Table of Contents

Read This First 1
1. BEA AquaLogic Service Registry Features Overview 1
2. Release Notes 2
2.1. What's New 2
2.2. Known Issues 3
2.2.1. UDDI Version 3 Specification 3
2.2.2. UDDI Version 2 Specification 3
2.2.3. Database 3
2.2.4. Consoles 4
2.2.5. Other 4
2.3. Change Log 5
2.3.1. BEA AquaLogic Service Registry 2.1 5
2.3.2. WASP UDDI 4.6 6
2.3.3. WASP UDDI 4.5.2 7
2.3.4. WASP UDDI 4.5.1 7
2.3.5. WASP UDDI 4.5 8
2.3.6. WASP UDDI 4.0 9
3. Supported Platforms 9
4. Specifications 10
5. Document Conventions 10
6. Legal 12
6.1. Third Party Licenses 12
6.1.1. HSQLDB License 12
6.1.2. PostgreSQL 7.4 13
6.1.3. The Apache XML License, Version 1.1 13
6.1.4. Apache Jakarta License, Version 1.1 14
6.1.5. CUP Parser Generator 15
6.1.6. Jetty License, Version 3.6 15
6.1.7. W3C Software Notice and License 17
6.1.8. Xalan, Version 2.5.1 17
6.1.9. XML Pull Parser for Java, 1.1.1 18
6.2. Copyright 19
6.3. Restricted Rights Legend 19
6.4. Trademarks or Service Marks 19
6.5. Acknowledgements 20
7. Support 20
7.1. Contact 20
User's Guide 21
1. Introduction to BEA AquaLogic Service Registry 21
1.1. UDDI's Role in the Web Services World - UDDI Benefits 22
1.2. Typical Application of a UDDI Registry 22
1.3. Basic Concepts of the UDDI Specification 22
1.3.1. UDDI Data Model 23
1.3.2. Taxonomic Classifications 24
1.3.3. Security Considerations 25
1.3.4. Notification and Subscription 25
1.3.5. Replication 25
1.3.6. UDDI APIs 25
1.3.7. Technical Notes 26
1.3.8. Benefits of UDDI Version 3 26
1.4. Subscriptions in BEA AquaLogic Service Registry 26

1.4.1. Subscription Arguments ... 26
1.4.2. Subscription Notification ... 27
1.4.3. XSLT Over Notification ... 27
1.4.4. Suppressing Empty Notifications ... 27
1.4.5. Related Links ... 27
1.5. Approval Process in BEA AquaLogic Service Registry ... 28
1.5.1. Requestor's Actions ... 29
1.5.2. Approver's Actions ... 31
1.5.3. Synchronization of Data ... 31
1.5.4. Related Links ... 32
2. Registry Consoles ... 33
3. Demo Data ... 33
3.1. Demo Data for Business Service Console ... 33
3.2. Demo data for Registry Console and demos ... 34
4. Business Service Console ... 35
4.1. Overview ... 36
4.2. User Account ... 38
4.2.1. User Profile Fields ... 40
4.2.2. Predefined User Profiles ... 41
4.3. Searching ... 42
4.3.1. Searching Providers ... 42
4.3.2. Searching Endpoints ... 44
4.4. Publishing ... 46
4.4.1. Publishing Providers ... 46
4.4.2. Publishing Services ... 48
4.5. Reports ... 52
4.6. Entities ... 53
4.6.1. Entity Details ... 53
4.6.2. Resources ... 54
4.7. Subscription and Notification ... 55
4.7.1. Subscription On Selected Entities ... 55
4.7.2. Subscription from Search Query ... 57
4.7.3. Manage Subscriptions ... 59
4.7.4. View Changed Entities ... 59
4.8. Approval Process ... 60
4.8.1. Requestor's Actions ... 60
4.8.2. Approver's Actions ... 66
5. Advanced Topics ... 68
5.1. Data Access Control: Principles ... 68
5.1.1. Explicit Permissions ... 69
5.1.2. Permission Rules ... 69
5.1.3. Composite Operations ... 70
5.1.4. Pre-installed Groups ... 70
5.1.5. ACL tModels ... 70
5.1.6. Setting ACLs on UDDI v3 Structures ... 71
5.1.7. Setting ACLs on UDDI v1/v2 Structures ... 71
5.2. Publisher-Assigned Keys ... 72
5.2.1. Generating Keys ... 72
5.2.2. Affiliations of Registries ... 73
5.3. Range Queries ... 74
5.3.1. Examples ... 74
5.4. Taxonomy: Principles, Creation and Validation ... 76
5.4.1. What Is a Taxonomy? ... 76
5.4.2. Taxonomy Types ... 76

5.4.3. Validation of Values ... 76
5.4.4. Types of keyValues ... 77
5.4.5. Taxonomy API ... 80
5.4.6. Predeployed Taxonomies ... 82
5.5. Registry Console Reference ... 91
5.5.1. Register/Create Account ... 91
5.5.2. Registry Console Overview ... 93
5.5.3. User Profile ... 95
5.5.4. Browsing ... 99
5.5.5. Searching ... 102
5.5.6. Publishing ... 111
5.6. Signer Tool ... 135
5.6.1. Starting the Signer ... 135
5.6.2. Main Screen ... 136
5.6.3. Sign ... 136
5.6.4. Validation ... 138
5.6.5. Remove Signatures ... 138
5.6.6. Publish Changes ... 138
5.6.7. Signer Configuration ... 138
Administrator's Guide ... 139
1. Registry Management ... 140
1.1. Accessing Registry Management ... 140
1.2. Account Management ... 141
1.2.1. Create Account ... 142
1.2.2. Edit Account ... 144
1.2.3. Delete Account ... 145
1.3. Group Management ... 145
1.3.1. Create and Manage Groups ... 146
1.3.2. Manage Group Membership ... 147
1.4. Permissions ... 147
1.4.1. Accessing Permission Management ... 148
1.4.2. Add Permission ... 148
1.4.3. Editing and Deleting Permissions ... 149
1.4.4. Assigning Administrator's Permission ... 150
1.5. Taxonomy Management ... 150
1.5.1. Adding Taxonomies ... 152
1.5.2. Finding Taxonomies ... 154
1.5.3. Editing Taxonomies ... 155
1.5.4. Editing a Taxonomy Structure ... 156
1.5.5. Uploading Taxonomies ... 160
1.5.6. Downloading Taxonomies ... 160
1.5.7. Deleting Taxonomies ... 160
1.6. Replication Management ... 160
1.6.1. Master Registry Setup ... 161
1.6.2. Slave Registry Setup ... 162
1.7. Approval Process Management ... 165
1.7.1. Loading the Approval Management Page ... 165
1.7.2. Create Approver ... 165
1.7.3. Create Requestor ... 166
1.8. Replacing UDDI Keys ... 167
1.8.1. Replacing tModel keys ... 167
1.8.2. Replacing businessEntity keys ... 168
1.8.3. Replacing businessService keys ... 168
1.8.4. Replacing bindingTemplate keys ... 168

1.9. Registry Statistics ... 168
2. Registry Configuration ... 171
2.1. Core Config ... 172
2.2. Database ... 172
2.3. Security ... 174
2.4. Account ... 175
2.5. Group ... 176
2.6. Subscription ... 176
2.7. Node ... 177
3. Business Service Console Configuration ... 178
3.1. Tabs Displayed ... 178
3.2. Search Result View ... 179
3.3. Browsable Taxonomies ... 180
3.4. Paging Limits ... 181
3.5. UI Configuration ... 181
3.6. Customizable Taxonomies ... 182
3.7. Customizing Individual Pages ... 185
4. Registry Console Configuration ... 188
4.1. Web Interface Configuration ... 189
4.2. Paging Configuration ... 190
5. Permissions: Principles ... 190
5.1. Permissions Definitions ... 191
5.2. BEA AquaLogic Service Registry Permission Rules ... 191
5.3. Setting Permissions ... 192
5.4. Permissions and User Roles ... 193
5.5. ApiManagerPermission Reference ... 193
6. Approval Process Principles ... 199
6.1. Approval Process Roles ... 200
6.1.1. Requestor ... 200
6.1.2. Approver ... 200
6.1.3. autoApprover ... 200
6.1.4. Administrator ... 201
6.2. Optional Content Checking Setup ... 201
7. PStore Tool ... 202
7.1. Commands Description ... 202
7.2. PStore Tool - GUI Version ... 203
7.2.1. Running the GUI PStore Tool ... 204
7.2.2. Opening and Closing the Protected Store ... 204
7.2.3. Open Next Protected Store ... 205
7.2.4. Copy Data Between Protected Stores ... 205
7.2.5. Key Store ... 205
7.2.6. User Store ... 207
Developer's Guide ... 209
1. Mapping of Resources ... 209
1.1. WSDL ... 209
1.1.1. WSDL PortTypes ... 210
1.1.2. WSDL Bindings ... 210
1.1.3. WSDL Service ... 211
1.1.4. Use Cases ... 211
1.2. XML ... 212
1.2.1. Use Cases ... 212
1.3. XSD ... 213
1.3.1. Use Cases ... 214
1.4. XSLT ... 214

1.4.1. Use Cases ... 215
2. Client-Side Development ... 216
2.1. UDDI APIs ... 216
2.1.1. Principles To Use UDDI API ... 216
2.1.2. UDDI Version 1 ... 222
2.1.3. UDDI Version 2 ... 223
2.1.4. UDDI Version 3 ... 223
2.1.5. UDDI Version 3 Extension ... 224
2.2. Advanced APIs ... 230
2.2.1. Validation ... 230
2.2.2. Taxonomy ... 231
2.2.3. Category ... 240
2.2.4. Approval ... 246
2.2.5. Administration Utilities ... 272
2.2.6. Replication ... 276
2.2.7. Statistics ... 277
2.2.8. WSDL Publishing ... 280
2.2.9. XML Publishing ... 290
2.2.10. XSD Publishing ... 297
2.2.11. XSLT Publishing ... 306
2.2.12. Inquiry UI ... 317
2.2.13. Subscription Ext ... 323
2.3. Security APIs ... 324
2.3.1. Account ... 324
2.3.2. Group ... 330
2.3.3. Permission ... 336
2.4. Registry Client ... 339
2.4.1. Client Package ... 340
2.4.2. JARs on the Client Classpath ... 341
2.5. Client Authentication ... 347
2.5.1. Sample Files ... 350
3. Server-Side Development ... 352
3.1. Accessing Backend APIs ... 352
3.2. Custom Registry Modules ... 356
3.2.1. Accessing Registry APIs ... 357
3.2.2. Custom Module Sample ... 358
3.3. Interceptors ... 360
3.3.1. Creating and Deploying Interceptors ... 360
3.3.2. Logging Interceptor Sample ... 361
3.3.3. Request Counter Interceptor Sample ... 363
3.4. Writing a Custom Validation Service ... 366
3.4.1. Deploying Validation Service ... 366
3.4.2. External Validation Service ... 367
3.4.3. Sample Files ... 369
3.5. Writing a Subscription Notification Service ... 369
3.5.1. Sample Files ... 371
3.6. Writing a Content Checker ... 372
3.7. Systinet Web Framework ... 375
3.7.1. Architecture Description ... 375
3.7.2. Directory Structure ... 380
3.7.3. Framework Configuration ... 381
3.7.4. syswf JSP tag library ... 383
3.7.5. Typical Customization Tasks ... 388
3.8. Business Service Console Framework ... 389

3.8.1. Business Service Console Localization ... 389
3.8.2. Directory Structure ... 392
3.8.3. Business Service Console Configuration ... 394
3.8.4. Entity Configuration ... 398
3.8.5. Permission support ... 412
3.8.6. Components and Tags ... 413
4. UDDI from Developer Tools ... 451
4.1. UDDI from Systinet Developer for Eclipse ... 452
4.1.1. Getting Data from UDDI ... 452
4.1.2. Publishing WSDL to UDDI ... 454
4.2. UDDI from MS Visual Studio ... 455
5. How to Debug ... 457
5.1. SOAPSpy Tool ... 457
5.1.1. Running SOAPSpy ... 457
5.1.2. Using SOAPSpy ... 458
5.1.3. SOAP Request Tab ... 458
5.1.4. How to Run Clients Using SOAPSpy ... 459
5.2. Logging ... 459
Demos ... 461
1. Basic Demos ... 461
1.1. UDDI v1 ... 461
1.1.1. Inquiry v1 ... 461
1.1.2. Publishing v1 ... 466
1.2. UDDI v2 ... 471
1.2.1. Inquiry v2 ... 471
1.2.2. Publishing v2 ... 475
1.3. UDDI v3 ... 481
1.3.1. Inquiry v3 ... 481
1.3.2. Publishing v3 ... 486
2. Advanced Demos ... 492
2.1. Advanced Inquiry - Range Queries ... 492
2.1.1. Prerequisites and Preparatory Steps: Code ... 493
2.1.2. Presentation and Functional Presentation ... 493
2.1.3. Building and Running Demos ... 494
2.2. Custody ... 497
2.2.1. Prerequisites and Preparatory Steps: Code ... 497
2.2.2. Presentation and Functional Presentation ... 498
2.2.3. Building and Running Demos ... 500
2.3. Subscription ... 501
2.3.1. Prerequisites and Preparatory Steps: Code ... 502
2.3.2. Presentation and Functional Presentation ... 502
2.3.3. Building and Running Demos ... 504
2.4. Validation ... 508
2.4.1. Prerequisites and Preparatory Steps: Code ... 508
2.4.2. Presentation and Functional Presentation ... 509
2.4.3. Building and Running Demos ... 510
2.5. Taxonomy ... 511
2.5.1. Prerequisites and Preparatory Steps: Code ... 512
2.5.2. Presentation and Functional Presentation ... 513
2.5.3. Building and Running Demos ... 515
3. Security Demos ... 516
3.1. Account ... 517
3.1.1. Prerequisites and Preparatory Steps: Code ... 517
3.1.2. Presentation and Functional Presentation ... 518

3.1.3. Building and Running Demos 519
3.2. Group 521
3.2.1. Prerequisites and Preparatory Steps: Code 521
3.2.2. Presentation and Functional Presentation 522
3.2.3. Building and Running Demos 523
3.3. Permission 525
3.3.1. Prerequisites and Preparatory Steps: Code 525
3.3.2. Presentation and Functional Presentation 526
3.3.3. Building and Running Demos 528
3.4. ACL 529
3.4.1. Prerequisites and Preparatory Steps: Code 529
3.4.2. Presentation and Functional Presentation 530
3.4.3. Building and Running Demos 532
4. Resources Demos 534
4.1. WSDL2UDDI v2 534
4.1.1. Prerequisites and Preparatory Steps: Code 535
4.1.2. Presentation and Functional Presentation 535
4.1.3. Building and Running Demos 538
4.2. WSDL2UDDI v3 540
4.2.1. Prerequisites and Preparatory Steps: Code 540
4.2.2. Presentation and Functional Presentation 541
4.2.3. Building and Running Demos 543
4.3. XML2UDDI 545
4.3.1. Prerequisites and Preparatory Steps: Code 545
4.3.2. Presentation and Functional Presentation 546
4.3.3. Building and Running Demos 547
4.4. XSD2UDDI 548
4.4.1. Prerequisites and Preparatory Steps: Code 549
4.4.2. Presentation and Functional Presentation 549
4.4.3. Building and Running Demos 550
4.5. XSLT2UDDI 553
4.5.1. Prerequisites and Preparatory Steps: Code 553
4.5.2. Presentation and Functional Presentation 554
4.5.3. Building and Running Demos 555
Glossary 557

Read This First

Welcome to BEA AquaLogic Service Registry!

BEA AquaLogic Service Registry is the leading business service registry, providing discovery, publishing and approval of SOA business services. With full support for version 3 of the UDDI (Universal Description, Discovery and Integration) standard, BEA AquaLogic Service Registry is a key component of a Service Oriented Architecture (SOA).

This product documentation contains the following sections:

Read This First This book is recommended for all readers. It provides a product overview, release notes, product changes, the typographical conventions used throughout this guide.

User's Guide This book describes how to manually maintain BEA AquaLogic Service Registry contents. All basic functions of the Registry Console are discussed here.

Developer's Guide Introduces the basics of creating extensions and client programs in BEA AquaLogic Service Registry. The Developer's Guide also documents the BEA AquaLogic Service Registry demo suite.

Administrator's Guide Explains BEA AquaLogic Service Registry's configuration and management, and introduces the tools and utilities you will need to perform these tasks.

1. BEA AquaLogic Service Registry Features Overview

BEA AquaLogic Service Registry is the only fully V3-compliant implementation of UDDI (Universal Description, Discovery and Integration), and is a key component of a Service Oriented Architecture (SOA). BEA AquaLogic Service Registry is an easy-to-use, standards-based mechanism for publishing and discovering Web services and related resources like XML Schemas or XSLT transformations.

BEA AquaLogic Service Registry fully implements the OASIS UDDI V3 standard. BEA AquaLogic Service Registry can be deployed in almost any Java environment and works with all popular database systems. In addition, the registry has been designed specifically for enterprise deployment and includes many advanced features that make it easy to configure, deploy, manage and secure. BEA AquaLogic Service Registry is also easy to customize to support different enterprise user communities.

BEA AquaLogic Service Registry extends the core UDDI V3 standard with unique functionality designed for enterprise applications:

- **Advanced Security** allows for defining granular access control for registered components. Component publisher can specify find, get, modify and delete access permissions for every published object.
- **Data Accuracy & Quality enforcement** mechanisms ensure that component registrations are accurate and up-to-date. BEA AquaLogic Service Registry clearly defines responsibility for every registered component. It offers component promotion & approval mechanisms for promoting components between development, QA and production environments.
- **Subscription & Notification** for automatically notifying registry users about changes to components that they depend on.
- **Selective Replication** among multiple registries allow for automated propagation between different registries (for e.g. between internal and external registries).
- **Advanced Taxonomy Management** for enforcement of well-defined taxonomies.
- **Powerful Management** for granular control, logging and auditing of the publishing and discovery processes.

- **Performance & Scalability** UDDI provides maximum performance and scalability by efficient implementation of web services stack and database algorithms and by supporting of a load balancing and clustering mechanism.

