails maps to the usageDescription attribute of 2340 the SpecificationLink instance. The instanceParms maps to the

2341 usageParameters attributes of the SpecificationLink instance. Note that JAXR 2342 allows multiple usageParameters while UDDI allows a single instanceParms. 2343 Thus a UDDI provider must throw an InvalidRequestException if a client attempts 2344 to set more than one usageParameter on a SpecificationLink instance.

2345 **D.6.4 tModel**

2346 In UDDI, tModel is an overloaded concept that can be used for a few different 2347 purposes. The following are two broad categories of purposes that tModels serve

2348 in UDDI:

2349 o To serve as a namespace for a taxonomy (e.g. NAICS) or identification

2350 scheme (DUNS)

2351 o To serve as a fingerprint or proxy for a technical specification that lives

2352 outside the registry in a bindingTemplate

2353 In the JAXR the above two uses of tModel are modeled separately. The 2354 namespace use is modeled with the ClassificationScheme interface, while the 2355 technical fingerprint use is modeled with any RegistryObject which in case of a 2356 UDDI provider must be a Concept. The

2357 SpecificationLink.getSpecificationObject method must return a 2358 Concept instance for a UDDI provider.

2359 **D.6.4.1 tModel Mapping to ClassificationScheme**

|  |  |  |
| --- | --- | --- |
| **tModel** | **Concept** | **Description** |
| tModelKey | ClassificationScheme.getKey |  |
| authorizedName | ClassificationScheme.getSlot | Read-only Slot named |

Sun Microsystems Page 102

JavaTM API for XML Registries April 10, 2002

|  |  |  |
| --- | --- | --- |
|  |  | *authorizedName* |
| operator | ClassificationScheme.getSlot | Read-only Slot named *operator* |
| name | ClassificationScheme.getName |  |
| description | ClassificationScheme.getDescription |  |
| overviewDoc | ClassificationScheme.getExternalLinks | See D.6.4.4. |
| identifierBag | ClassificationSche me.getExternalIdentifiers | See D.7.2. |
| categoryBag | ClassificationScheme.getClassifications | See D.7.3. |

2360

2361 D.6.4.2 tModel Mapping to Concept

**2362**

|  |  |  |
| --- | --- | --- |
| **tModel** | **Concept** | **Description** |
| tModelKey | Concept.getKey |  |
| authorizedName | Concept.getSlot | Read-only Slot named *authorizedName* |
| operator | Concept.getSlot | Read-only Slot named *operator* |
| name | Concept.getName |  |
| description | Concept.getDescription |  |
| overviewDoc | Concept.getExternalLinks | See D.6.4.4. |
| identifierBag | Concept.getExternalIdentifiers | See D.7.2. |
| categoryBag | Concept.getClassifications | See D.7.3. |

2363

2364 **D.6.4.3 Mapping of tModels During JAXR Find Operations**

2365 During JAXR find operations, the JAXR provider must be able to determine 2366 whether a tModel in UDDI is a namespace tModel or whether it is a fingerprint 2367 tModel. This is necessary in order to decide whether to map the tModel to a 2368 ClassificationScheme or Concept.

Sun Microsystems Page 103

JavaTM API for XML Registries April 10, 2002

2369 The JAXR UDDI provider must use the catgegoryBag information association 2370 with a UDDI tModel in order to make the correct mapping to a JAXR

2371 ClassificationScheme or Concept.

2372 A JAXR provider must map a UDDI tModel to a ClassificationScheme if the 2373 tModel’s categoryBag has a keyedReference that uses the well-known uddi-2374 org:types taxonomy in UDDI (with tModelKey uuid:C1ACF26D-9672-4404-2375 9D70-39B756E62AB4) for its tModel and uses any of the following taxonomy 2376 values:

2377 o Identifier

2378 o Namespace

2379 o Categorization

2380 o PostalAddress

2381 In all other cases, a JAXR provider must map a UDDI tModel to a Concept.

2382 Note that it is possible that a UDDI tModel was intended to be a

2383 ClassificationScheme but was not properly categorized in UDDI. In such cases 2384 the tModel would be mapped to a Concept instead of a ClassificationScheme. 2385 The user must explicitly determine which such Concepts are actually

2386 ClassificationSchemes and then use the createClassificationScheme(Concept) 2387 method of LifeCycleManager to safely cast the Concept to a

2388 ClassificationScheme. Note that such cases indicate problems within UDDI 2389 content and should be reported to the content’s owner.