BEA AquaLogic Service Registry is a platform-independent solution that can easy be deployed in a wide variety of settings. The registry can run either standalone or within an application server: Many application servers, such as BEA WebLogic, are supported. BEA AquaLogic Service Registry also offers unrivalled support for a broad set of database management systems for storing registrations (e.g. Oracle, MS SQL Server, Sybase, IBM DB/2, PostgreSQL and HSQL). Crucially, BEA AquaLogic Service Registry also integrates with both LDAP and Microsoft ActiveDirectory.

2. Release Notes

2.1. What's New

- Business Service Console:
  - The **Home** tab has been redesigned as a dashboard of the most frequently used features;
  - Context menus for Catalog tree - right click to display the set of operations allowed on the selected entity type;
  - The user interface now only displays links for actions that the user has permission to perform;
  - Quick search - the user can search all data structures by keyword;
  - The navigation panel on the left-hand side of the **Catalog** and **Reports** tabs can be hidden, with a mouse click or **Alt-Q**;
  - Duplicate scrollbars have been eliminated from the UI;
- Entities in the BSC:
  - When viewing entity details, a new **System Info** tab provides information about the owner, creation and modification dates and UDDI keys;
  - Custom Entity Types - an administrator can define a new entity type based on a UDDI entity type and a specific categorization. For example, a "Policy" can be a tModel (UDDI type) with a keyedReference to `uddi:schemas.xmlsoap.org:policytypes:2003_03` with "policy" as the keyValue. Custom types are added seamlessly to the Catalog tree and **Reports** tab;
  - References between entities - it is possible to create and browse references between entities. The user can view all references from the current entity to other entities and find all entities which refer to the current entity;
  - Configurable Searches - an administrator can configure the search dialog for an entity type by changing the appropriate categorization;
- Localization - the registry console and Business Service Console are prepared for localization to other languages;
- Publishng Services:
  - A user can publish a service from a WSDL document stored on a web server requiring HTTP Basic authentication;
  - The performance of WSDL to UDDI publishing has been improved;
- Approval Process:

- Wizards for submission of approval requests have been simplified and improved. A user can immediately submit an approval request when an entity is published;
- Errors are prevented by automatic closure checking. A list of entities that must be added to a request is automatically supplied;

- Server-Side Development:
  - Business Services Console Framework - enhancements to support customization and integration.

2.2. Known Issues

2.2.1. UDDI Version 3 Specification

The following parts of the UDDI Version 3 specification are not implemented:

- Inter-Node operation - this part of the specification is not implemented.
- Replication Specification - The Replication Specification describes the data replication process and the programming interface required to achieve complete replication between UDDI Operators in the UBR (Universal Business Registry ~ UDDI operator cloud). This part of the specification is mandatory for members of the UBR and is not implemented.
- Policy - The policy description is not defined.
- Exclusive XML Canonicalization [http://www.w3.org/2001/10/xml-exc-c14n#] is used for canonicalization of digital signatures. Schema-centric XML Canonicalization is not yet implemented.

2.2.2. UDDI Version 2 Specification

The following parts of the UDDI Version 2 specification are not implemented:

- Operator Specification - This part of the specification is mandatory for members of the UBR and is implemented with the exceptions described in this section.
- Custody transfer from version 2 is not implemented.
- Replication Specification - The Replication Specification describes the data replication process and the programming interface required to achieve complete replication between UDDI Operators in the UBR. This part of the specification is mandatory for members of the UBR and is not implemented.

2.2.3. Database

- Sybase ASE (Adaptive Server Enterprise) has a limit of 16 sub-selects for queries (`SELECT ... FROM ... WHERE EXISTS (SELECT...)`). Because of this limit, some more complex queries (such as find by category bag with more keyed references) do not work.
- There are the following caveats in data migration and backup:
  - Deletion history for subscriptions is not migrated and backed up.
  - Custody transfer requests are not migrated and backed up.
  - Migration and backup of approval requests and relationships between requestors and approvers are not yet implemented.

- We do not recommend installing BEA AquaLogic Service Registry with the HSQL database under IBM Java 1.4.x since the installation may time out.

2.2.4. Consoles

- The Firefox web browser interprets **Alt** key combinations in a non-standard way. One consequence of this is that use of **Alt**+**1**, **Alt**+**2** etc. to change tabs may change the Firefox tab instead of the Business Service Console tab.

- On completing an operation, the page displayed by the Business Service Console is not always accurately reflected in the state of the browser, including the current URL and POST data. Consequently, clicking the browser's refresh button may result in an erroneous attempt to repeat the operation. For an operation such as deleting a resource, this will result in error code `E_invalidKeyPassed` because the resource has already been deleted. To avoid this problem, use the refresh button provided by the Business Service Console instead.

- If the user's login expires because of a prolonged pause during execution of a wizard, he will be required to login before the wizard resumes. However, resumption of the wizard is not always reliable, resulting in subsequent errors. This is known to occur in the wizard that adds a reference to an entity (from that entity to another). See Section 4.6, Entities;

- If a browsable taxonomy is checked then any of its categories that contain items should appear in the reports tree, as described in Section 3.3, Browsable Taxonomies. However, when a category contains no items and an item is added, the reports tree is not immediately updated because it is cached. To ensure it is updated the user must take some action to clear the cache, such as closing and reopening their browser;

- It is possible for an administrator to configure an internal taxonomy (that has a fixed set of categories) represented using input mode on pages. See Section 3.6, Customizable Taxonomies. The user is then able to enter arbitrary text as the category and an error will occur if the value entered is not one of the defined categories;

- The `uddi-org:wsdl:categorization:transport` taxonomy appears on the **Search endpoints** page of the Business Service Console, in the **Binding properties** composite area with caption **Transport**. However, an administrator attempting to use **Customizable taxonomies** to edit this taxonony is initially told that it is not compatible with Endpoints. Subsequently they are given the opportunity to choose the area on the **Search endpoints** page where the taxonomy appears. This can confuse users. This taxonomy is not compatible with Endpoints but searching Endpoints by transport is implemented as a special case using find_tModel;

2.2.5. Other

- Use of `SubjectAlternativeName` in certificates is not yet supported. This has potential impact wherever SSL is used and the secure host has more than one hostname. See WSDL Publishing below. The result is a `java.net.ssl.SSLException` with a message that hostnames do not match.

- Installation fails if the installation path contains non-ASCII characters;

- Attempting to undeploy BEA AquaLogic Service Registry from an application server may appear to have been successful but can leave files locked until the application server and its JVM exit. This means than an attempt to redeploy BEA AquaLogic Service Registry to the application server will fail because these files exist and cannot be overwritten. A workaround is to restart the application server;

- Selective One-way Replication has the following caveats:

  - Checked taxonomies are replicated as unchecked. Taxonomy data replication and change of taxonomy to checked must be done manually.

  - Custody transfer requests are not replicated.

  - Publisher assertions are not replicated.

- Approval process has the following caveats:
  - Promotion of projected services is not supported.
  - Promotion of publisher assertions is not implemented yet.
- LDAP
  - Dynamic groups in LDAP account backends are not processed.
  - The approximateMatch find qualifier is not supported in LDAP account backends. There is no wildcard that can represent any single character in the directory (LDAP or AD). `%` is mapped to `*`, it is not possible to map `_`.
  - Groups from disabled domains are visible in the Registry Console.
- Intranet identity association is not implemented; the system#intranet group is reserved for future use.
- Password structure and length checking, expiration, checking of repeated failed logins and IP mask restriction are not implemented.
- The Signer tool does not support the refresh operation. If you start the Signer and then modify a UDDI structure, you must restart the Signer Tool.
- The Setup tool throws an exception when you try to configure registry ports on BEA AquaLogic Service Registry that are not connected to a database. The exception does not affect the port configuration.
- WSDL Publishing:
  - Unable to unpublish unreachable WSDLs in Registry Console.
  - Publishing a WSDL at a URL that has `https` as protocol may fail because the server certificate uses `SubjectAlternativeName` to specify alternative hostnames. This is not yet supported as noted above. The result may be a `WSDLException` with fault code `INVALID_WSDL` but the underlying cause is in fact a `java.net.ssl.SSLException` with a message that hostnames do not match.
- If you change the BEA AquaLogic Service Registry configuration using the Setup tool, demo data is always imported the registry database.

2.3. Change Log

2.3.1. BEA AquaLogic Service Registry 2.1

- Business Service Console - The functionality of the Business Service Console has been extended in the following areas:
  - Approval Process - The approval process has been implemented in the Business Service Console for requestors and approvers. Requestors can create and submit requests, manage their requests, and clone requests to the request work area. Requestors can also send reminders to their approvers. Approvers can approve/reject requests and view approval histories.
  - Subscriptions and Notifications - The Business Service Console allows you to create and manage subscriptions for monitoring new, changed, and deleted entities. The following entities can be monitored: providers, services, interfaces, and endpoints, as well as resources (WSDL, XML, XSD and XSLT).

  - User Profiles - BEA AquaLogic Service Registry contains a list of predefined user profiles which differ in which main menu tabs will be available to them. Each user profile also contains a definition of default formats for result views. The registry administrator can adjust these user profiles.
  - Reports are based on taxonomic classifications.
  - Paging and large results set support - The Business Service Console supports paging for displaying large result sets. The maximum number of pages and number or rows per page can be configured for each component.
  - Overall performance of the Business Service Console has been increased by Business Service Console framework optimization.

- Approval Process
  - Changed terminology from previous release - the *staging registry* has been renamed to *publication registry*; the *production registry* has been renamed to *discovery registry*.
  - New installation/configuration scenarios have been added. The approval process can be installed with multiple publication registries and the approval process can be performed in multiple steps.

- Backup functionality [http://edocs.bea.com/alsr/docs21/install/backup.html]- Backup functionality allows you to save the BEA AquaLogic Service Registry data and configuration to a filesystem directory. Later the backup data can serve for a full restore of BEA AquaLogic Service Registry data and configuration.
- Documentation
  - Introduction to BEA AquaLogic Service Registry
  - Accessing UDDI from Developer Tools

2.3.2. WASP UDDI 4.6

- Evaluation License Enforcement Mechanism - evaluation version of BEA AquaLogic Service Registry requires an evaluation license
- Integration with LDAP/MS Active Directory - BEA AquaLogic Service Registry accounts able to integrate with legacy systems using WASP Userstore
- Approval Process - staging-production pattern used to approve data stored in the registry;
- Direct access to back-end services - BEA AquaLogic Service Registry services implementations are now directly accessible
- Administration
  - configuration is now transparent for clustered installations
  - selected elements in configuration file can be signed to avoid their changes
  - created registry privileged users - extended administrators
  - admin and superuser able to switch to different user identity

- Localization - support for easier localization.

- Wildcards - selected databases support wildcard queries.
- Demos - demos simplified and refactored.
- WSDL Best Practice - Using WSDL in a UDDI Registry, Version 2.0 Technical Note supported.
- UDDI Client
- Operation timeout can be set per request.
- Serialization of UDDI API structures from/to XML file, DOM, String.
- Distribution contains the new UDDI client to be used in future releases of BEA AquaLogic Service Registry.

2.3.3. WASP UDDI 4.5.2

- Bugfixes - Fixes of major bugs found after 4.5 and 4.5.1 releases
- New application servers - Sun ONE Application Server 7
- Taxonomies - Added possibility to configure all combinations of tModelKey and keyName, and keyValue (tModelKey and keyName; tModelKey and keyValue; and tModelKey, keyName, and keyValue) when searching for specific taxonomies by keyedReferences.
- Administration - Added cleaner for account audit and subscriptions

2.3.4. WASP UDDI 4.5.1

- Runtime - Used WASP Server for Java, 4.5.1 runtime.
- Database schema - Database schemas changed to reflect optimizations.
- Performance optimizations - Improved performance for high load of data in database.
- New application servers - BEA WebLogic 6.1 SP3, BEA WebLogic 7.0.
- Database installation - Added database installation to BEA AquaLogic Service Registry installation.
- GUI database tool - New database tool for database creation, delete and migration.
- Security Enhancements - Security enhanced with:
  - password structure and length checking
  - password/account expiration
  - repeated failed logins checking
  - access to configuration access can be restricted by IP mask
- WASP Secure Identity - Integration with WASP Secure Identity is not supported any more.
- Web Interface look and feel - New web interface look and feel used.
- Support for NT service - BEA AquaLogic Service Registry can be now run as NT service.

2.3.5. WASP UDDI 4.5

- Hypersonic SQL - Embedded Hypersonic SQL 1.7.1 database. New demo database pre-configured for evaluation purposes.
- GUI Upgrade - New graphical upgrade of both registry and database.
- Taxonomy refactoring - Taxonomy publication and validation refactored.
  - Added new TaxonomyAdminApi for taxonomy administration.
  - Changed specification of taxonomy compatibility
  - Unified definition of validation services as specified in Providing a Taxonomy for Use in UDDI Version 2.
  - Created Validation Plug-ins to allow creation of custom taxonomy validators.
- Change UUID - UUIDs can be now changed for all UDDI basic data structures (businessEntity, businessService, bindingTemplate, tModel) using AdminToolApi
- Category dependencies - New tModel systinet-org:dependency introduced to allow specification of dependencies between UDDI entities.
- Other API Changes:
  - UDDIProxy - added save_wsdlTmodel methods
  - find_relatedServices extended with fromServiceKey and toServiceKey
- Demos - Created new demos structure.
- Database schema - Database schemas changed to reflect new features.
- GUI Installation - New graphical installation.
- Subscriptions - Allows client to subscribe for changes of any UDDI entities that occur in BEA AquaLogic Service Registry. There are two basic ways how the subscription is used: asynchronous notification and synchronous pull subscription.
- BEA AquaLogic Service Registry Interceptor API - The UDDI interceptor allows implementing customized handling of UDDI requests and responses.
- Selective One Way Replication - Replication based on subscription-notification mechanism. An asynchronous subscription listener listens to incoming subscription data from a master registry.
- UDDI Errata - Incorporated last errata from UDDI.org
  - UDDI Version 2.04 API
  - UDDI Version 2.03 Data Structure Reference
- API Extensions - Extended Inquiry Extensions merged with Access Control API and enhanced with:
  - new assertion related API calls

- enhanced wsdl related API calls
- added categoryBag into bindingTemplate and related API calls extended with categoryBag

- Administration - Configurable direct deletion of tModels.

2.3.6. WASP UDDI 4.0

- InstallShield - Graphical installation tool, InstallShield added.
- PointBase - Support for PointBase 4.3 database added.
- Oracle 9i - Oracle 9i AS (OC4J) port added.
- Disabled Runtime Services - System services removed from BEA AquaLogic Service Registry runtime.
- Extended installation - Installation extended with security providers configuration.
- Web interface design changed - Improved the look and feel of the web interface.
- JDK 1.4 Support - BEA AquaLogic Service Registry now support Sun's implementation of JDK 1.4.
- Porting - BEA WebLogic and Orion port scripts and documentation included.
- Taxonomy and Validation - Additional Taxonomy and Validation services integrated into the web interface.

3. Supported Platforms

BEA AquaLogic Service Registry 6.5 has been tested on the following platforms.

- Operating systems:
  - RedHat Enterprise 2.1 and 3.0 [http://www.redhat.com]
  - Solaris 9 [http://www.sun.com/software/solaris/]
  - Windows 2003 Server [http://www.microsoft.com/windows2003/]
  - Windows 2000 SP4 [http://www.microsoft.com/windows2000/]
  - Windows XP SP2 [http://www.microsoft.com/windowsxp/]
  - AIX 5.2 and 5.3 [http://www-1.ibm.com/servers/aix/]

- JDKs:
  - Sun JDK 1.4.2 and 1.5.0 [http://java.sun.com/j2se/]
  - BEA JRockit 1.4.2 and 1.5
  - IBM Java 1.4 or higher

- Databases:
  - Oracle 9.2.0.7 and 10.0.1.5 [http://www.oracle.com]

  - Microsoft SQL Server 2000 [http://www.microsoft.com/sql/default.asp]
  - DB2 8.X [http://www-3.ibm.com/software/data/db2/]
  - PostgreSQL 7.3 [http://www.postgresql.org]
  - Sybase ASE 12.5 [http://www.sybase.org]
  - HSQL 1.7.3 [http://hsqldb.org]

- LDAP:
  - Sun One Directory Server 5.2 [http://www.sun.com]
  - Microsoft Active Directory (Windows 2003 Server) [http://www.microsoft.com]

- Application Servers:
  - BEAWebLogic 8.x, 9.0, and 9.1 [http://www.bea.com]

- Browsers:
  - Microsoft Internet Explorer 5.5 and 6.0
  - Firefox 1.0

4. Specifications

BEA AquaLogic Service Registry conforms to the following specifications:

- UDDI Specifications [http://uddi.org/specification.html]
- UDDI Version 1 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1]
- UDDI Version 2 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2]
- UDDI Version 3 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]
- Technical Note Using WSDL in a UDDI Registry, Version 2.0 [http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm]

5. Document Conventions

This section describes conventions used in this document.

Command syntax	The syntax of operating system commands. We use square brackets ([ ]) to indicate optional parameters, a vertical bar (|) to indicate a choice of parameters, ... to indicate that the parameter is repeatable and { } to indicate that at least one repeatable parameter must be included. For example: Enter this command to start the UserStore tool: `UserStoreTool` {[-t *`target_server`*] | [--file *`userstore_file`*]...} [*`option`*...]

`Command` instances	Operating system commands and other user input that you can type on the command line and press **Enter** to invoke. These may be contained within text, as in this example: The command **java -jar systinet-server-java.jar** does not work on some encodings. If you have any problems starting the installer, try running **java -classpath systinet-server-java.jar Install -***-installation_option* instead. The command line may be separated, in which case it has a screen background: `java -jar systinet-server-java.jar --help`
`Filename`	Filenames, directory names, paths and package names. For example: Run the `install.bat` or `install.sh` script from the `bin` directory of the new distribution.
`XML tags`	XML element and attribute names. For example: `use ref="customSerialization"`

`Code block`	Program source code. For example:
	
	`package examples.helloWorld;`
	
	`public interface HelloWorldProxy {`
	`  String hello (String message);`
	`}`

Key-Key	A combination of keystrokes. Press the indicated keys simultaneously. For example: Press **Ctrl-Alt-Del** to reboot your computer.
GUI elements	A label, word or phrase in a GUI window, often clickable. For example: To edit a server's attributes, click on the server's link in the **Security Domain Tree** or click on **Detail** in the server's row in the **Managed BEA AquaLogic Servers** form. GUI buttons have a special icon, for example: Click on the **Save Configs** button to save your changes after editing any Domain or Server properties.
A menu selection	For example: Select **Property**->**Add property**

We use the following formatting elements to draw your attention to certain pieces of information:

Note

A Note indicates information that emphasizes or supplements points within the main text. Typically, a note provides information that may apply only in specific situations.

Tip

A Tip provides a helpful hint concerning procedures described in the text. It may suggest alternative methods or provide useful information about the capabilities of the product.

Important

An Important note provides critical information for the completion of a task. Do not disregard an Important note.

Caution

A Caution describes a situation where failure to take or avoid a specified action could result in a loss of data.

6. Legal

6.1. Third Party Licenses

6.1.1. HSQLDB License

Copyright (c) 1995-2000, The Hypersonic SQL Group. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the Hypersonic SQL Group nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE HYPERSONIC SQL GROUP, OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Hypersonic SQL Group.

For work added by the HSQL Development Group:

Copyright (c) 2001-2004, The HSQL Development Group All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of the HSQL Development Group nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL HSQL DEVELOPMENT GROUP, HSQLDB.ORG, OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.1.2. PostgreSQL 7.4

Legal Notice

Copyright © 1997-2005, PostgreSQL Global Development Group

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of the PostgreSQL Global Development Group nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.1.3. The Apache XML License, Version 1.1

The Apache Software License, Version 1.1

Copyright (c) 1999-2000 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xerces" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==========

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, International Business Machines, Inc., http://www.ibm.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

6.1.4. Apache Jakarta License, Version 1.1

==========

The Apache Software License, Version 1.1

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowlegement: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowlegement may appear in the software itself, if and wherever such third-party acknowlegements normally appear.

4. The names "The Jakarta Project", "Tomcat", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written permission of the Apache Group.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==========

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

6.1.5. CUP Parser Generator

CUP Parser Generator Copyright Notice, License, and Disclaimer

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of the authors or their employers not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

6.1.6. Jetty License, Version 3.6

Jetty License

Revision: 3.6

Preamble:

The intent of this document is to state the conditions under which the Jetty Package may be copied, such that the Copyright Holder maintains some semblance of control over the development of the package, while giving the users of the package the right to use, distribute and make reasonable modifications to the Package in accordance with the goals and ideals of the Open Source concept as described at http://www.opensource.org.

It is the intent of this license to allow commercial usage of the Jetty package, so long as the source code is distributed or suitable visible credit given or other arrangements made with the copyright holders.

Definitions:

- "Jetty" refers to the collection of Java classes that are distributed as a HTTP server with servlet capabilities and associated utilities.

- "Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

- "Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder.

- "Copyright Holder" is whoever is named in the copyright or copyrights for the package.

Mort Bay Consulting Pty. Ltd. (Australia) is the "Copyright Holder" for the Jetty package.

- "You" is you, if you're thinking about copying or distributing this Package.

- "Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

- "Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

0. The Jetty Package is Copyright (c) Mort Bay Consulting Pty. Ltd. (Australia) and others. Individual files in this package may contain additional copyright notices. The javax.servlet packages are copyright Sun Microsystems Inc.

1. The Standard Version of the Jetty package is available from http://www.mortbay.com.

2. You may make and distribute verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you include this license and all of the original copyright notices and associated disclaimers.

3. You may make and distribute verbatim copies of the compiled form of the Standard Version of this Package without restriction, provided that you include this license.

4. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.

5. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:

a) Place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as ftp.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.

b) Use the modified Package only within your corporation or organization.

c) Rename any non-standard classes so the names do not conflict with standard classes, which must also be provided, and provide a separate manual page for each non-standard class that clearly documents how it differs from the Standard Version.

d) Make other arrangements with the Copyright Holder.

6. You may distribute modifications or subsets of this Package in source code or compiled form, provided that you do at least ONE of the following:

a) Distribute this license and all original copyright messages, together with instructions (in the about dialog, manual page or equivalent) on where to get the complete Standard Version.

b) Accompany the distribution with the machine-readable source of the Package with your modifications. The modified package must include this license and all of the original copyright notices and associated disclaimers, together with instructions on where to get the complete Standard Version.

c) Make other arrangements with the Copyright Holder.

7. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you meet the other distribution requirements of this license.

8. Input to or the output produced from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package.

9. Any program subroutines supplied by you and linked into this Package shall not be considered part of this Package.

10. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

11. This license may change with each release of a Standard Version of the Package. You may choose to use the license associated with version you are using or the license of the latest Standard Version.

12. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

13. If any superior law implies a warranty, the sole remedy under such shall be , at the Copyright Holders option either a) return of any price paid or b) use or reasonable endeavours to repair or replace the software.

14. This license shall be read under the laws of Australia.

6.1.7. W3C Software Notice and License

W3C(C) SOFTWARE NOTICE AND LICENSE

Copyright (C) 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including modifications, that you make:

The full text of this NOTICE in a location viewable to users of the redistributed or derivative work. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, a short notice of the following form (hypertext is preferred, text is permitted) should be used within the body of any redistributed or derivative code: "Copyright (C) [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/"

Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

6.1.8. Xalan, Version 2.5.1

The Apache Software License, Version 1.1

Copyright (c) 1999-2003 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Xalan" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.

5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
==========

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright (c) 1999, Lotus Development Corporation., http://www.lotus.com. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

6.1.9. XML Pull Parser for Java, 1.1.1

Indiana University Extreme! Lab Software License

Version 1.1.1

Copyright (c) 2002 Extreme! Lab, Indiana University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu/)."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Indiana Univeristy" and "Indiana Univeristy Extreme! Lab" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact http://www.extreme.indiana.edu/.

5. Products derived from this software may not use "Indiana Univeristy" name nor may "Indiana Univeristy" appear in their name, without prior written permission of the Indiana University.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS, COPYRIGHT HOLDERS OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

6.2. Copyright

Copyright © 2006 BEA Systems, Inc. All Rights Reserved.

6.3. Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

6.4. Trademarks or Service Marks

BEA, BEA JRockit, BEA Liquid Data for WebLogic, BEA WebLogic Server, Built on BEA, Jolt, JoltBeans, SteelThread, Top End, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA AquaLogic, BEA AquaLogic Data Services Platform, BEA AquaLogic Enterprise Security, BEA AquaLogic Service Bus, BEA AquaLogic Service Registry, BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Manager, BEA MessageQ, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Java Adapter for Mainframe, BEA WebLogic JDriver, BEA WebLogic JRockit, BEA WebLogic Log Central, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic WorkGroup Edition, BEA WebLogic Workshop, and Liquid Computing are trademarks of BEA Systems, Inc. BEA Mission Critical Support is a service mark of BEA Systems, Inc. All other company and product names may be the subject of intellectual property rights reserved by third parties.

All other trademarks are the property of their respective companies.

6.5. Acknowledgements

This product includes software developed by the Apache Software Foundation (http://www.apache.org).

This product includes code licensed from RSA Data Security (http://www.rsasecurity.com).

This product includes software developed by jGuru.com (MageLang Institute) (http://www.jGuru.com).

This product includes Antlr (http://www.antlr.org).

This product contains components derived from software developed by the Indiana University Extreme! Lab (http://www.extreme.indiana.edu).

The Standard Version of the Jetty package is available from http://www.mortbay.com.

7. Support

If you need support for any BEA product, you can access the BEA customer support web site at http://support.bea.com/.

You can also find links to Discussion Lists, FAQs, Demos, Tutorials, and other valuable support resources on our developer site at http://dev2dev.bea.com/.

Documentation for the full BEA range of products is available at http://edocs.bea.com/.

7.1. Contact

BEA Corporate Headquarters

BEA Systems, Inc.

2315 North First Street

San Jose, CA 95131

+1.800.817.4BEA (US toll free)

+1.408.570.8000 phone

+1.408.570.8901 fax

User's Guide

The BEA AquaLogic Service Registry User's Guide is mainly focused on the web user interface. The users to whom this guide is addressed are those who query the registry or publish to it using this interface as opposed to accessing the registry over SOAP. It is comprised of the following sections:

Introduction to BEA AquaLogic Service Registry

This section is a brief intoduction to BEA AquaLogic Service Registry including basic concepts of UDDI specifications.

Registry Consoles

This section presents both Business Service Console and Registry Console

Demo Data Description

The BEA AquaLogic Service Registry's Demo Data chapter describes the business domain and UDDI data structures used in the BEA AquaLogic Service Registry Demo Suite and both registry consoles.

Business Service Console

Describes the Business Service Console and basic tasks you can perform with it.

Advanced Topics

Access Control Principles

Describes principles of permissions and access control to UDDI data structures.

Publisher-Assigned Keys

Under UDDI v3, users may assign alpha-numeric keys to structures rather than having these keys automatically generated by the registry (as was the case under UDDI v1 and v2).

Range Queries

BEA AquaLogic Service Registry's range queries functionality allows you to search UDDI entities with the ability to use comparative operators (>, <).

Taxonomy: Principles, Creation and Validation

This section gives you a brief overview of taxonomy classification in BEA AquaLogic Service Registry

Registry Console Reference

Describes the Registry Console and basic tasks you can perform with it.

Signer Tool

Allows the user to digitally sign published UDDI structures and validate digital signatures.

1. Introduction to BEA AquaLogic Service Registry

BEA AquaLogic Service Registry is a fully V3-compliant implementation of the UDDI (Universal Description, Discovery and Integration) specification, and is a key component of a Service Oriented Architecture (SOA). A UDDI registry provides a standards-based foundation for locating services, invoking services and managing metadata about services (security, transport or quality of service). A UDDI registry can store and provide these metadata using arbitrary categorizations. These categorizations are called taxonomies.

This introduction has the following sections:

- Section 1.1, UDDI's Role in the Web Services World - UDDI Benefits
- Section 1.2, Typical Application of a UDDI Registry
- Section 1.3, Basic Concepts of the UDDI Specification

- Section 1.4, Subscriptions in BEA AquaLogic Service Registry
- Section 1.5, Approval Process in BEA AquaLogic Service Registry

1.1. UDDI's Role in the Web Services World - UDDI Benefits

When development teams start to build Web service interfaces into their applications, they face such issues as code reuse, ongoing maintenance and documentation. The need to manage these services can increase rapidly.

The UDDI registry can help to address these issues and provides the following benefits:

- It delivers **visibility** when identifying which services within the organization can be reused to address a business need.
- It promotes **reuse** and prevents reinvention. It accelerates development time and improves productivity. This ability of UDDI to categorize a growing portfolio of services makes it easier to manage them. It helps you understand relationships between components, supports versioning and manages dependencies.
- It supports service **configurability and adaptability** by using the service-oriented architectural principle of location and transport independence. Users can dynamically discover services stored in the UDDI registry.
- It allows you to understand and manage **relationships** between services, component versions and dependencies.
- It makes it possible to manage the **business service lifecycle**. For example, the process of moving services through each phase of development, from coding to public deployment. For more information, see the Approval Process.

1.2. Typical Application of a UDDI Registry

A UDDI registry stores data and metadata about business services. A UDDI registry offers a standards-based mechanism to classify, catalog and manage Web services so that they can be discovered and consumed by other applications. As part of a generalized strategy of indirection among services-based applications, UDDI offers several benefits to IT managers at both design-time and run-time, including increasing code reuse and improving infrastructure management by:

- Publishing information about Web services and categorization rules (taxonomies) specific to an organization.
- Finding Web services that meet given criteria.
- Determining the security and transport protocols supported by a given Web service and the parameters necessary to invoke the service.
- Providing a means to insulate applications (and providing fail-over and intelligent routing) from failures or changes in invoked services.

1.3. Basic Concepts of the UDDI Specification

UDDI is based upon several established industry standards, including HTTP, XML, XML Schema (XSD), SOAP, and WSDL. The latest version of the UDDI specification is available at: http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3.

The UDDI specification describes a registry of Web services and its programmatic interfaces. UDDI itself is a set of Web services. The UDDI specification defines services that support the description and discovery of:

- Businesses, organizations and other providers of Web services;
- The Web services they make available;
- The technical interfaces which may be used to access and manage those services.

1.3.1. UDDI Data Model

The basic information model and interaction framework of UDDI registries consist of the following data structures:

- A description of a service business function is represented as a `businessService`.
- Information about a provider that publishes the service is put into a `businessEntity`.
- The service's technical details, including a reference to the service's programmatic interface or API, is stored in a `bindingTemplate`.
- Various other attributes, or metadata, such as taxonomy, transports, and policies, are stored in `tModel`s.

These UDDI data structures are expressed in XML and are stored persistently by a UDDI registry. Within a UDDI registry, each core data structure is assigned a unique identifier according to a standard scheme. This identifier is referred as a UDDI key.

Business Entity

A business entity represents an organization or group of people responsible for a set of services (a service provider). It can also represent anything that overreaches a set of services; for example a development project, department or organization. The business entity structure contains the following elements:

- Names and Descriptions. The business entity can have a set of names and descriptions, in a variety of languages if necessary.
- Contacts. The list of people who are associated with the business entity. A contact can include, for example, a contact name, addresses, phone numbers, and use type.
- Categories. Set of categories that represent the business entity's features or quantities. For example the business entity can be associated with the category California to say that the business entity is located in that geographical area.
- Identifiers. The business entity can be associated with arbitrary number of identifiers that uniquely identify it. For example, the business entity can be identified by a department number or D-U-N-S number.
- Discovery URLs are additional links to documents describing the business entity.

Business entities can be linked to one another using so-called assertions that model a relationships between them.

Business Service

Business services represent functionality or resources provided by business entities. A business entity can reference multiple business services. A business service is described by the following elements:

- Names and descriptions. The business service can have a set of names and descriptions, in a variety of languages if necessary.
- Categories. A set of categories that represent the business service features and quantities. For example, the business service can be associated by a category that represents service availability, version, etc.

A business service in a UDDI registry does not necessarily represent a Web service. The UDDI registry can register arbitrary services such as example EJB, CORBA, etc.

Binding Template

A business service can contain one or more binding templates. A binding template represents the technical details of how to invoke its service. Binding templates are described by the following elements:

- Access point represents the service endpoint. It contains endpoint URI and specification of the protocol.
- tModel instance infos can be used to represent any other information about the binding template
- Categories. The binding template can be associated with categories to reference specific features of the binding template, for example certification status (test, production) or versions.

tModel

The tModel provides a reference to an abstraction describing compliance with a specification and concepts. TModels are described by the following elements:

- Name and description. The tModel can have a set of names and descriptions, in different languages if required.
- An overview document is a reference to a document that specifies the tModel's purpose.
- Categories. Like all the other UDDI entities, tModels can be categorized.
- Identifiers. The tModel can be associated with an arbitrary number of identifiers that uniquely identify it.

UDDI entities are categorized through tModels via taxonomies. Business entities, business services, and binding templates declare associations to a certain category by presence of specific tModels in their categoryBags.

1.3.2. Taxonomic Classifications

UDDI provides a foundation and best practices that help provide semantic structure to the information about Web services contained in a registry. UDDI allows users to define multiple taxonomies that can be used in a registry. Users can employ an unlimited number of appropriate classification systems simultaneously. UDDI also defines a consistent way for a publisher to add new classification schemes to their registrations.

Taxonomies are used for representing various UDDI entity features and qualities (such as product types, geographical regions or departments in a company).

The UDDI specification mandates several standard taxonomies that must be shipped with each UDDI registry product. Some are internal UDDI taxonomies such as the UDDI types taxonomy or geographical taxonomy. A taxonomy can be marked as specific to business, service, binding template or tModel or it can be used with any type of the UDDI entity

Enterprise Taxonomies

Enterprise taxonomies are taxonomies that are specific to the particular enterprise or application. These taxonomies reflect specific categories like company departments, types of applications, and access protocols.

BEA AquaLogic Service Registry allows definition of enterprise taxonomies. Users can also download and upload any taxonomy as an XML file. BEA AquaLogic Service Registry offers tools for creating, modifying and browsing taxonomies on both the web user interface and SOAP API levels.

Checked and Unchecked Taxonomies

There are two types of taxonomies: checked and unchecked. Checked taxonomies are rigid, meaning that the UDDI registry does not allow the use of any categories other than those predefined in the taxonomy. Checked taxonomies are usually used when the taxonomy author can enumerate all distinct values within the taxonomy. A checked taxonomy can be **validated** using the internal validation service that is available in BEA AquaLogic Service Registry or by using an external validation service.

Unchecked taxonomies do not prescribe any set of fixed values and any name and value pair can be used for categorization of UDDI entities. Unchecked taxonomies are used for things like volume, weight, price, etc. A special case of the unchecked taxonomy is the `general_keywords` taxonomy that allows categorizations using arbitrary keywords.

1.3.3. Security Considerations

UDDI specification does not define an access control mechanism. The UDDI specification allows modification of the specific entity only by its owner (creator). This does not scale in the enterprise environment where the right to modify or delete a specific UDDI entity must be assigned with more identities or even better with some role.

BEA AquaLogic Service Registry addresses this issue with the ACL (Access Control List) extension to the UDDI security model. Every UDDI entity can be associated with the ACL that defines who can find (list it in some UDDI query result), get (retrieve all details of the UDDI object), modify or delete it. The ACL can reference either the specific user account or user group.

The UDDI v3 specification provides support for digital signatures. In BEA AquaLogic Service Registry, the publisher of a UDDI structure can digitally sign that structure. The digital signature can be validated to verify the information is unmodified by any means and confirm the publisher's identity.

1.3.4. Notification and Subscription

The UDDI v3 specification introduces notification and subscription features. Any UDDI registry user can subscribe to a set of UDDI entities and monitor their creation, modification and deletion. The subscription is defined using standard UDDI get or find API calls. The UDDI registry notifies the user whenever any entity that matches the subscription query changes even if the change causes the entity to not match the query anymore. It also notifies about entities that were changed in a way that after the change they match the subscription query.