2390 **D.6.4.4 Mapping to tModels During JAXR Save Operations**

2391 During JAXR Save operations, ClassificationSchemes and Concepts that have 2392 no parent or ClassificationScheme are mapped to tModels.

2393 It is suggested but not required that a JAXR provider for UDDI attempt to

2394 categorize tModels based upon information available on their intended usage.

2395 ClassificationSchemes related tModels *may* be automatically categorized by the

2396 well-known uddi-org:types taxonomy in UDDI (with tModelKey

2397 uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4) as follows:

2398 o PostalAddress schemes are assigned the taxonomy value of

2399 PostalAddress

2400 o ClassificationSchemes used for classification purposes are assigned the

2401 taxonomy value of categorization

2402 o ClassificationSchemes used for identification purposes are assigned the

2403 taxonomy value of identification

Sun Microsystems Page 104

JavaTM API for XML Registries April 10, 2002

2404 o If there is not enough information then the default taxonomy value is

2405 categorization.

2406 Specification related tModels mapped from Concept may be automatically 2407 categorized by the well-known uddi-org:types taxonomy in UDDI (with 2408 tModelKey uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4) as follows:

2409 o The keyed reference is assigned a taxonomy value of *specification*. 2410 **D.6.4.5 overviewDoc**

2411 An overviewDoc maps to an ExternalLink in JAXR. The ExternalLink is

2412 associated with the Concept or ClassificationScheme that the tModel is mapped 2413 to. Since UDDI allows only one overviewDoc on a tModel or a instanceDetails, a 2414 JAXR UDDI provider must throw UnsupportedCapabilityException if more than 2415 one ExternalLink is added to a Concept (tModel), ClassificationScheme (tModel) 2416 or SpecificationLink (instanceDetails). A JAXR UDDI provider should also throw 2417 UnsupportedCapabilityException if an ExternalLink is added to any other object

2418 besides Organization, Concept, ClassificationScheme or SpecificationLink.

2419

|  |  |  |
| --- | --- | --- |
| **OverviewDoc** | **ExternalLink** | **Description** |
| description | ExternalLink.getDescription |  |
| overviewURL | ExternalLink.getExternalURI |  |

2420

2421 **D.7 Mapping of Common Data Types**

2422 So far, we have described the highest-level mapping between the main data 2423 structures in UDDI and interfaces in the JAXR information model. The 2424 subsequent section describes the mapping between data structures that are 2425 commonly reused in UDDI and the JAXR API.

2426 **D.7.1 keyedReference**

2427 A keyedReference element is used either to contain a group of

2428 classifications or to contain a group of identifiers for an object. To that

2429 end, keyedReference can map to either ExternalIdentifiers or Classifications.

2430 For this reason, there are actually 2 tables specifying each individual

2431 mapping.

2432

2433 When keyedReference is being used in an identifierBag, it is mapped to a JAXR

2434 ExternalIdentifier. A JAXR information model object being marshaled to XML for

Sun Microsystems Page 105

JavaTM API for XML Registries April 10, 2002

2435 UDDI will have all its external identifiers marshaled into identifierBag. Similarly, 2436 keyedReferences in an identifierBag in a UDDI response are un-marshaled into 2437 ExternalIdentifiers by the JAXR provider.

2438

|  |  |  |
| --- | --- | --- |
| **keyedReference** | **ExternalIdentifier** | **Description** |
| tModelKey | ExternalIdentifier.getIdentificationScheme().getKey | From the RegistryObject |
| keyName | ExternalIdentifier.getName | From the RegistryObject. This is symbolic (such as Tax Id). |
| keyValue | ExternalIdentifier.getValue | This is the unique id (e.g. tax id), which identifies the ebusiness entity. |

2439

2440 When keyedReference is used in a categoryBag, it is mapped to a JAXR

2441 Classification.

2442

2443 When a JAXR object is being marshaled to XML for UDDI, all its Classifications

2444 are marshaled into categoryBag according to the mapping described below.

2445 When the Classification uses an internal taxonomy, the JAXR provider for UDDI

2446 must validate all keyValues in the resulting UDDI keyedReference.

2447

2448 In UDDI a keyedReference can have only one keyName. In contrast in the JAXR

2449 API, a Classification or an ExternalIdentifier may have multiple key names as

2450 defined by the name attribute. When a keyedReference in mapped to a JAXR

2451 Classification or an ExternalIdentifier, the keyValue must be mapped to the

2452 LocalizedString in the default Locale for the client. When a JAXR Classification or

2453 an ExternalIdentifier is mapped to a UDDI keyedReference, the keyValue is

2454 chosen using the following precedence rules:

2455

2456 1. Use the value specified in LocalizedString for the default locale if available

2457 2. Use the value specified in LocalizedString for the en\_US locale if available

2458 3. Use the first available name in any locale if available

2459 4. If all of above fail then do not specify the keyName

2460

|  |  |  |
| --- | --- | --- |
| **keyedReference** | **Concept** | **Description** |

Sun Microsystems Page 106

JavaTM API for XML Registries April 10, 2002

|  |  |  |
| --- | --- | --- |
| tModelKey | Classification.getClassificationScheme.getKey |  |
| keyName | Classification.getName | This is the name of a taxonomy element |
| keyValue | Concept.getValue | This is the value of identifying a taxonomy element |

2461

2462 **D.7.2 identifierBag**

2463 identifierBag is a collection of keyedReferences. An identifierBag is modeled in 2464 the JAXR information model as a Collection of ExternalIdentifiers.

2465 **D.7.3 categoryBag**

2466 categoryBag is a collection of keyedReferences. A categoryBag is modeled in 2467 the JAXR information model as a Collection of Classifications.

2468 **D.7.4 tModelBag**

2469 tModelBag is a collection of tModel *uuid\_key* values that represents the technical 2470 *fingerprint* of a bindingTemplate structure contained within the businessService

2471 specified by the serviceKey value.

2472 A tModelBag is modeled in the JAXR information model as a Collection of

2473 Concepts that represent technical fingerprint Concepts that serve as proxies for

2474 technical specification is a ServiceBinding (bindingTemplate).

2475 **D.8 Mapping of UDDI phone Element**

2476 UDDI allows a single String for the entire phoneNumber and an optional 2477 useType. The JAXR TelephoneNumber class provides a more structured 2478 representation of the phone number. Therefore, for JAXR UDDI providers the 2479 only relevant attributes are phoneType and number.

Sun Microsystems Page 107

JavaTM API for XML Registries April 10, 2002

2480 **D.8.1 Mapping of phone During Save Operations**

2481 A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client

2482 programmer tries to call any of the following operations:

2483 o setAreaCode

2484 o setCountryCode

2485 o setExtension

2486 o setURL

2487 The client programmer is expected to mainly use the setNumber method to set 2488 the complete telephone number as unstructured free-form text.

2489 The client programmer may also set a type on the TelephoneNumber using

2490 setType method. In this case, the type specified should be used in the useType

2491 attribute for the UDDI phone element.

2492 **D.8.2 Mapping of phone During Find Operations**

2493 A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client

2494 programmer tries to call any of the following operations:

2495 o getAreaCode

2496 o getCountryCode

2497 o getExtension

2498 o getURL

2499 A JAXR UDDI provider must map the CDATA of the phone element to the 2500 number attribute of telephoneNumber.

2501 If a useType is present for the phone element, then a JAXR UDDI provider must 2502 map the useType to the type attribute of TelephoneNumber.

2503 The client programmer is expected to mainly use the getNumber method to get 2504 the complete telephone number as unstructured free-form text.

2505 If the client programmer calls the getType method they should either get a type

2506 String or they should get null.

2507 **D.9 Mapping of name to PersonName**

2508 UDDI personName element allows a single String for the entire name of a

2509 person. The JAXR PersonName class provides a more structured representation

2510 of the a person’s name. Therefore, for JAXR UDDI providers the only relevant

2511 attributes is fullName.

Sun Microsystems Page 108

JavaTM API for XML Registries April 10, 2002

2512 A JAXR UDDI provider must throw an UnsupportedCapabilityException if a client 2513 programmer tries to call any method of the PersonName class other than the

2514 following operations:

2515 o getFullName

2516 o setFullName

2517

2518 **D.10 Example of JAXR-UDDI Mapping**

2519 Figure 32 below shows a simplified example described in terms of UDDI data 2520 structures. Figure 33 then shows the same example in terms of JAXR information

2521 model using the mapping described above.

2522 In this example a UDDI businessEntity is classified by an external classification 2523 using the taxonomy element with name "*Automobile and Light Duty Motor* 2524 *Vehicle Manufacturing*" and value 33611 in the NAICS taxonomy. It is also 2525 identified using a DUNS number of 45232 using the DUNS identification scheme. 2526 The businessEntity has a single businessService for a purchasing service that 2527 has a single bindingTemplate that has a single specification document that is a 2528 WSDL file.