The notification might be synchronous or asynchronous. By synchronous, we mean solicited notification when the interested party explicitly asks for all changes that have happened since the last notification. Asynchronous notifications are run periodically in a configurable interval and the interested party is notified whenever the matched entity is created, modified, or deleted.

1.3.5. Replication

Content of the UDDI registry can be replicated using the simple master-slave model. The UDDI registry can replicate data according to multiple replication definitions that are defined using UDDI standard queries. The master-slave relationship is specific to the replication definition. So one registry might be master for one specific replication definition and slave for another. The security settings (ACL, users, and groups) are not subject to replication but you can set permissions on replicated data.

1.3.6. UDDI APIs

The core data management tools functions of a UDDI registry are:

- Publishing information about a service to a registry.
- Searching a UDDI registry for information about a service.

The UDDI specification also includes concepts of:

- Replicating and transferring custody of data about a service.
- Registration key generation and management.
- Registration subscription API set.
- Security and authorization.

The UDDI specification divides these functions into `Node API sets` that are supported by a UDDI server and `Client API Sets` that are supported by a UDDI client .

1.3.7. Technical Notes

Technical Notes (TN) are non-normative documents accompanying the UDDI Specification that provide guidance on how to use UDDI registries. Technical Notes can be found at http://www.oasis-open.org/committees/uddi-spec/doc/tns.htm. One of the most important TNs is "Using WSDL in a UDDI Registry".

1.3.8. Benefits of UDDI Version 3

The most important features include:

- **User-friendly identifiers** facilitate reuse of service descriptions among registries.
- **Support for digital signatures** allows UDDI to deliver a higher degree of data integrity and authenticity.
- **Extended discovery features** can combine previous, multi-step queries into a single-step, complex query. UDDI now also provides the ability to nest sub-queries within a single query, letting clients narrow their searches much more efficiently.

1.4. Subscriptions in BEA AquaLogic Service Registry

Subscriptions are used to alert interested users in changes made to structures in BEA AquaLogic Service Registry. The BEA AquaLogic Service Registry Subscription API provides users the ability to manage (save and delete) subscriptions and evaluate notification. Notifications are lists of changes made within a specified time interval. The Subscription mechanism allows the user to monitor new, changed, and deleted entries for businessEntities, businessServices, bindingTemplates, tModels or publisherAssertions. The set of entities in which a user is interested is expressed by a SubscriptionFilter, which can be any one of the following UDDI v3 API queries:

- `find_business`, `find_relatedBusinesses`, `find_services`, `find_bindings`, `find_tmodel`
- `get_businessDetail`, `get_serviceDetail`, `get_bindingDetail`, `get_tModelDetail`, `get_assertionStatusReport`

Note

In Business Service Console, users can also create subscriptions also resources (WSDL, XML, XSD and XSLT) without a detailed knowledge of how resources are mapped to UDDI data structures.

1.4.1. Subscription Arguments

A subscription is the subscriber's interest in changes made to entities as defined by the following arguments:

- `SubscriptionKey` - The identifier of the subscription, as generated by the server when the subscription is registered.
- `Subscription Filter` - Specifies the set of entities in which the user is interested. This field is required. Note that once the subscription filter is set, it cannot be changed.
- `Expires After` - The time after which the subscription is invalid (optional).
- `Notification Interval` - How often the client will be notified (optional). The server can extend it to the minimum supported notification interval supported by the server as configured by the administrator.

  For more information, please see Administrator's Guide, Section 2, Registry Configuration.
- `Max Entities` - how many entities can be listed in a notification (optional). When the number of entities in a notification exceeds max entities, the notification will contain only the number of entities specified here or in the registry configuration. A chunkToken different from "0" will be specified in the notification. This chunkToken can be used to retrieve trailing entities.

- `BindingKey` - points to the bindingTemplate that includes the endpoint of the notification handling service (optional). Only http and mail transports are currently supported. If this bindingKey is not specified, the notification can be retrieved only by synchronous calls.

- `Brief` - By default, notifications contain results corresponding to the type of the Subscription Filter. For example, when the subscription filter is find_business, notifications contain Business Entities in the businessInfos form. If brief is toggled on, notifications will contain only the keys of entities. (optional)

1.4.2. Subscription Notification

Notification is the mechanism by which subscribers learn about changes. Notifications inform subscribers about entities that:

1. Satisfy the Subscription Filter now and were last changed, or created, within a given time period. The entities are included in a list of the appropriate data type by default. For example, when find_business represents the Subscription Filter, notifications contain Business Entities in the businessList/businessInfo form. (If the brief switch is toggled on, only the entity keys in the keyBag are included.)

2. Were changed or deleted in the given time period and no longer satisfy the Subscription Filter. Only the keys of the appropriate entities are included in the keyBag structure and the `deleted` flag is toggled on.

There are two types of notifications:

- `Asynchronous notification` - Using asynchronous notification, the server periodically checks for changes and offers them to the client via HTTP or SMTP. HTTP is suitable for services listening to UDDI changes. SMTP (that is, mail notification) is suitable for both services and users. With this transport, the user is notified at each notification interval by email. To perform asynchronous notification, the subscription must be populated with `notification interval` and `bindingKey`. See Developer's Guide, Section 3.5, Writing a Subscription Notification Service for details.

- `Synchronous notification` - Using synchronous notification, the server checks for changes and offers them when the client explicitly asks for them outside of periodical asynchronous notifications. It is useful for client applications which cannot listen for notifications, and for services that want to manage the time of notification by themselves. See Demos, Section 2.3, Subscription for details.

1.4.3. XSLT Over Notification

To improve the readability of notifications sent to users via email, BEA AquaLogic Service Registry provides the ability to process the XSL transformation before the notification is sent. To enable this feature:

1. Register the XSL transformation in UDDI as a tModel that refers to XSL transformation in its first overviewDoc.

2. Modify the bindingTemplate (with the bindingKey specified in the subscription) to refer to the XSLT tModel by its tModelInstanceInfo.

3. Tag the XSLT tModel by a keyedReference to `uddi:uddi.org:resource:type` with the keyValue="`xslt`".

1.4.4. Suppressing Empty Notifications

Another BEA AquaLogic Service Registry extension to the specification is the ability to suppress empty notifications. To do this, tag the bindingTemplate referenced from the subscription with a keyedReference to the tModel `uddi:uddi.org:categorization:general_keywords` with keyValue="`suppressEmptyNotification`" and keyName="`suppressEmptyNotification`".

1.4.5. Related Links

- To manage subscriptions via the Business Service Console, see the section Business Service Console Subscriptions.

- To manage subscriptions via the Registry Console, see the Registry Console Reference.
- To use and manage subscriptions, see the Subscription API.
- More details about subscriptions can be found in the Subscription API [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047327] chapter of the UDDI v3 Specification.

1.5. Approval Process in BEA AquaLogic Service Registry

The approval process provides functionality to ensure consistency and quality of data stored in BEA AquaLogic Service Registry. There are two registries in the approval process:

- a *publication registry* is used for testing and verification of data;
- a *discovery registry* only contains data that has been approved and promoted from the publication registry.

See the Installation Guide [http://edocs.bea.com/alsr/docs21/install/] for details of how to install and configure these registries.

The approval process includes two types of users:

- A *requestor* is a user of the *publication* registry who can request approval of data for promotion to the *discovery* registry;
- An *approver* is a user who can approve or reject requests for promotion of data.

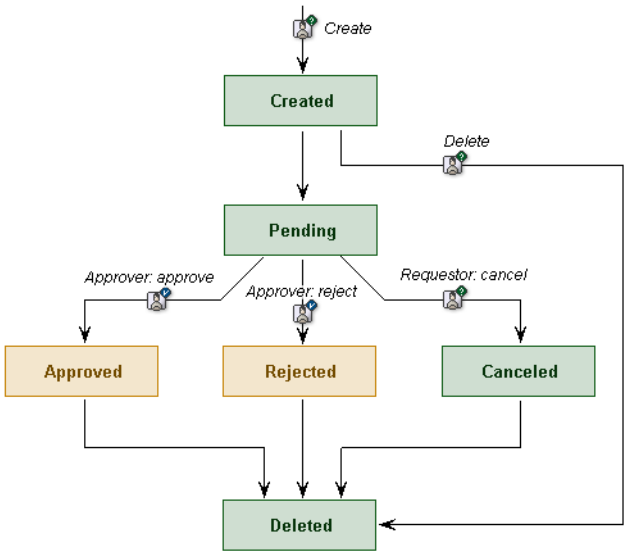
Adminstrators can specify:

- the users or groups of users who are approvers;
- the users or groups of users whose requests they can approve;

Every user can ask for approval, but to have data considered for promotion, a user must have an administrator-assigned approver.

For more information see Section 1.7, Approval Process Management in the *Administrator's Guide*.

Figure 1. Approval Request Lifecycle



Approval requests have a lifecycle shown in Figure 1. A requestor can create a request. Once the request is created, the requestor can add UDDI data structures (described in Section 1.3.1, UDDI Data Model) or resources (WSDL, XML, XSD and XSLT) to the request. Note that the requestor does not need to know how resources are mapped to UDDI data structures. When the requestor adds a resource to the request, all underlying UDDI structures (bindings, tModels) the resource represents are automatically added to the request. Once the requestor specifies all entities to promote, the request may be submitted for approval.

The approver will review incoming requests, and then can approve or reject the request. If the approver approves the request, the requested data is immediately promoted to the *discovery* registry. If the requestor is not satisfied with the approver's response time, this user can remind the approver to review the requests. The requestor can also cancel submitted requests.

In the following section, we will look at requestor's and approver's actions in detail.

1.5.1. Requestor's Actions

A requestor may perform the following actions:

- Submit a request for approval of data promotion

  After submitting the request, all data referenced in the request is blocked (locked for writing) until the request is either canceled by the requestor, approved for promotion, or rejected by the approver.

  Note

  A requestor may request approval for the promotion of the same set of data several times, and may have several unprocessed requests at one time.

- Find request.

  This action provides the requestor with the ability to list information about all requests. If the requestor has privileged access on the Requestor API, then it is possible to get brief information on the requests of other users. Otherwise only the requestor's own requests may be viewed.

- Get request

  This action returns full information about the given request. If the requestor has privileged access on the Requestor API then they can obtain full details of other user's requests. Otherwise only the requestor's own requests may be accessed.

- Cancel request

  Provides requestor with the ability to cancel the given request. Only requestors with privileged access can cancel the requests of other requestors.

- Synchronize data

  This action enables the requestor to synchronize data on the publication registry with data on the discovery registry. There are three types of synchronization - publication priority, partial discovery priority, and full discovery priority. For detailed information about synchronization, please see Synchronization of Data.

To publish data to the discovery registry, the data must first be published to the publication registry and then approved by an appropriate approver. Once the requestor is satisfied with the quality of data, it is possible to ask for data promotion.

Requestors can publish data on the publication registry for testing. Once this data is ready for approval, the requestor asks for approval. An approval request contains two different sets of keys - keys for saving and keys for deletion. The keys select the data. Keys for saving are used for entities to be published (saved or updated) to the discovery registry. Keys for deletion can be used for deletion of any entity from the discovery registry. Approval requests can contain data (keys of entities) either for saving or for deletion.

Both types of keys can contain keys for businessEntities, businessServices, bindingTemplates, tModels or publisherAssertions. For example, if a requestor wants to promote a businessEntity to the discovery registry and remove a bindingTemplate from a service on the discovery registry then the request for approval must contain the key of the businessEntity in the keys for saving and the key of the bindingTemplate in the keys for deletion. After successful approval the business entity is saved (created or updated) to the discovery registry and the binding template is deleted from the discovery registry.

Context Checking

During a request for approval, and when approval is granted, automatic context checking is processed to ensure the integrity of data from a request. The context checker has the following rules:

- If an entity is contained in keys for saving, then the parent entity must already exist on the *discovery* registry or be contained in keys for saving to the *discovery* registry. For a businessService, the parent is a businessEntity; for a bindingTemplate, the parent is a businessService.

- An entity whose key is included in those for deletion may not be referenced by an entity whose key is included in those being saved.

- An entity whose key is included in those for deletion must exist on the *discovery* registry.

- Deleting a tModel that is referenced by entities on the *discovery* registry is not allowed.

- If a publisher assertion is included in keys for saving, then its business entities (specified in fromKey, toKey) and tModel must already exist on the *discovery* registry or be contained in keys for saving.

If the data is valid, according to these rules, the request for approval is made.

If data is invalid (for example, an entity is included in keys for deletion that does not exist on the *discovery* registry), an exception is thrown and the request for approval is not made.

If context checking fails, the requestor is informed that the data must somehow be changed before requesting approval again.

A Special Approval Case

If the registry administrator trusts a requestor, that requestor may be assigned the approval contact `AutoApprover`. Under this approval contact, there is no human review of the data. The data is automatically promoted to the *discovery registry* as long as automatic context checking is successful.

1.5.2. Approver's Actions

Approval contacts are assigned by users who have permission to set up the approval process via the ApprovalConfiguration API (such as registry administrator). The approval contact reviews requests to promote data to the *discovery* registry and approves or rejects these requests.

If enabled, content checking (additional rules applied to approved data) is performed at this time as well.

If context checking and content checking are successful, an email is sent to the requestor indicating the successful promotion of data, and including any message entered in the **Message for requestor** box.

Optional Content Checking

Optional content checking provides an approver with the ability to programmatically check data for approval. For example, the approver can set a policy that:

- Each business service must include a binding template, or
- Each business service must be categorized by specified categories

To enforce such a policy, a developer can write an implementation of the Checker API to enforce these checks. The implementation is called automatically during the approval process when an approver presses the **Approve request** button. So the approver does not have to check it manually. For more information on setting up optional content checking, see Section 6.2, Optional Content Checking Setup in the Administrator's Guide.

1.5.3. Synchronization of Data

Requestor's synchronization is used to synchronize the information on the *publication* and *discovery* registries. There are three different kinds of synchronization described below - `publication priority`, `partial discovery priority` and `full discovery priority`. Each is performed on all data structures associated with the synchronizing user's account. Synchronization is performed only upon request.

Note

These tools do not change information on the *discovery* registry. The only way to change data on *discovery* registry is via the *publication* registry and the approval process. Only administrator can publish to *discovery* registry.

Publication priority

Publication priority has the following rules:

- If an entity exists only on the *discovery* registry then it is copied to the *publication registry*.
- If an entity exists only on the *publication* registry then it is preserved.
- If an entity exists on both registries, then the *publication* registry takes priority over the *discovery* registry.

Publication Priority Example

Before synchronization, structures A and X exist on the *publication* registry and structures X and B exist on the *discovery* registry.

The Publication Priority synchronization copies structure B to the *publication* registry. Structure X on *publication* registry remains the same because when the same entity exists on both servers, Publication Priority synchronization favors the *publication* registry.

Partial Discovery Priority

Partial discovery priority has the following rules:

- If an entity exists only on the *discovery* registry, then it is copied from the *discovery* registry to the *publication* registry.
- If an entity exists only on the *publication* registry then it is preserved.
- If an entity exists on both registries, then data on the *publication* registry is overwritten by data from the *discovery* registry.

Partial Discovery Example

Before this synchronization, structures A and X exist on the *publication* registry and structures X and B exist on the *discovery* registry.

Partial discovery synchronization copies structure B to the *publication* registry and overwrites the version of structure X on the *publication* registry with that from the *discovery* registry.

Full Discovery Priority

Under this synchronization scenario, all the user's data on the *publication* registry is deleted, and all the user's data from *discovery* registry is copied to the *publication* registry. After full discovery priority synchronization, data on the *discovery* and *publication* registries is identical.

Important

The BEA AquaLogic Service Registry administrator cannot execute full discovery priority synchronization.

Full Discovery Example

Before synchronizing, structures A, X, Y and B exist on the *publication* registry and structures A, X and B exist on the *discovery* registry.

Full discovery synchronization deletes structures A, X, Y and B from the *publication* registry, and replaces them with A, X, and B from the *discovery* registry.

1.5.4. Related Links

- Installation of publication and discovery registries - Installation Guide [http://edocs.bea.com/alsr/docs21/install/].
- Approval process via Business Service Console - User's Guide, Section 4.8, Approval Process
- Configure requestors and approvers - Administrator's Guide, Section 1.7, Approval Process Management

2. Registry Consoles

BEA AquaLogic Service Registry provides two user interfaces.

- **Systinet Business Service Console** Using the Business Service Console developers, architects and business users can browse the various perspectives of the registry including business-relevant classifications such as service and interface lifecycle, compliance or operational/readiness status. They can browse information through business-relevant abstractions of SOA information such as schemas, interface local names or namespaces. The Business Service Console also provides easy to use and customizable publication wizards.

  The Business Service Console can be found at `http://<hostname>:<port>/uddi/bsc/web`. Host name and port are defined when BEA AquaLogic Service Registry is installed. The default port is 8080. See Section 4, Business Service Console

- **Registry Console** Using the Registry Console users can browse and publish registry contents, create subscriptions and perform ownership changes. The Registry Console is the primary console for administrators to perform registry management.

  The Registry Console can be found at `http://<hostname>:<port>/uddi/web`. Host name and port are defined when BEA AquaLogic Service Registry is installed. The default port is 8080. See Section 5.5.2, Registry Console Overview

Note

Make sure your browser allows HTTPS connections, supports JavaScript and does not block popup windows.

3. Demo Data

Demo data is pre-installed with BEA AquaLogic Service Registry. There are two demo data sets:

- demo data to demonstrate Business Service Console
- demo data to demonstrate Registry Console and Demo Suite

3.1. Demo Data for Business Service Console

Demo data is pre-installed with BEA AquaLogic Service Registry for use with the Business Service Console. This data describes a financial institution (bank) with several departments. It contains entities providing services for its operations. Entities providing services are modelled as service providers. There are the following providers and their services in the demo data:

Account Services

Account Services provides services related to account information, transfers, check orders, bill pay, online statements.

- Account - The account service provides the account related operations :`getAccount`, `listAccountDetail`, `listRelatedAccounts`, `listTransactionHistory`.
- Bill Payment - The bill payment service provides the ability to establish bill payment service, cancel bill payment service and get information about bill payment for a customer. Operations: `authorizeAcctForBillPymt`, `cancelBillPymtSvc`, `createBillPymtSvc`.
- Check Order - This service supports new check orders, check reorders, check order inquiry. Operations: `getLastCheckOrder`, `orderChecks`, `reorderChecks`.