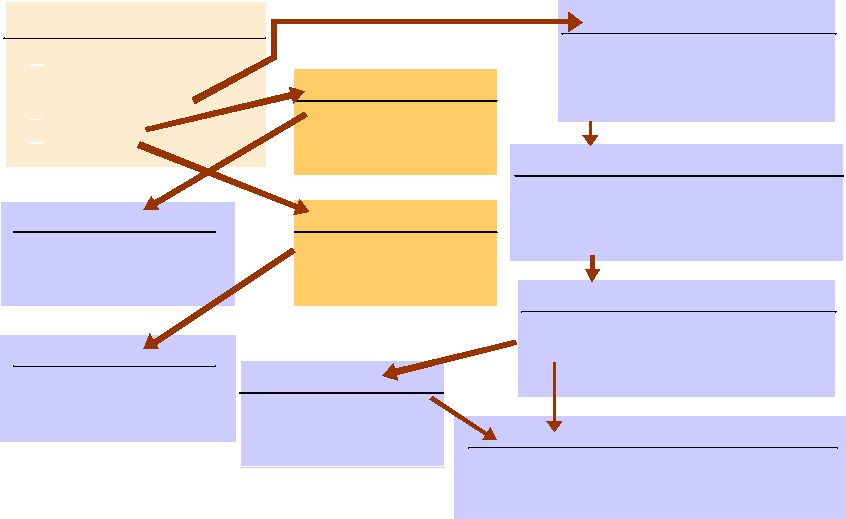
2529

Sun Microsystems Page 109

JavaTM API for XML Registries April 10, 2002

2529

**TmodelKey=**C0B9F... **keyName=**"Transportat.." **keyValue=**33611



**TmodelKey=** C0B9F... **name=**"ntis-gov:naics:1997" **description=**"NAICS...."

**TmodelKey=**8609C81E-... **name=**"dnb-com:D-U-N-S" **description=**"DUNS...."

**categoryBag keyValue=**45232 **BindingTemplate**

**businessKey** = TB993... **name** = "Fjord Auto Inc." **businessServices identifierBag**

**BusinessEntity**

**Tmodel**

**Tmodel**

**Tmodel**

**TmodelKey=**4453D6...

**name=**"myWSDLFile"

**description=**"...."

**TmodelKey=**DFE-2B... **keyName=**"JustJava"

**keyedReference**

**keyedReference**

**description=**"Fjord Autos purchasing service " **overviewURL=**"<http://www.fjordauto/purchasing>"

**Key=**5E2D412E5-44EE-...

**accessPoint=**"[http://www.fjordauto.com](http://www.fjordauto.com/)"

**tModelInstanceDetails**

**TmodelInstanceInfo**

**TmodelKey=**4453D6FC-223C-3ED0... **tModelInstanceDetails**

**ServiceKey=**23T701e54683nf... **name=**"Purchasing service" **BindingTemplates**

**OverviewDoc**

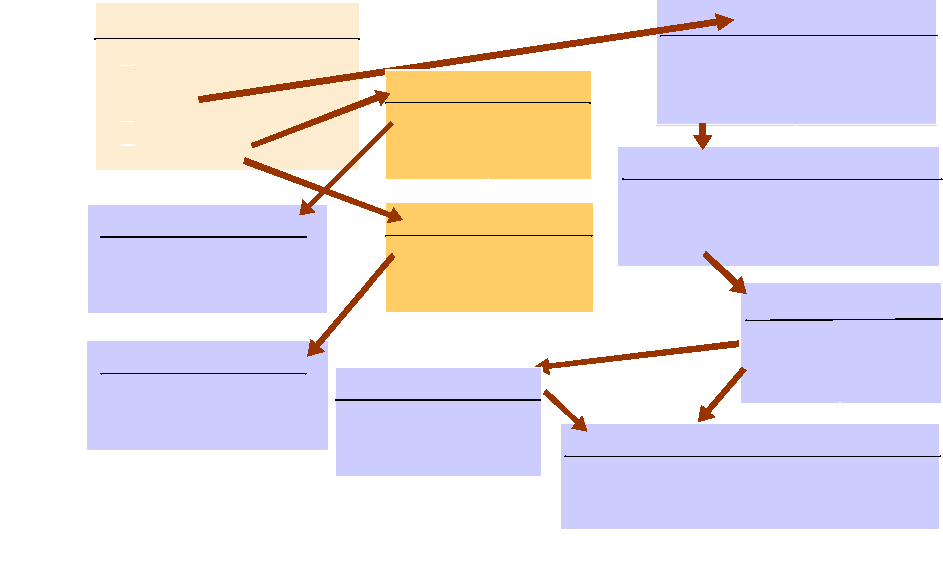
**BusinessService**

2530 Figure 32: Example in terms of UDDI Data Structures

Sun Microsystems Page 110

**cla..scheme=**C0B9F... **name=**"Transportat.." **value=**33611

2531



2532

2533 **Figure 33: UDDI Example Mapped to JAXR**

**ClassificationScheme**

[**key.id**](http://key.id/)**=**C0B9F...

**name=**"ntis-gov:naics:1997" **description=**"NAICS...."

**ClassificationScheme**

[**key.id**](http://key.id/)**=**8609C81E-... **name=**"dnb-com:D-U-N-S" **description=**"DUNS...."

[**key.id**](http://key.id/)= TB993... **name** = "Fjord Auto Inc." **services externalIdentifiers classifications**

**Organization**

[**key.id**](http://key.id/)**=**4453D6... **name=**"myWSDLFile" **description=**"...."

**id..Scheme=**DFE-2B... **name=**"JustJava" **value=**45232

**ExternalIdentifier**

**Concept**

**Classification**

**description=**"JustJava's purchasing service" **externalURI=**" <http://www.fjordauto.com/purchasing>"

**ServiceBinding**

[**key.id**](http://key.id/)**=**5E2D412E5-44EE-...

**accessURI=**"[http://www.fjordauto.com](http://www.fjordauto.com/).." **specificationLink**

**ExternalLink**

[**key.id**](http://key.id/)**=**23T701e54683nf... **name=**"Purchasing service" **serviceBindings**