- Direct Deposit Advance -This service supports the operations used to set up the advancement of money. Operation: `addDirectDepositAdvance`.
- Notification Services - This service is used to provide notifications. Operation: `sendAccountTransferNotification`.
- Stop Payment - This service allows stops to be set and maintained. Operations: addStopPaymentForCheck, `cancelStopPay`
- Transfer Funds - This service allows funds to be transferred from one account to another. Operations: `authorizeTransfer`, `sendInvoicePayment`, `transferFunds`.

Customer Management System

Customer relationship and management system.

- Add Customer - This service allows a customer to be added to the enterprise customer system. Operation: `addCustomer`.
- Customer Notification - This service provides notification messages for various customer changes. Operations: `customerNameChangeNotif`, `customerAddressChangeNotif`.

Outlet Locator

Provides information about outlets and sites.

- Outlet - The Outlet service gets all of the information about a Company outlet. Operation: `getOutletDetail`.
- Site - This service gets information about a site. Operations: `getSiteDetail`, `listSites`, `searchSites`

Document Services

Provides access to company forms.

- Electronic Forms - Provides access to company forms. Operations: `updateAddrPhone`, `updateNameAndTitle`.

Transaction Services

Middleware applications for posting transactions with high performance SLA.

- Monetary Transaction - Monetary Posting. Operation: `postTransaction`.

Each service has a WSDL definition. Demo data also contains information about service interfaces and endpoints including categorization as certification statuses, availability statuses, and stages of lifecycle.

3.2. Demo data for Registry Console and demos

Demo data describes a multinational company with offices in several locations and BEA AquaLogic Service Registry installed in its headquarters division. The headquarters division has two departments: IT and HR.

There are two predefined users, demo_john and demo_jane. The passwords for these users are the same as their log on names.

Departments are represented as the following Business Entities:

- Headquarters
- HR
- IT

The following taxonomies are used:

demo:hierarchy

Represents the organizational structure (hierarchy). KeyValue is the businessKey of the parent department.

demo:location:floor

Represents the geographical location of departments. Headquarters is located in a building; IT and HR are located in different floors of the same building. KeyValue is the number of the floor.

demo:departmentID

Identifies each department uniquely. The value from keyValue can be used as an argument in WSDL services.

Pre-published services are shown in Table 1, "Pre-published Demo Web Services":

Table 1. Pre-published Demo Web Services

Name	WSDL Service	Description
Holiday request	Yes	stored in the HR department; used by employees to submit holiday request
Phone support	No	stored in the IT department; used by employees to call IT phone support for help with their PCs.
Employee list	Yes	stored in the HR department, projected to IT department; takes single argument - departmentId; used by employees to view a list of employees that belong to a department.

Assertions are an alternate way to represent relationships between business entities. In the BEA AquaLogic Service Registry demo data, assertions are created between the Headquarters and HR departments.

The demo data also contains the following resource files located in the `REGISTRY_HOME/demos/conf` directory:

- EmployeeList.wsdl
- employees.xml
- employees.xsd
- employeesToDepartments.xsl
- departments.xml
- departments.xsd

4. Business Service Console

Using the Business Service Console, developers, architects and business users can browse the various perspectives of the registry including business-relevant classifications such as service and interface lifecycle, compliance or operational/readiness status. They can browse information through business-relevant abstractions of SOA information such as schemas, interface local names or namespaces. The Business Service Console also provides easy to use and customizable publication wizards.

The Business Service Console is designed to be consistent, intuitive and user friendly. This documentation demonstrates general procedures using typical examples. It has the following subsections:

Section 4.1, Overview A general description of the Business Service Console user interface.

Section 4.2, User Account User accounts and profiles.

Section 4.3, Searching Searching for providers and endpoints.

Section 4.4, Publishing Publishing providers and services.

Section 4.5, Reports The **Reports** tab.

Section 3, Business Service Console Configuration How an administrator can configure the Business Service Console according to your needs.

Section 4.7, Subscription and Notification How to create and manage subscriptions so that you are notified of changes to data stored in the registry.

Section 4.8, Approval Process The process for approval of publications from the perspective of a requestor or approver.

4.1. Overview

Figure 2 illustrates common features of the Business Service Console:

A: Main Menu Tabs The appearance of the **Main** menu tabs depends on your user profile.

Home

This is a good place to start navigating the Business Service Console since it contains many links.

Catalog

This tab allows you to list, search and publish entities on BEA AquaLogic Service Registry.

Tools

This tab allows you to view and manage subscriptions and approval requests.

Report

This tab allows you to view the predefined set of reports.

Configure

This tab allows you to configure the Business Service Console.

B: History Path (bread crumbs) This area displays the log of your recent actions. You can return to any of these previous actions by clicking on the hyperlinks.

C: Side Bar On some screens a side bar is available showing a list of item types.

D: Hide/Show Side Bar Click here to hide or show the side bar when available.

E: Main Display Area Information chosen from the tabs and the tree display is made available in the Main Display Area.

F: User Profile The name of the user profile of the currently logged in user.

G: Login/logout Here you can log in as a particular user or logout and use the Business Service Console anonymously.

H: Registry Name The name of the registry is taken from the name of the Operational Business Entity which represents the UDDI registry.

I: Action Icons There are two icons in this area. The first one allows you to refresh the page content, while the second one opens the product documentation page.

J: Reference Links Links at the bottom of the page. These are always the same and always there if you need them.

Figure 2. Example Business Service Console page

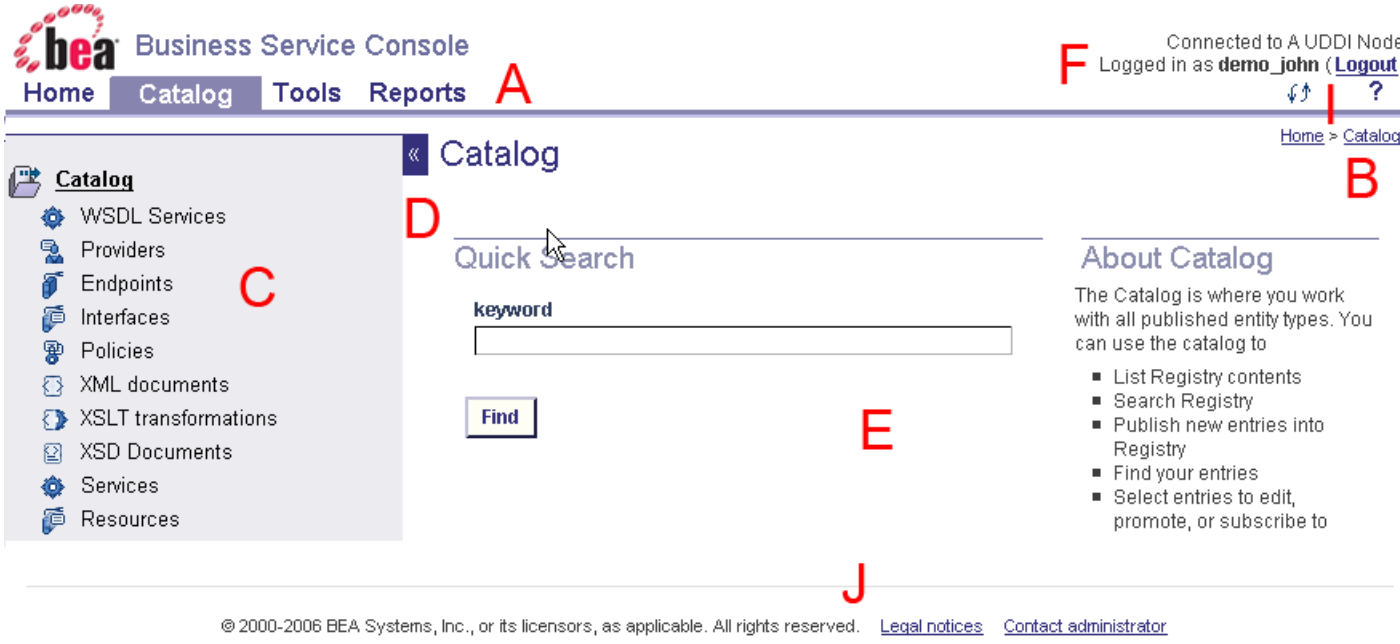


Figure shows features available on other screens:

V: Link to an entity References to entities or other resources appear in many places as links. Generally, clicking on such a link displays details of the resource. See Section 4.6.1, Entity Details.

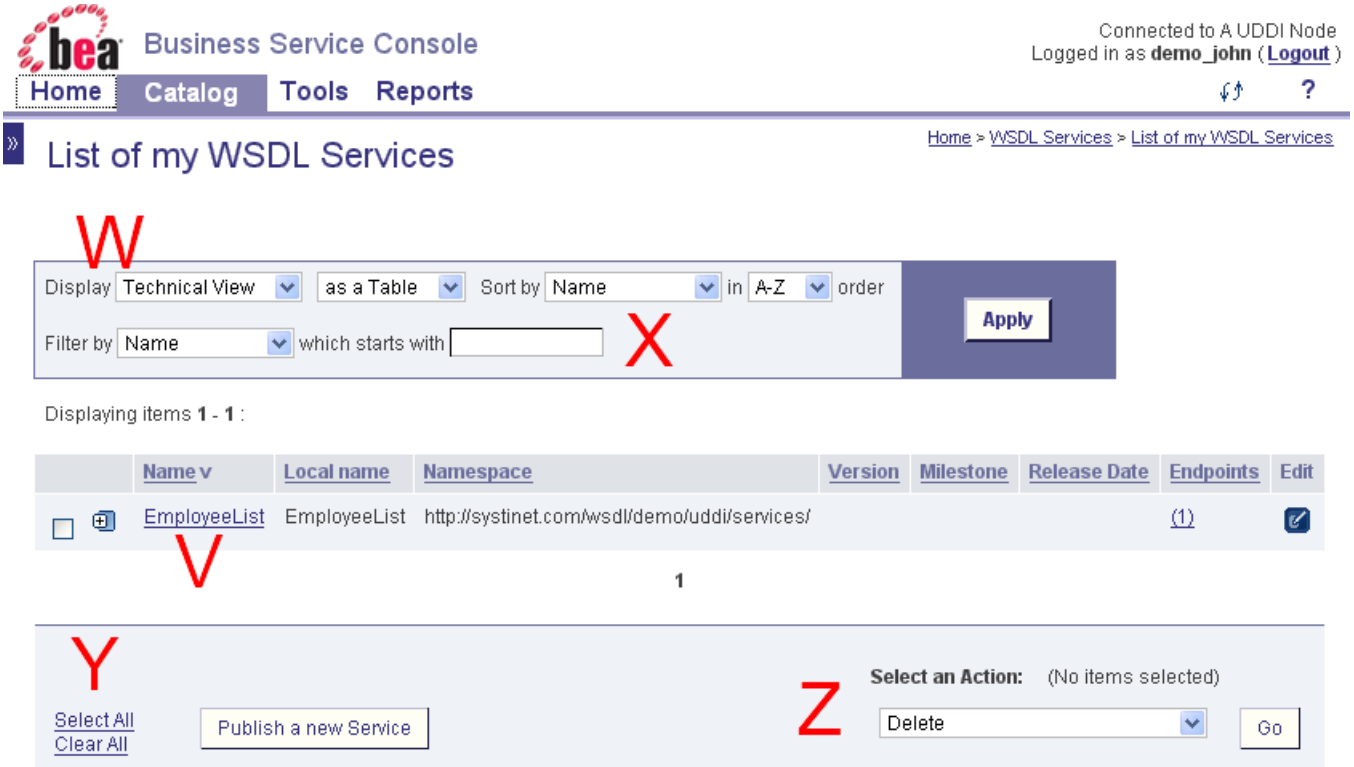
W: Result View Drop Down List This feature allows you to toggle among business, technical, and common views. Views differ in formatting and column selection.

X: Filters You can filter data you wish to display. To perform a filter, select a column name from the **Filter Column** drop down list, enter the **Filter value**, then click the **Filter** button. You can use wild card characters.

Y: Links for entity selection This section contains a set of links for selecting entities in the main display area. If you select all entities or clear (deselect) all entities displayed in the main display area will be selected. If the display area contains multiple pages, the **Select All** link will select entities in all pages.

Z: Action Drop Down List The action drop down list allows you to perform operations with selected entities. To perform the selected action, click the **Go** button.

Figure 3. Example Business Service Console page



4.2. User Account

Before you can publish data to the registry, you must have a BEA AquaLogic Service Registry account. Follow these steps to create a user account:

1. Click the **Create Account** link on the Business Service Console home page. This returns the **Create account** page shown in Figure 4.
2. Fill in all fields. Those labeled with an asterisk (*) are required. Your email address may be used later for enabling your account.
3. Switch to the **My profile** tab, shown in Figure 5 to specify profile preferences and subscription preferences.
4. When finished, click **Create Account**.

Note

BEA AquaLogic Service Registry may be configured to require email confirmation in order to enable the user account. In this case, the registry sends an email confirmation. Follow the instructions in this email to enable your account.

Figure 4. Create Account

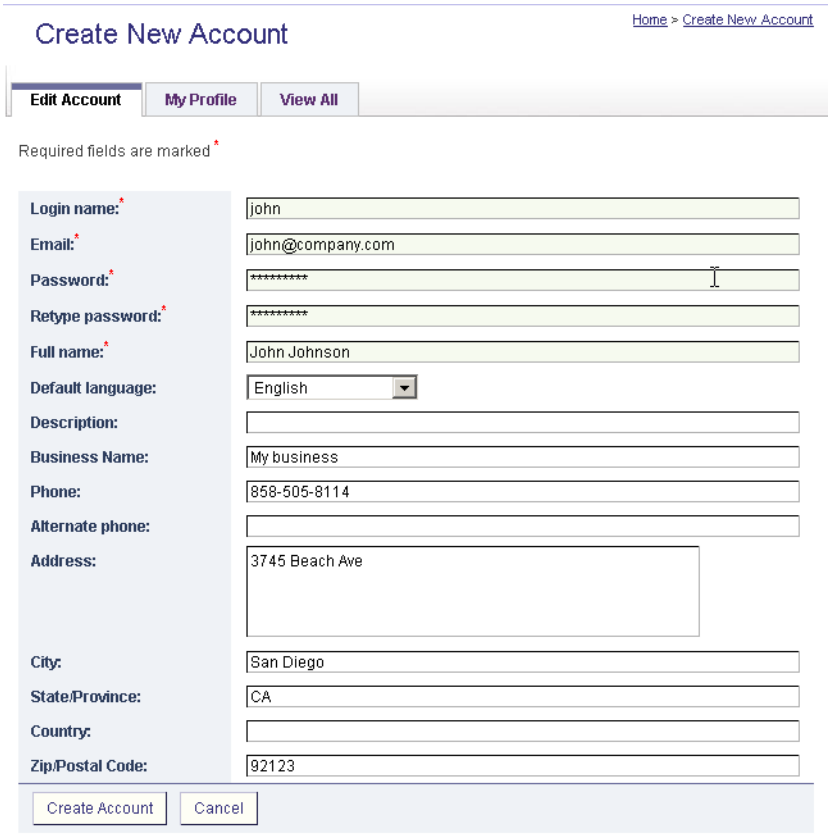
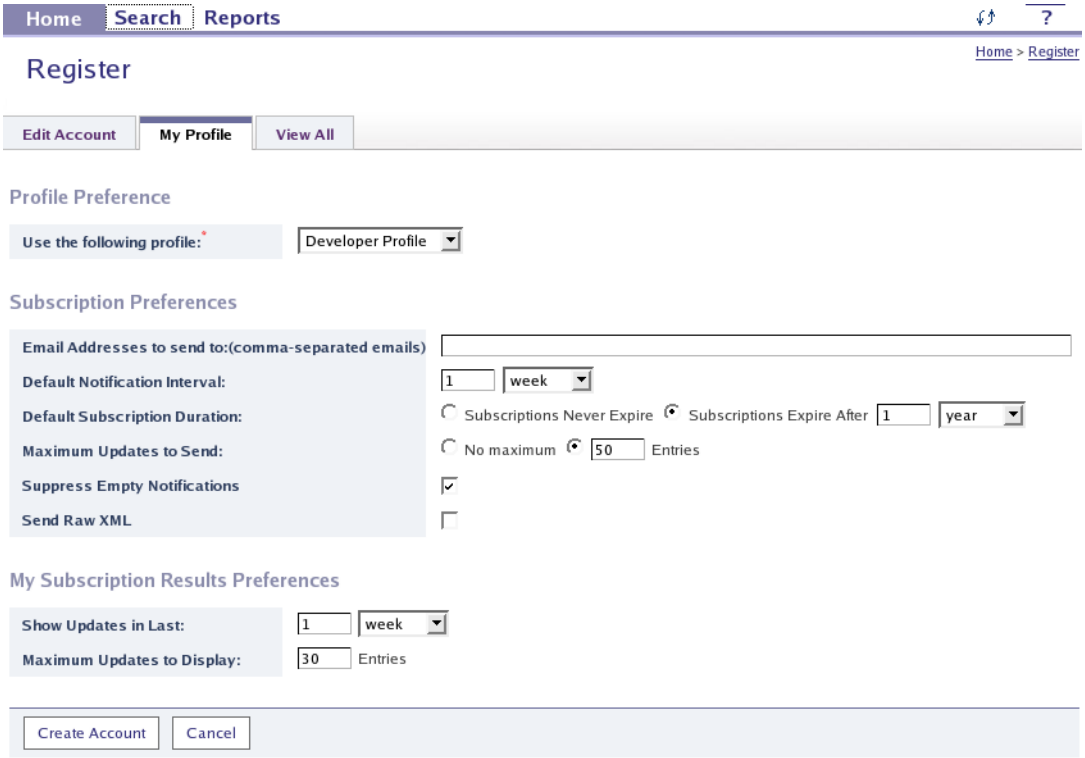


Figure 5. User Profile



4.2.1. User Profile Fields

The **My Profile** tab has the following fields:

- **Profile preference** - Select your preferred predefined user profile from this drop down list

  Note

  BEA AquaLogic Service Registry Administrator can disable selection of user profiles. In this case, a default user profile appears in a noneditable field.

- **E-mail addresses to send subscription notification** - You can enter a list of e-mail addresses to which email notifications will be sent. These addresses will be defaulted on the **Create subscription** page.
- **Default notification interval** - Specify how often email notifications will be sent.
- **Default subscription duration** - Enter the default subscription lifetime here.
- **Maximum Updates to Send** - Use this field to limit number of entries sent by an email notification.
- **Suppress Empty Notifications** - If checked, empty notifications will not be sent.
- **Send Raw XML** - If checked, email notifications will be sent in XML format.
- **Show Updates in Last** - If you want to view the updates made in the most recent period, specify the period here. For example, if you want to view updates made in the last three days, enter 3 in the first box and select **days** from the drop down list.

- **Maximum Updates to Display** - Enter how many items will be displayed .

4.2.2. Predefined User Profiles

BEA AquaLogic Service Registry contains a list of predefined user profiles which differ in which main menu tabs will be available to them. Each user profile also contains a definition of default formats for result views. The registry administrator can adjust these user profiles. See Section 3, Business Service Console Configuration.

The predefined user profiles are:

- **Business Expert** - Understands problems that needs to be solved and relationships and implications to other systems within the enterprise. The Business Expert proposes reusable functional components (future business services) and how these solve particular problems. This user associates both functional and non-functional requirements with the components. The Business Expert also suggests reuse of existing services.
  - Functional requirements are usually provided as descriptions attached to proposed components.
  - Non-functional requirements are usually represented with high-level capabilities and constraints that are rendered as categories (For example, secure, 24x7 uptime, transactional etc.).

- **Developer** - Implements business services according to description and associated capabilities/constraints (such as compliance). This user reuses low-level infrastructure services for the implementation.

  Business service implementation usually undergoes some QA and testing after development.

- **SOA Architect** - Re-factors input from the Business Expert. This user performs the following:
  - Translates Business Expert deliverables into a set of reusable business services.
  - Transforms high-level capabilities/constraints into standards-based capabilities/constraints that can be enforced and implemented by other roles (developers, administrators and operation managers).
  - Defines capabilities/constraints (such as compliance constraints) that enforce standards-compliance and common implementation and deployment service practices in the enterprise.
  - Enforces compliance to selected standards (SOAP, WSDL, UDDI, WS-S, WS-RM etc.).
  - Suggests reuse of existing business services.

- **Operator** - Deploys and manages business services implemented by the Developer into the production environment. This user also:
  - Publishes service endpoint and other runtime data about the deployed service.
  - Ensures that the business service is properly managed and secured by tagging the service with the appropriate category that triggers security and WSM registration processes.

- **SOA Administrator** - This user performs the same functions as the Operator, but has higher priviledges:
  - Publishes service endpoint and other runtime data about the deployed service.
  - Ensures that the business service is properly managed and secured by tagging the service with the appropriate category that triggers security and WSM registration processes.

- **Anonymous User Profile** - This profile applies to not authenticated users. The profile is a configuration placeholder for users that did not log in to the Business Service Console

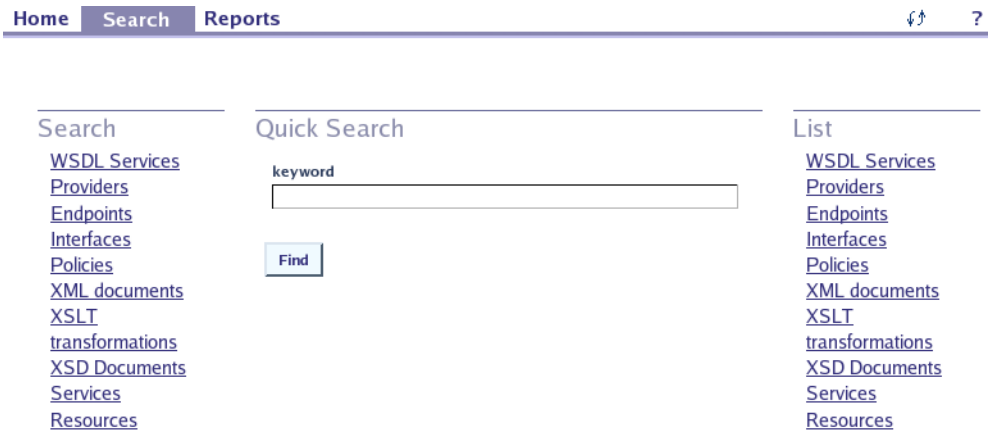
4.3. Searching

The Business Service Console allows you to search BEA AquaLogic Service Registry. You can search for providers, services, endpoints and interfaces. The tab also allows you to search for artifacts that have been published to BEA AquaLogic Service Registry.

Properties of search criteria are used in conjunction with one another. The search returns all records that satisfy any of the search criteria property values.

Searching functions are under the **Search** main menu tab.

Figure 6. Search Tab



We will explain how to search in the following examples:

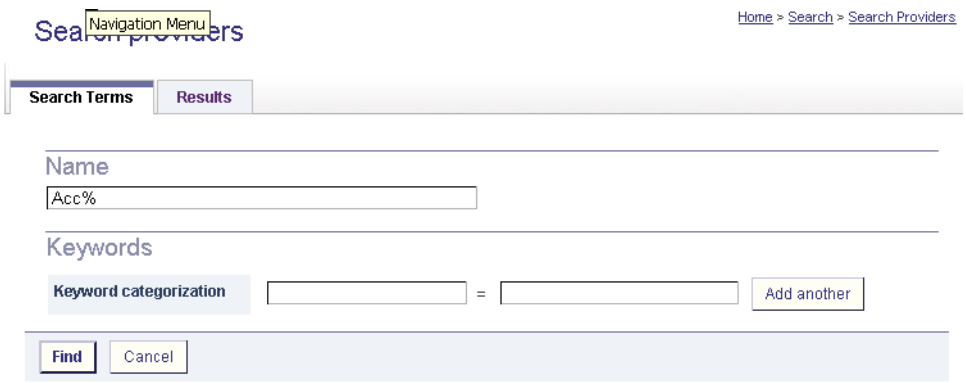
- Searching for providers
- Searching for endpoints

4.3.1. Searching Providers

To search for providers:

1. On the **Home** main menu tab select the **Search providers** link in the right display area. The page shown in Figure 7 appears.

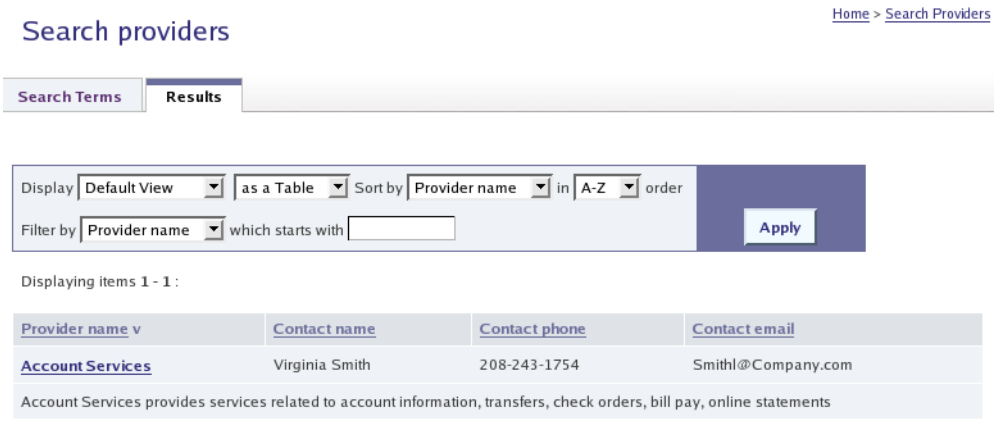
Figure 7. Searching Providers



Enter search criteria. You can enter wild card characters. Then click **Find**.

2. Search results will be displayed on the page shown in Figure 8.

Figure 8. Searching Providers - Result



In Figure 8, you can also switch result views using the **Display** drop down list. The default result view is configurable for each user profile. See Section 3, Business Service Console Configuration for more information.

If the result view contains too many records, you can filter which records will be displayed as follows:

1. Select the **Filter by** on which you wish to apply the filter.

2. Enter the filter string in the Filter value edit box. Wildcards can be used. The "%" character is replaced by any number of characters. The "_" character is replaced by any single character. The end of the string is treated as if it has a "%" wildcard suffix so there is no need to add a terminating wildcard.

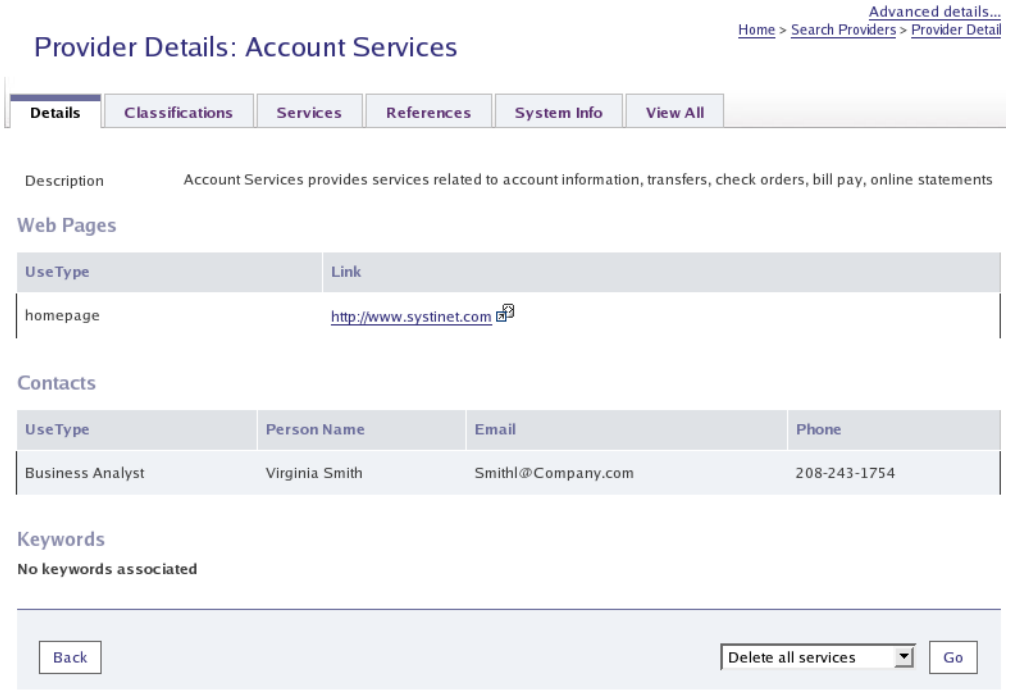
3. Click the **Apply** button. The view is updated with only those records matching the filter.

The result view table can view sorted by each column. To sort, just click on the appropriate column header.

Large result lists are divided into multiple pages. The number of records per page can be configured by administrator. See Section 3.4, Paging Limits for details.

If you click on the provider's name, provider details will displayed as shown in Figure 9.

Figure 9. Searching Providers - Provider Detail



See Section 4.6.1, Entity Details for more information.

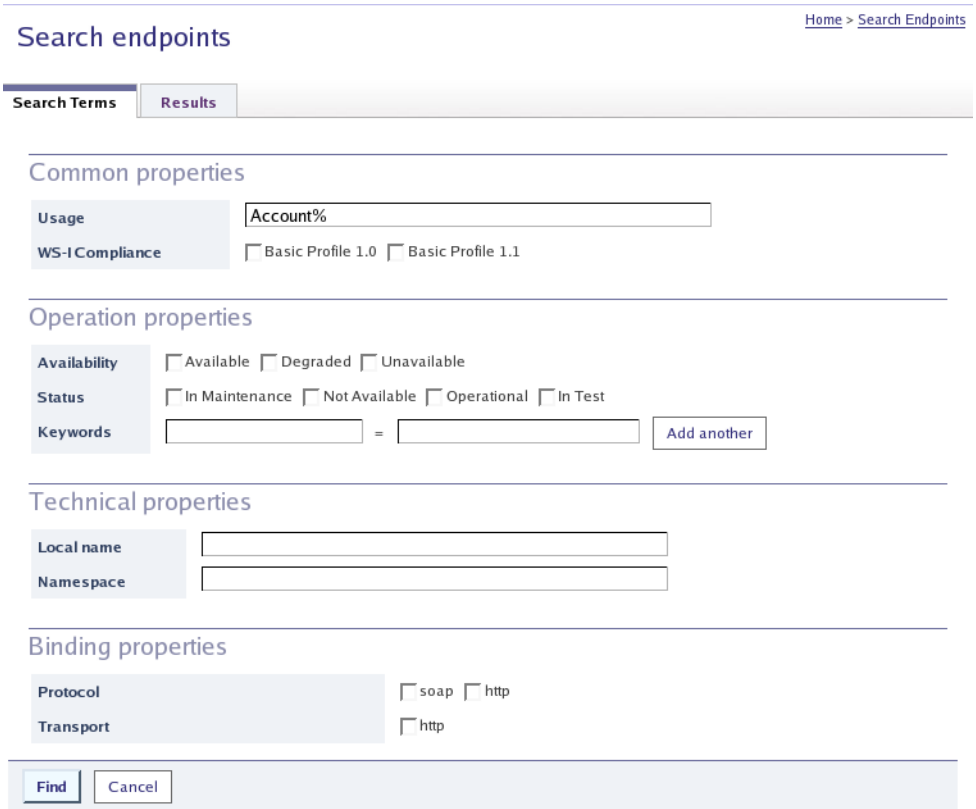
If you access a detail screen from the result view under the **Catalog** tab, the entity can be edited or deleted. You can also request approval of the entity (on a publication registry) or create a subscription.

4.3.2. Searching Endpoints

To search for service endpoints:

1. On the **Home** main menu tab select the **Search endpoints** link in the right display area. The page in Figure 10 is displayed.

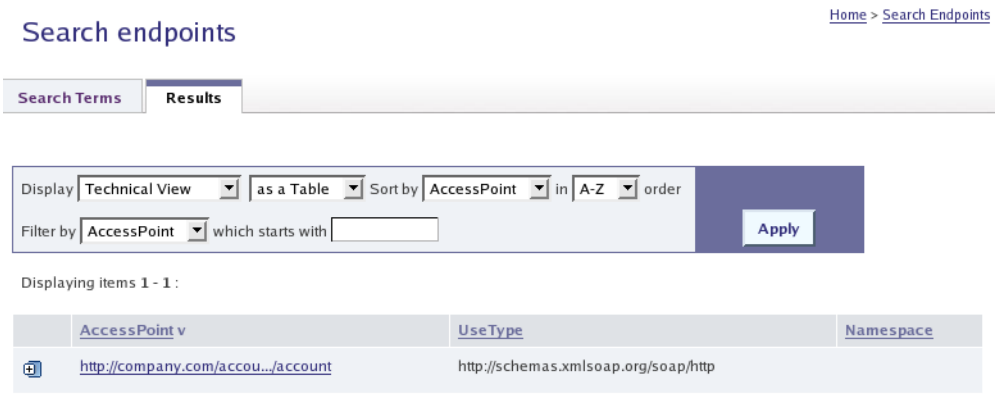
Figure 10. Searching Endpoints



Enter your search criteria. You can enter wild card characters. Then click **Find**.

2. The search results will be displayed on the page shown in Figure 11.

Figure 11. Searching Endpoints - Result



To display complete information about an endpoint, click on the endpoint URL in the result view. Endpoint details will be displayed. See Section 4.6.1, Entity Details for more information.

4.4. Publishing

Under the **Catalog** main menu tab, you can use publishing wizards to publish data to BEA AquaLogic Service Registry.

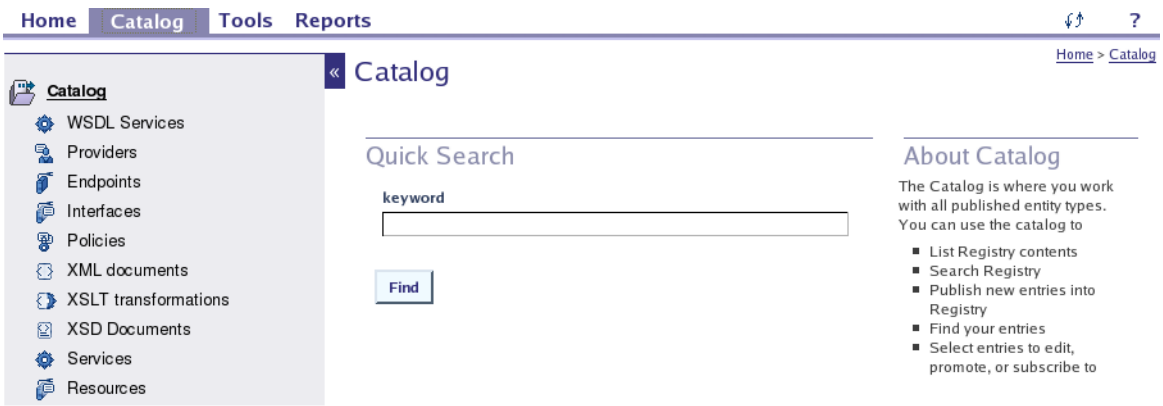
Note

You must be logged in to publish data to BEA AquaLogic Service Registry. See Section 4.2, User Account to learn how to register your user account.

Tip

To try publishing wizards, you can use the demo data account with the username `demo_john` and password `demo_john`.

Figure 12. Catalog Tree



You can publish the following data to BEA AquaLogic Service Registry:

- **Providers** - A two-step publishing wizard allows you to enter provider's name and description, provider's taxonomy classification and contact persons.
- **Services** - A four-step publishing wizard guides you through publishing a service, its interfaces, and its endpoints.
- **Interfaces** - A wizard for publishing and republishing service interfaces.
- **Resources** - This node allows you to start publishing wizards for publishing WSDL files, XML files, XML schema, and XSL transformations.

We will demonstrate publishing wizards in the following examples:

- Publishing providers
- Publishing services

4.4.1. Publishing Providers

In this section we show, step by step, how to publish a provider. We will create the provider HR Services. To publish this provider:

1. **Login** to BEA AquaLogic Service Registry using the link under the **Home** main menu tab.