**SpecificationLink**

**registryObject usageDescription usageParameters**

**Service**

JavaTM API for XML Registries April 10, 2002

2534 **D.11 Provider Generated id Attributes**

2535 Due to differences between the information models of JAXR and UDDI, there are 2536 several cases where a JAXR interface must have an id while its counterpart in 2537 UDDI does not have an id defined.

2538 In such cases, the JAXR provider must deal with this impedance mismatch 2539 transparent to the user, and generate id values in a deterministic manner.

2540 This following table itemizes each case where id needs to be generated and 2541 suggests a nor-normative algorithm that may be used to generate ids for each 2542 such case. Note that text is wrapped due to shortage of horizontal space in 2543 columns.

2544

|  |  |  |
| --- | --- | --- |
| **JAXR Interface** | **Algorithm** | **Example** |
| ExternalIdentifier | <RegistryObjectId>: | a2345678-1234-1234- |
|  | <identificationSchemeId>: | 123456789012:a2345678- |
|  |  | 1234-1234- |
|  | <value> | 123456789013:Social Security |
|  |  | Number |
| Association | <sourceObjectId>: | a2345678-1234-1234- |
|  | <targetObjectId>: | 123456789012:a2345678-1234-1234- |
|  | <associationType> | 975123456789013:Supersedes |
| Classification | <classifiedObjectId>: | a2345678-1234-1234- |
| (internal) | <classificationNodeId> | 123456789012:a2345678- |
|  |  | 1234-1234-123456789013 |
| Classification | <classifiedObjectId>: | a2345678-1234-1234- |
| (external) | <classificationSchemeId>: | 123456789012:a2345678- |
|  |  | 1234-1234-123456789013:61 |
|  | <nodeRepresentation> |  |
| ExternalLink | <externalURI>:<sequenceId> | [http://www.sun.com:1](http://www.sun.com:1/) |
| SpecificationLink | <serviceid>:<accessURI>: | a2345678-1234-1234- |
|  | <targetBindingId>: | 123456789012: |
|  | <sequenceId>: | [http://www.sun.com:](http://www.sun.com:/):1: |
|  |
|  | <specificationObjectId> | a2345678-1234-1234-  123456789013 |

2545

Sun Microsystems Page 111

JavaTM API for XML Registries April 10, 2002

2546 **D.12 Supporting Taxonomy Service In JAXR UDDI Providers**

2547 **D.12.1 Normative Description**

2548 This section provides a normative description of all required features when 2549 implementing a taxonomy service within a JAXR UDDI provider:

2550 o A JAXR UDDI provider must support the NAICS, UNSPSC and ISO 3166

2551 Geography taxonomies as internal taxonomies available within the

2552 implementation specific taxonomy service.

2553 o A JAXR UDDI provider must provide a provider-specific way for users to

2554 configure and manage arbitrary taxonomies within the taxonomy server.

2555 **D.12.2 Non-normative Description**

2556 This section provides a non-normative description of how a JAXR UDDI provider 2557 may support a taxonomy service, thus allowing entire taxonomies complete with 2558 their taxonomy structure to be available as internal ClassificationSchemes.

2559 A taxonomy service may be implemented in several forms. Below are some 2560 examples:

2561 o The JAXR provider may read one or more client side configuration files

2562 upon startup that contain taxonomy information.

2563 o The JAXR provider may have a server-side component that provides the

2564 taxonomy service function.

2565 o The JAXR provider may allow a level 1 registry to be configured as the

2566 taxonomy server.

2567 Regardless of the implementation choice for a taxonomy service internal to the 2568 JAXR UDDI provider, the following guidelines apply:

2569 o Taxonomy information in the taxonomy server is updated or deleted via

2570 out-of-band means not described the JAXR specification.

2571 o The JAXR UDDI provider never updates or deletes taxonomy information

2572 in the taxonomy server based upon a client call to a JAXR API method.

2573 Note that updates to the taxonomy server are done through out-of-band

2574 provider specific means.

2575 o The saveClassificationSchemes and

2576 deleteClassificationSchemes calls in the JAXR API only affect the

2577 UDDI registry and not the taxonomy server.

Sun Microsystems Page 112

JavaTM API for XML Registries April 10, 2002

2578 o The JAXR UDDI provider must query both the taxonomy server and the

2579 UDDI registry during findClassificationSchemes operations. It

2580 should combine the results of querying the taxonomy server and the

2581 UDDI registry and present a unified result to the JAXR client.

2582 o When combining results of the findClassificationSchemes

2583 operation, the provider must cull duplicates. In case of duplicates, the

2584 taxonomy server version should be kept since it has taxonomy structure.

2585

2586 **D.13 UDDI Functionality Not Supported By JAXR**

2587 The following table declares all UDDI functionality that is not accessible via JAXR 2588 API. Any potential omissions from this list are specification errors and should be

2589 reported.

2590

**UDDI Feature**

**Disposition**

**Description**

BusinessEntityExt

functionality

This functionality is not suitable for abstraction in JAXR. No plans to provide this in JAXR.

Deliberately not supported. Use makeRegistrySpecificRequest backdoor method in RegistryService interface.

2591

Sun Microsystems Page 113

JavaTM API for XML Registries April 10, 2002

2591 **Appendix E Value-Added Features of the JAXR API**

2592 This section described some features that are available to JAXR clients that

2593 provide unique value beyond the capabilities provided by underlying registries.

2594 **E.1 Taxonomy Browsing**

2595 The JAXR API allows a JAXR client of a UDDI registry to be able to browse the 2596 full structure of taxonomies or classification schemes. This is a unique capability, 2597 which is not available to typical (non-JAXR) UDDI clients. Even though the UDDI 2598 registry does not provide clients the ability to browse taxonomy structure, a JAXR 2599 provider for UDDI enables this useful feature. A JAXR provider comes pre-2600 configured with standard taxonomies such as NAICS, UNSPSC and ISO 3166 2601 Geography. It can also be extended to include any other user-defined taxonomy.

2602 **E.2 Taxonomy Validation**

2603 The JAXR API automatically validates all taxonomy values when a JAXR client 2604 creates an internal Classification using an internal taxonomy. This prevents the

2605 JAXR client from creating invalid classifications.

2606 **E.3 Smart Queries**

2607 The JAXR API enables smart queries that take advantage of the knowledge of 2608 taxonomy structures within internal classifications. This e nables a JAXR client to 2609 search for an Organization classified by Asia and all sub-Concepts of Asia. This

2610 enables clients to find Organizations that are directly or indirectly classified by the 2611 Asia Concept. Client may use the getDescendantConcepts method of the 2612 Concept interface to get all the descendents of a Concept and use them in the 2613 findOrganization query.

2614 **E.4 Enhanced Data Integrity and Validation**

2615 The JAXR API validates all URL links submitted as part of ExternalLink objects at

2616 the time of submission. The JAXR provider must ping the URL and throw an

2617 Exception if the URL is not valid and accessible. Joint research by SalCentral

2618 and WebServicesArchitect (see

2619 [http://www.webservicesarchitect.com/content/articles/clark04.asp](http://www.webservicesarchitect.com/content/articles/clark04.asp))) showed that

2620 nearly 48% of all URL links in UDDI are invalid.