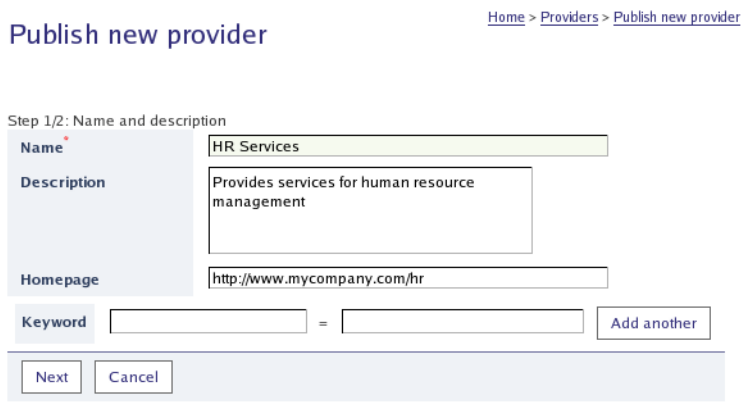
2. Click on the **Catalog** main menu tab. Click on the **Providers** link in the **Catalog** tree. Then, click on the **Publish a new provider** link in the right-hand display area.

Note

If you do not see the **Catalog** main menu tab, log in with username `demo_john` and password `demo_john` in order to follow this example.

3. The page shown in Figure 13 appears.

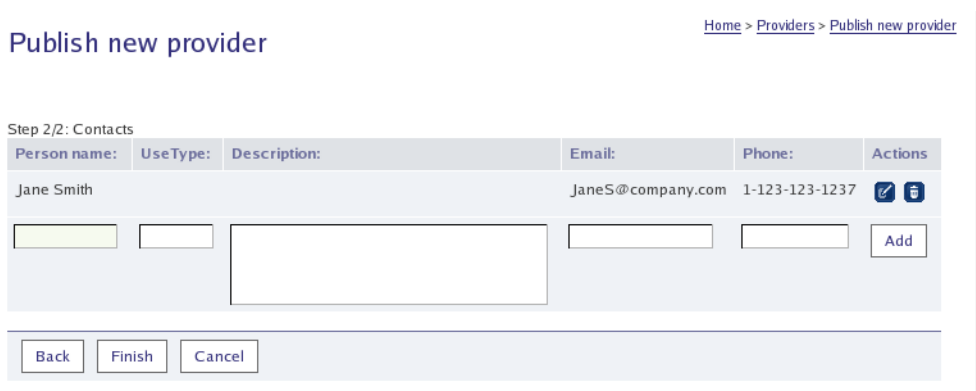
Figure 13. Publish Provider - Step 1



4. Enter the provider name and description. You can also enter the home page URL of the provider. Click **Next**.

5. The page shown in Figure 14 appears.

Figure 14. Publish Provider - Step 2

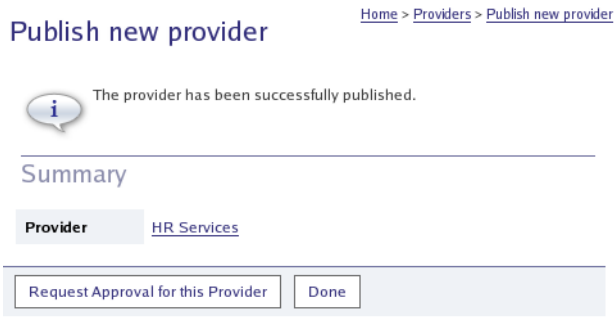


6. Enter the contact's data, and click **Add**. This returns a list of contacts you have entered and a blank New Contact form. Click **Finish** when you have entered all of your contacts.

   The person's name is only a required field when you enter any contact information. It is possible to create a provider without a contact.

7. On a publication registry, you then have the opportunity to request approval for the new provider as shown in Figure 15.

Figure 15. Publish Provider - Approval Step



For more information see Section 4.8.1, Requestor's Actions.

4.4.2. Publishing Services

In this section, we will show you how to publish a business service step-by-step. The service will be created from a WSDL file accessible from the registry server. Note that it is also possible to publish a service without a WSDL, in which case some additional details must be entered.

The following locations are supported for the WSDL and documents it imports:

- the server filesystem, perhaps on a network drive shared with user workstations;
- an HTTP server, optionally:
  - requiring HTTP Basic authentication;
  - using SSL (https);

If BEA AquaLogic Service Registry receives the response `401-Unauthorized` when attempting to retrieve the WSDL or a (direct or indirect) import, you will be prompted for HTTP Basic authentication credentials (a login name and password). If necessary these will be used to retrieve subsequent imports. This assumes that the server for each import requires the same credentials or none at all.

Note

BEA AquaLogic Service Registry will always attempt to retrieve imported documents without credentials first and will only try sending credentials if this results in a `401-Unauthorized` response. A potential security issue is that a third-party server may be intentionally configured to return the `401-Unauthorized` response to gain knownledge of credentials from BEA AquaLogic Service Registry.

Tip

In an SOA it is desirable for such documents to be widely accessible without unnecessary security constraints. Furthermore, once published to the registry, the documents will be accessible without the same credentials. The security policies governing the registry and servers from which WSDL documents and imports are retrieved, must take these issues of trust into account.

To publish a business service:

1. **Login** to BEA AquaLogic Service Registry using the link under the **Home** main menu tab.

2. Click on the **Catalog** main menu tab. Click on the **WSDL Services** link in the **Catalog** tree. Then, click **Publish a new service** in the right-hand display area.

   ## Note

   If you do not see the **Catalog** main menu tab, log in with username `demo_john` and password `demo_john`.

3. The page appears as in Figure 16:

   ### Figure 16. Publish Services - Step 1

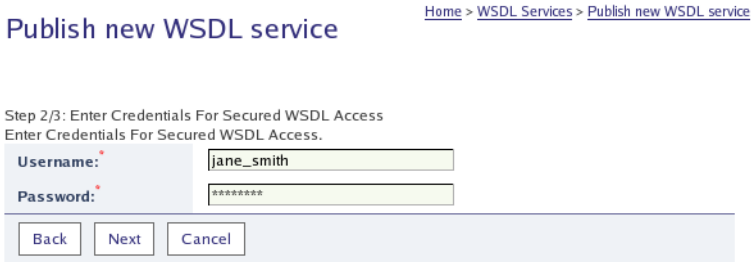
4. From the **Provider** drop down list, select a provider. Which providers are listed depends on the user's permissions. The user must have permission to write to the provider. You can use the provider created in the previous section.

   Enter the location of the WSDL file. You can use the WSDL in the demo data located in the `REGISTRY_HOME/demos/conf/employeeList.wsdl`. You need to prefix the path with `file://` in that case. For example, under windows the path might be `file:///c:/systinet/registry/demos/conf/employeeList.wsdl`.

   Click **Next**.

5. If HTTP Basic authentication is required to access the WSDL then you will be presented with the screen shown in Figure 17.

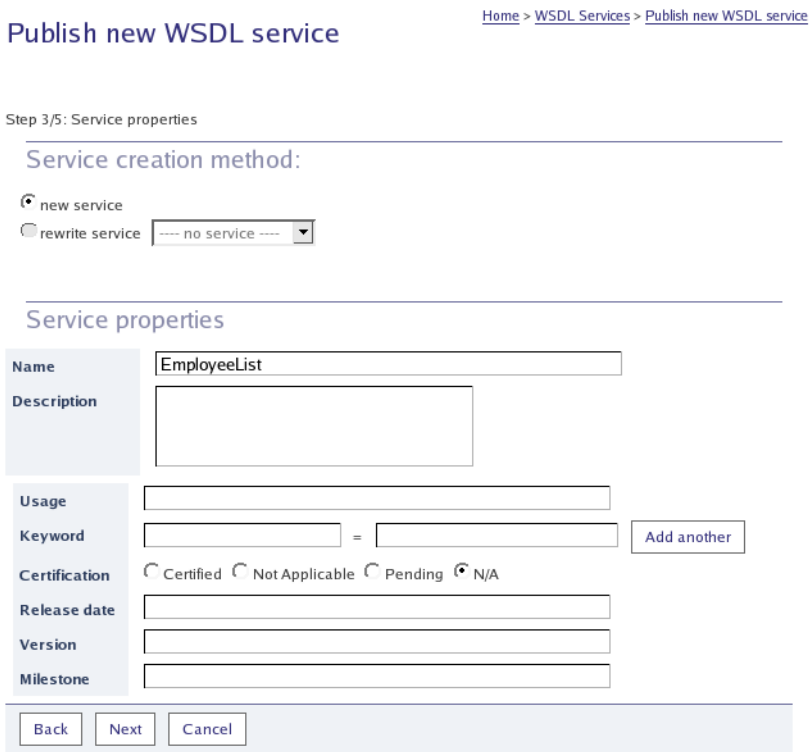
   ### Figure 17. Entering HTTP Basic credentials

   Enter credentials and click **Next**.

6. The page shown in Figure 18 will appear.

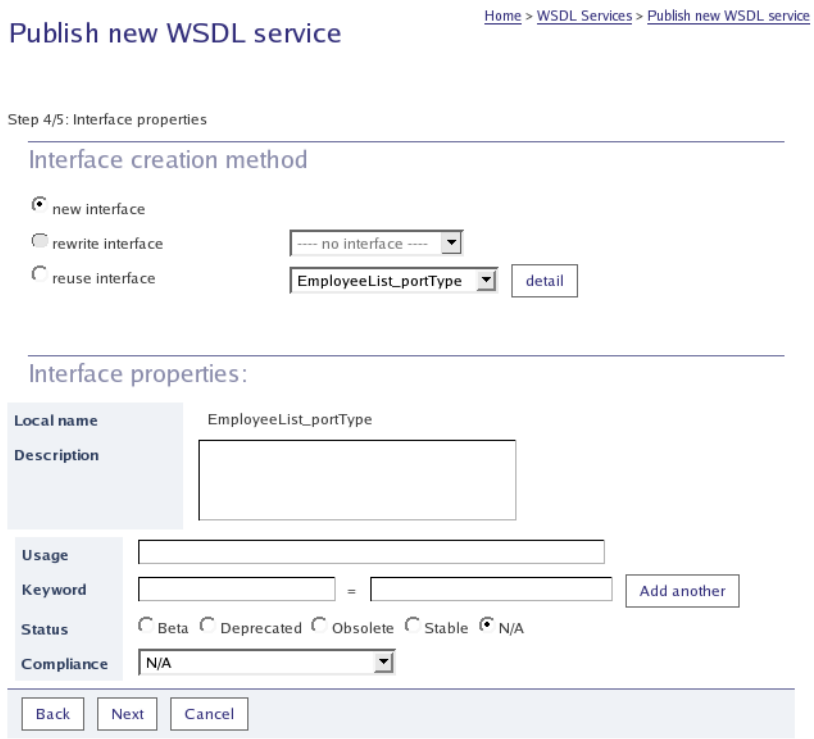
Figure 18. Publish Service - Service Properties



7. You can optionally specify service properties. The service **Usage** will classify the service by functional areas. You can enter the service certification status, release date, version and milestone. Then click **Next** .

8. The next step allows you to specify service interface properties. You can specify the interface status and compliance.

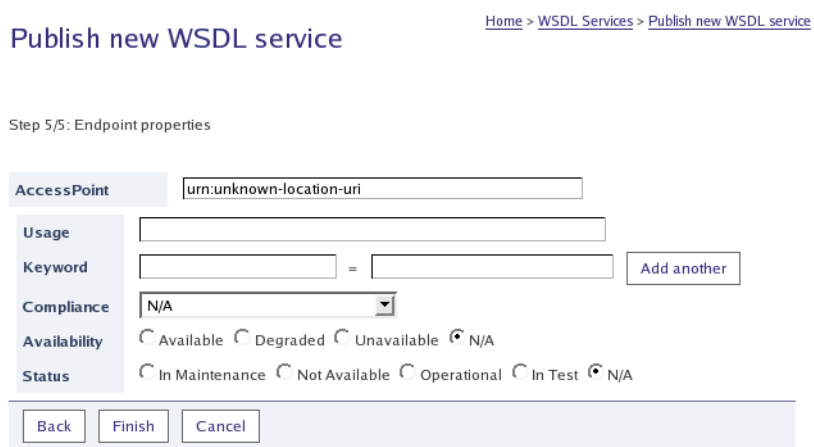
Figure 19. Publish Service - Interface Properties



Then click **Next**.

9. The last step of the wizard allows you to specify service endpoint properties.

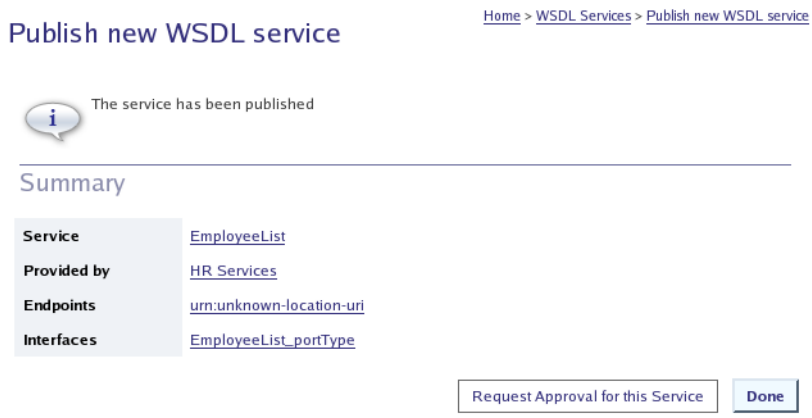
Figure 20. Publish Service - Endpoint Properties



Then click **Finish**.

10. A summary of how the service has been published to BEA AquaLogic Service Registry will appear, as shown in Figure 21.

Figure 21. Publish Service - Summary

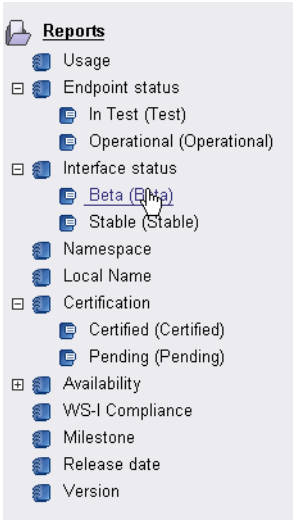


On a publication registry, you then have the opportunity to request approval for the new service as shown in Figure 21. For more information see Section 4.8.1, Requestor's Actions.

4.5. Reports

Under the **Reports** main menu tab you can browse various reports. In the reports tree shown in Figure 22 you can select a report which will be shown in the right display area. Most of the reports can be displayed in different views. The Business Service Console contains the predefined reports shown in Figure 22. If you see different reports in the tree, they have been reconfigured (Browsable Classification) by an administrator.

Figure 22. Reports Tree



The Business Service Console includes the following predefined reports:

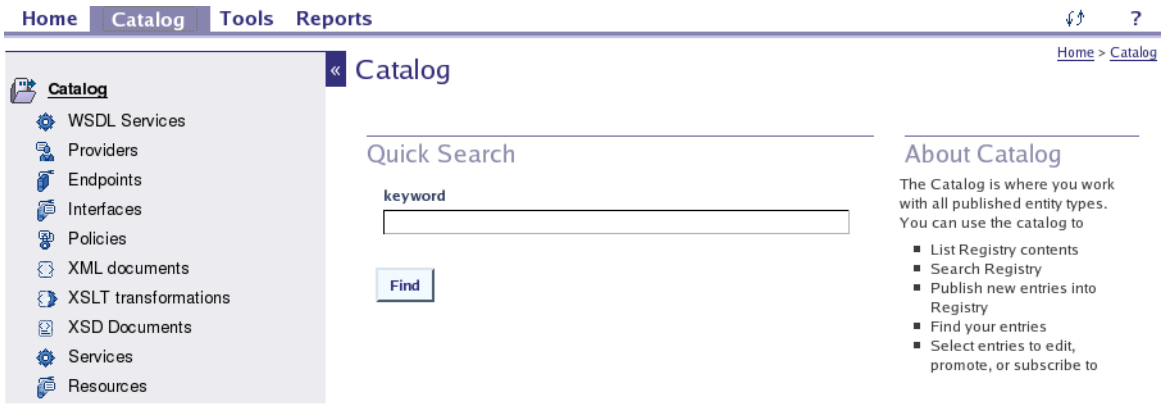
- **Usage** - This report shows services, endpoints, and interfaces categorized by the `systinet-com:taxonomy:usage` taxonomy.
- **Endpoint status** - This report shows endpoints categorized by the `systinet-com:taxonomy:endpoint:status` taxonomy.

- **Interface status** - This report shows interfaces categorized by the `systinet-com:taxonomy:interface:status` taxonomy.
- **Namespace** - This report shows services, endpoints, interfaces, and resources categorized by the `uddi-org:xml:namespace` taxonomy.
- **Local Name** - This report shows services and endpoints categorized by the `uddi-org:xml:localName` taxonomy.
- **Certification** - This report shows services categorized by the `systinet-com:taxonomy:service:certification` taxonomy.
- **Availability** - This report shows endpoints categorized by the `systinet-com:taxonomy:endpoint:availability` taxonomy.
- **WS-I Compliance** - This report shows endpoints and interfaces categorized by the `ws-i-org:conformsTo:2002_12` taxonomy.
- **Milestone** - This report shows services categorized by the `systinet-com:versioning:milestone` taxonomy.
- **Release date** - This report shows services categorized by the `systinet-com:versioning:releaseDate` taxonomy.
- **Version** - This report shows services categorized by the `systinet-com:versioning:version` taxonomy.

4.6. Entities

The preceding sections describe how to navigate to entities by Section 4.3, Searching or with reports. The **Catalog** tab provides a data-centric approach. It lists types of entity and allows the user to select a type before performing an action. One way to perform actions on an entity type is to bring up the Context Menu by right-clicking on an entity type.