2621 By validating all URLs at the time of submission, the JAXR API enhances data

2622 integrity for data submitted to UDDI and other registries.

Sun Microsystems Page 114

JavaTM API for XML Registries April 10, 2002

2623 **E.5 Automatic Categorization of UDDI tModels**

2624 In UDDI, a tModel is an overloaded concept with several different usages. For 2625 this reason the UDDI specifications suggest that all tModels be categorized 2626 according their type of usage. The JAXR information model has a separate 2627 interface for each of the unique uses of a UDDI tModel. A JAXR provder for 2628 UDDI may automatically categorize these tModels in many cases.

2629 **E.6 Simplified Programming Model**

2630 There are several areas where JAXR provides a simpler programming model 2631 than compared to raw interface defined by the underlying registry. Some 2632 examples follow:

2633 **E.6.1.1 Unification of find and get Methods**

2634 [UDDI-API2] defines two sets of methods. One is a set of *find* methods and the 2635 other is a set of *get* methods. The *find* methods perform a query for Objects in 2636 UDDI and return their identifier. The client must then use *get* methods to

2637 separately retrieve the details of specific objects. The JAXR API is simpler and 2638 only provides the *find* methods. UDDI *get* methods are called transparently within 2639 the provider if the JAXR client attempts to access detailed information about an 2640 object. This lazy fetching of objects from UDDI enables the JAXR API to present 2641 a simplified programming model to the JAXR client programmer.

2642 **E.6.1.2 Generic Handling of Object**

2643 Using the Object-Oriented principle of polymorphism, the JAXR API provides 2644 several methods that allow object operations without knowing the type of the 2645 object. For example one can call QueryManager.saveObjects instead of 2646 more specific save methods in the BusinessQueryManager and in a single 2647 operation save many different types of objects.

2648 **E.7 Simplified User Authentication**

2649 The JAXR API allows the user to set their Credentials on the JAXR Connection 2650 and from that point on user authentication with the target registry is completely 2651 hidden from the user. In fact the JAXR API frees the client programmer from

2652 knowing which API calls requires authentication with the target registry. The 2653 JAXR provider is smart enough to know on its own that it must authenticate with 2654 a UDDI registry for *save* and *delete* operations but not for *find* and *get*

2655 operations.

Sun Microsystems Page 115

JavaTM API for XML Registries April 10, 2002

2656 **E.8 Enforce No New References to Deprecated Objects**

2657 The JAXR API makes sure that no new references are allowed to be created to 2658 deprecated objects by JAXR client. This is a value-added feature for level 1 2659 registries such as the ebXML Registry.

2660 **Appendix F Frequently Asked Questions**

2661 Question: Why do we need a new JAXR API when we have the JNDI API?

2662 Answer: The JNDI API was designed with a very different set of requirements 2663 than JAXR. Both are abstraction APIs over existing specifications. However, the 2664 abstraction in directory services differ quite a bit from those of XML Registries 2665 used for publishing and discovery of web services. JAXR needs richer metadata

2666 capabilities for classification and association, as well as richer query capabilities.

2667

2668 Question: Would not be better to have enhanced the JNDI API with the added 2669 functionality of JAXR?

2670 Answer: That option was considered. Meeting the additional requirements of 2671 XML Registries requires an elaborate information model. The JNDI API already 2672 has an existing information model that is constrained by design to address the 2673 requirements for directory services. Extending the JNDI API would overly 2674 constrain JAXR and would create backward compatibility issues for the JNDI 2675 API.

2676

2677 Question: Why is JAXR an abstraction API and not targeted to a specific registry 2678 such as UDDI or ebXML?

2679 Answer: An abstraction based JAXR API provides developers the ability to write

2680 registry client programs that portable across different target registries. This is 2681 consistent with the Java philosophy of “Write Once Run Anywhere (WORA)”. It 2682 also enables value-added capabilities as described in Appendix E. These

2683 capabilities are above-and-beyond the capabilities of underlying registries. For 2684 example, a non-JAXR UDDI client does not have the ability to do taxonomy 2685 browsing, and taxonomy aware smart queries, which are available to a JAXR

2686 client for UDDI.

2687

2688 Question: Why does the JAXR API not use UDDI terms and concepts?

2689 Answer: The JAXR API is not specific to UDDI or any other registry specification.

2690 It is an abstraction API that covers multiple specifications. It is designed to

2691 enable developer choice in use of a web service registry and/or repository.

Sun Microsystems Page 116

JavaTM API for XML Registries April 10, 2002

2692 The JAXR API uses UDDI terms and concepts when they fit the JAXR 2693 information model (e.g. Service, ServiceBinding, and method names in 2694 BusinessQueryManager and BusinessLifeCycleManager)

2695

2696 Question: Why did the JAXR information model use the ebXML Registry 2697 Information Model as its basis rather than the UDDI data structures?

2698 Answer: The JAXR API is designed to support multiple registries. The ebXML 2699 Registry Information Model is more generic and extensible than the UDDI data 2700 structures. Because of this characteristic, it was possible to extend the ebXML

2701 Registry Information Model to satisfy the needs of UDDI and other registries. 2702

2703 Question: Why was the JAXR information model not designed from the ground 2704 up?

2705 Answer: Information models take time to develop. It was easier to study an 2706 existing information model and improve upon it.

2707

Sun Microsystems Page 117

JavaTM API for XML Registries April 10, 2002

2707 **11 References**

2708 [JAXRF] JAXR developer forum: <http://groups.yahoo.com/group/jaxr-discussion> 2709 [ISO] ISO 11179 Information Model

2710 <http://208.226.167.205/SC32/jtc1sc32.nsf/576871ad2f11bba78525662100>

2711 5419d7/b83fc7816a6064c68525690e0065f913?OpenDocument

2712 [ebXML-RSS] ebXML Registry Services Specification

2713 <http://www.ebxml.org/project_teams/registry/private/registryServicesSpecificationv1.0.pdf> 2714 [ebXML-RIM] ebXML Registry Information Model

2715 <http://www.ebxml.org/project_teams/registry/private/registryInfoModelv1.0.pdf> 2716 [UDDI-DS] UDDI Version 1.0 Data Structure Specification

2717 <http://www.uddi.org/pubs/DataStructure-V1.pdf>

2718 [UDDI-DS2] UDDI Version 2.0 Data Structure Reference

2719 <http://www.uddi.org/pubs/DataStructure-V2.00-Open-20010608.pdf> 2720 [UDDI-API] UDDI Version 1.0 Programmers API

2721 <http://www.uddi.org/pubs/ProgrammersAPI-V1-1.pdf>

2722 [UDDI-API2] UDDI Version 2.0 API Specification

2723 <http://www.uddi.org/pubs/ProgrammersAPI-V2.00-Open-20010608.pdf>

2724 [SQL] Structured Query Language (FIPS PUB 127-2)

2725 <http://www.itl.nist.gov/fipspubs/fip127-2.htm>

2726 [SQL/PSM] Database Language SQL — Part 4: Persistent Sto red Modules

2727 (SQL/PSM) [ISO/IEC 9075-4:1996]

2728

2729 IANA] IANA (Internet Assigned Numbers Authority).

2730 Official Names for Character Sets, ed. Keld Simonsen et al.

2731 [ftp://ftp.isi.edu/in-notes/iana/assignments/character-sets](ftps://ftp.isi.edu/in-notes/iana/assignments/character-sets)

2732

2733 [REC-XML] W3C Recommendation. Extensible Markup language 2734 (XML)1.0(Second Edition)

2735 <http://www.w3.org/TR/REC-xml>

2736 [UUID] DCE 128 bit Universal Unique Identifier

2737 [http://www.opengroup.org/onlinepubs/009629399/apdxa.htm#tagcjh\_20](http://www.opengroup.org/onlinepubs/009629399/apdxa.htm" \l "tagcjh_20)

2738 <http://www.opengroup.org/publications/catalog/c706.htm>

2739 [Futures1] Mohr, E.; Kranz, D; Halstead, R.: Lazy Task Creation: A Technique

Sun Microsystems Page 118

JavaTM API for XML Registries April 10, 2002

2740 for Increasing the Granularity of Parallel Programs. IEEE

2741 Transactions on Parallel and Distributed Systems, 1990.

2742 <http://www4.informatik.uni-erlangen.de/~tsthiel/Papers/alewife-lazy-task-creation.ps.gz>

2743 [Futures2] <http://www.cs.wisc.edu/~fischer/cs538.s01/multilisp.pdf> 2744

Sun Microsystems Page 119