Figure 23. Catalog tab

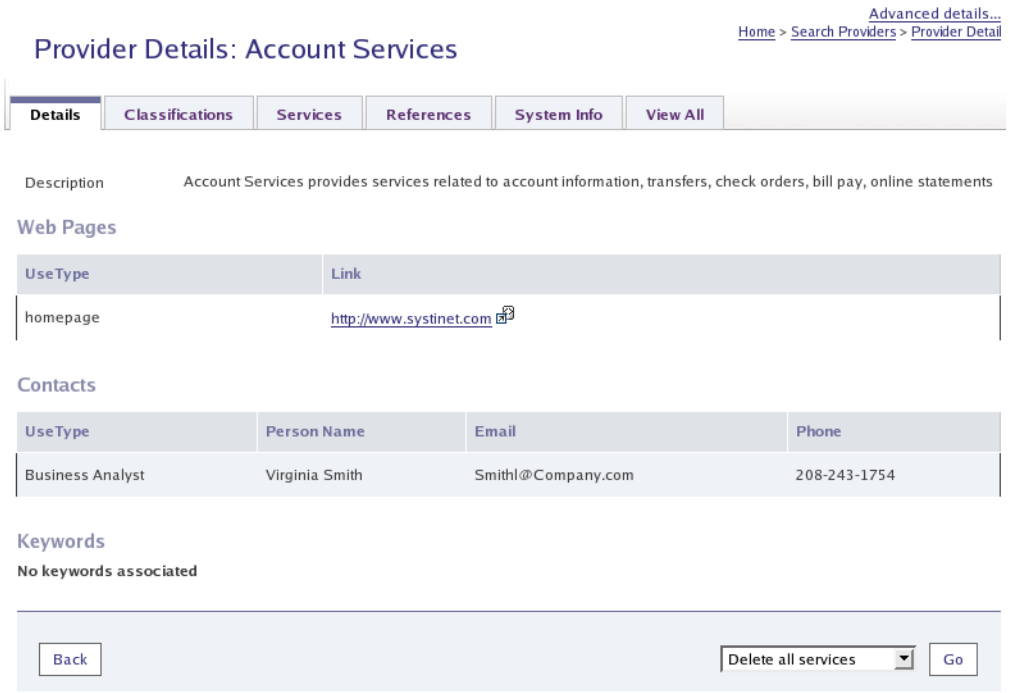


This section focuses on the entity types listed in the catalog and how they are displayed by the Business Service Console.

4.6.1. Entity Details

References to entities and resources on the Business Service Console are generally hyperlinks, allowing you to navigate to them by various routes. Clicking such a link displays a details page. For example, in Section 4.3.1, Searching Providers the example resulted in the page shown in Figure 24.

Figure 24. Provider Details



Some of the tabs are specific to the entity type. For example, **Services** in the above example. This section focuses on general purpose tabs.

Table 2. General Purpose Detail Tabs

Label	Description
Details	Basic details relating to the entity, depending on its type.
Classifications	How this entity is classified using taxonomies.
References	References to related entities. Note that there is also a **Referenced by** action to list other entities that refer to the entity.
System Info	Information relating to storage of the entity in BEA AquaLogic Service Registry, including ownership, creation and modification dates and UDDI keys that uniquely identify it.
View All	This tab displays all the information on the other tables on a single screen.

Note that the tabs displayed and their content depend on:

- the user's profile. See Section 4.2, User Account;
- customization of the Business Service Console by administrators. See Section 3, Business Service Console Configuration;

4.6.2. Resources

Resources are essentially entities that are documents, identified by a URL. Together with generic features such as classifications and references, resources are the means by which BEA AquaLogic Service Registry supports arbitrary document types. BEA AquaLogic Service Registry provides special support for the following types of resource.

Table 3. Special Resources

Type	Description
XML Documents	eXtensible markup language documents
XSLT Transformations	XML Stylesheet Language Transformations specifying how an XML document can be transformed into another document, typically also an XML document.
XSD Documents	XML Schema Document, specifying a particular type of XML document.
Policies	WS-Policy documents that can be attached to other entities to specify: • conformance constraints on entities implementing SOA governance policies; • constraints on how a client may use a service, to facilitate establishment of a contract between a provider and a service user; Policies attached to entities are visible as references.

Note that all of the above are XML documents. Furthermore, there is a generic type on the **Catalog** tab with label **Resource**. This enables all types of resource, including the above, to be processed using the flexible generic features of BEA AquaLogic Service Registry.

Note

Resources are represented as UDDI tModels. This representation is visible on the Registry Console.

4.7. Subscription and Notification

Subscriptions are used to alert interested users in changes to structures made in BEA AquaLogic Service Registry. The Business Service Console allows you to create and manage subscriptions for monitoring new, changed, and deleted entities. The following entities can be monitored: providers, services, interfaces, and endpoints, as well as resources (WSDL, XML, XSD and XSLT). You can establish a subscription based on a set of entities in which you are interested or on a specific search query. Users can receive notifications about modified structures via email messages or they may view the modified entities under the **Tools** main menu tab in the **My Subscription Results** section.

Note

If you wish to create more advanced subscriptions, see Advanced Topics, Section Publishing Subscriptions.

In this chapter, we will show you on demo data the following actions:

- Creating Subscriptions on Selected Entities
- Creating Subscriptions from Search Query
- Managing Subscriptions
- Viewing Changed Entities

4.7.1. Subscription On Selected Entities

In this section we will show you how to create subscriptions on selected entities. The following steps guide you to create a subscription on the HR provider from demo data. You will then be notified about each modification made to the HR provider, and modifications made to all of its child entities: services, interfaces, endpoints etc.

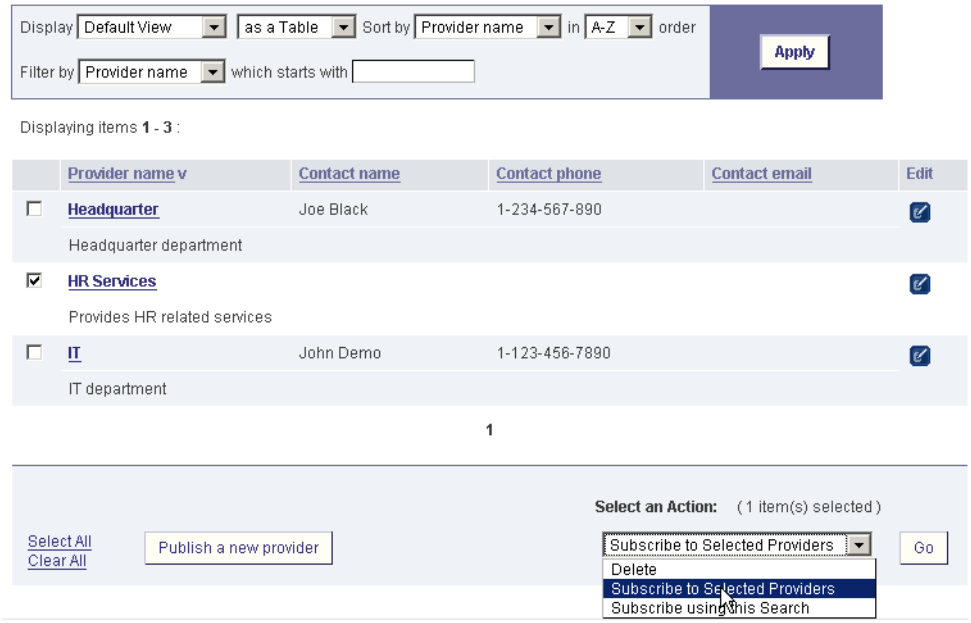
1. Under the **Catalog** main menu tab, click on the **Providers** branch in the tree menu. Then click on the link, **List all providers**.

2. Locate the HR provider and toggle the check box in front of the provider's name.

   If the list contains multiple pages, you can navigate between pages and select entities on multiple pages.

3. From the drop down list labeled **Select an Action**, located at the bottom of the page, select **Subscribe to Selected Providers** as shown in Figure 25.

Figure 25. Subscription From Providers List

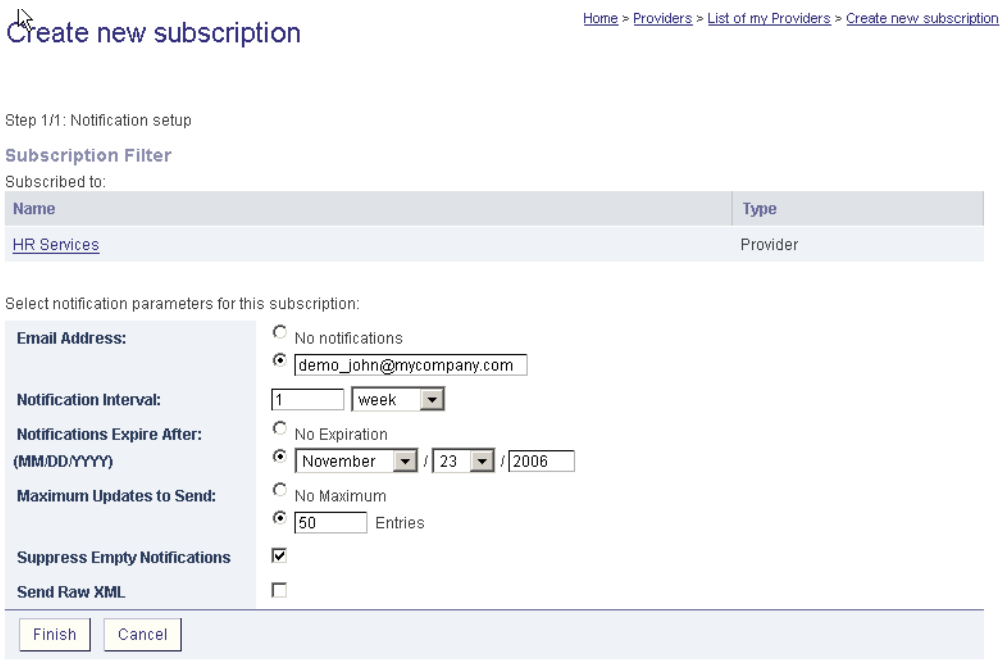


4. Click **Go** to start the subscription wizard. The page shown in Figure 26 will appear.

Note

You can also create a subscription from an entity detail page.

Figure 26. Create Subscription



5. The subscription filter contains a list of the entities you have selected on the previous screen. You can specify an email address to which notification messages will be sent. If do not want to receive email notifications, select the option **No notifications**. Configure the frequency of mail notifications using the drop down lists labeled **Notification Interval**.

   You can specify the default email address and notification interval values in your profile.

6. Enter additional information on this panel. The default values are entered in your profile. Click **Finish** when done.

7. You can review your subscriptions under the **Tools** main menu tab, section **Manage My Subscriptions**. The page shown in Figure 29 will appear.

4.7.2. Subscription from Search Query

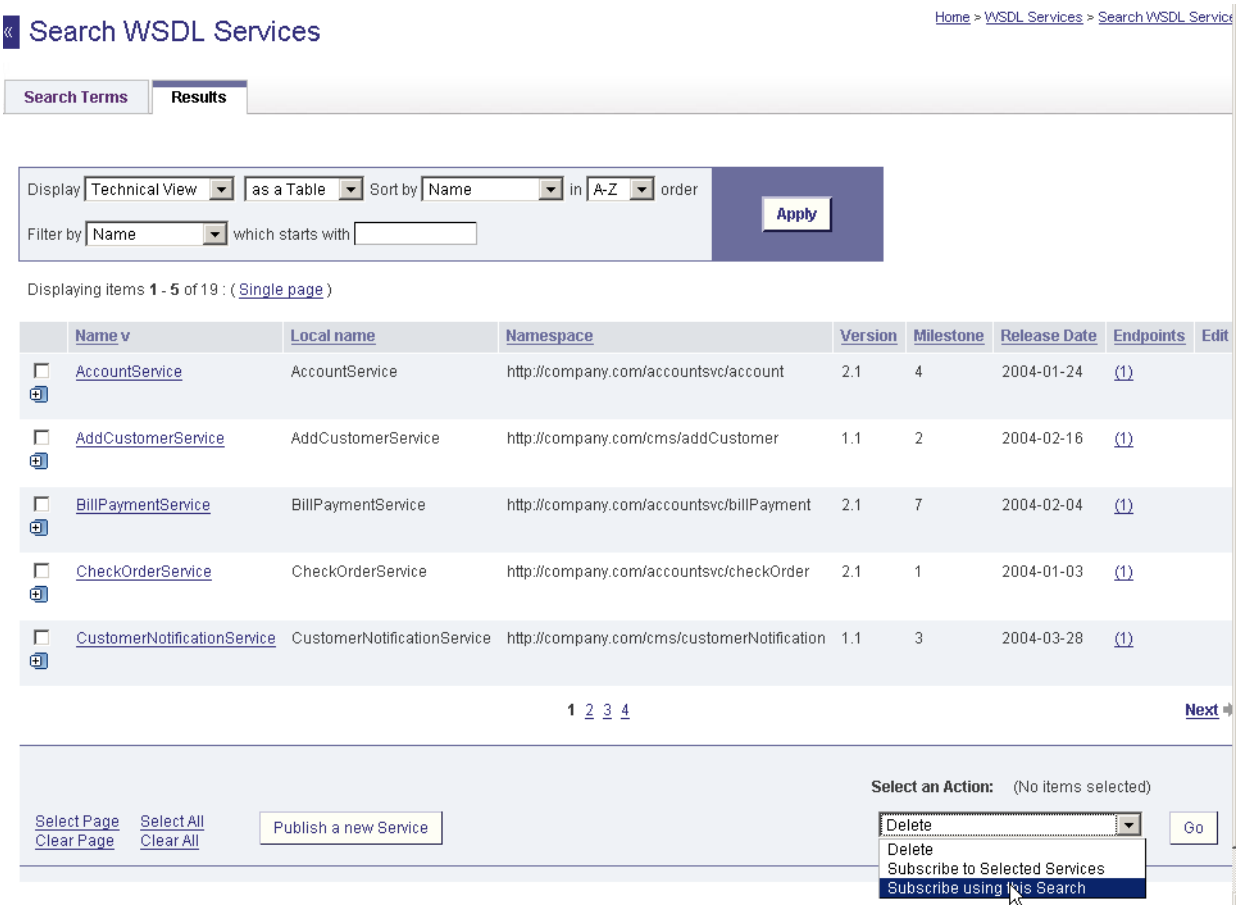
In this section, we will show you how to create a subscription based on a search query. Our subscription will monitor all certified services, even newly created certified services.

1. Under the **Catalog** main menu tab, click on the **Services** branch in the tree menu. Then click on the link **Search services**.

   On the **Search services** page, check the certification status **certified** located under **Business properties**, and click **Find**.

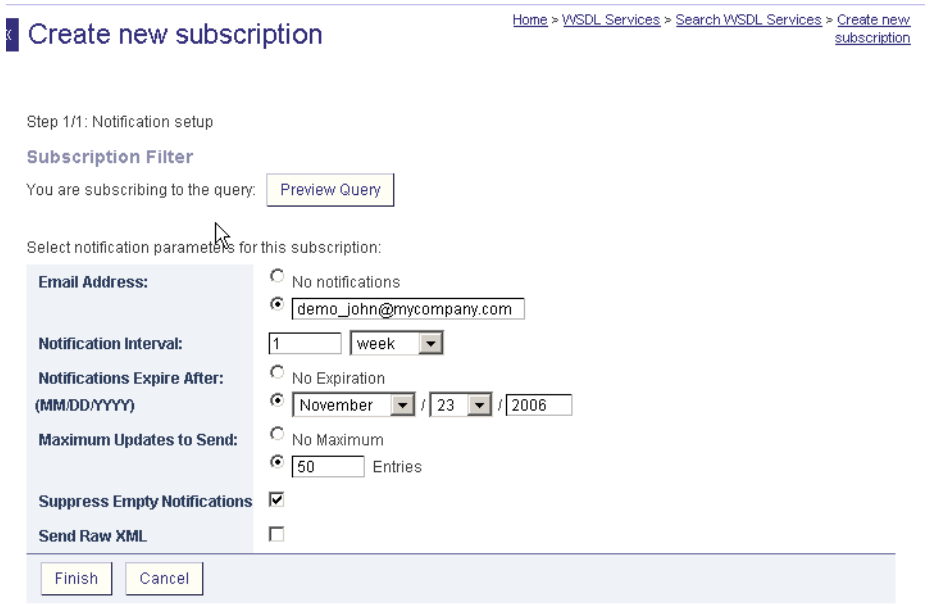
2. The page shown in Figure 27 contain a list of all certified services.

Figure 27. Subscription From Services Search



3. From the drop down list labeled **Select an Action**, located in the bottom of the page, select **Subscribe using this Search** as shown in Figure 27. The page shown in Figure 28 will appear.

Figure 28. Create Subscription



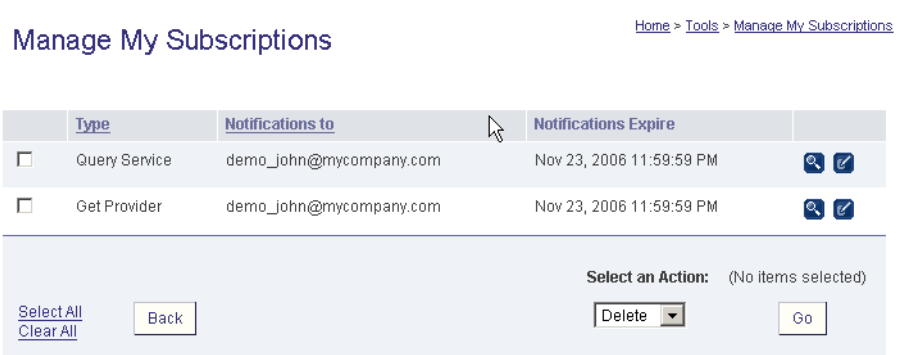
4. The subscription filter contains the search query. You can execute the query to review the query specification. It is not possible to modify the query, so if you wish to change the query, click **Cancel** button and recreate the steps above.

5. To review your subscriptions, select the **Tools** main menu tab, and click on **Manage My Subscriptions**. The page shown in Figure 29 will appear.

4.7.3. Manage Subscriptions

You can manage your subscription when you click on **Manage My Subscriptions** under the **Tools** main menu tab. On the **Manage my subscription** page shown in Figure 29, you can edit, delete or view subscription detail information.

Figure 29. Manage Subscriptions

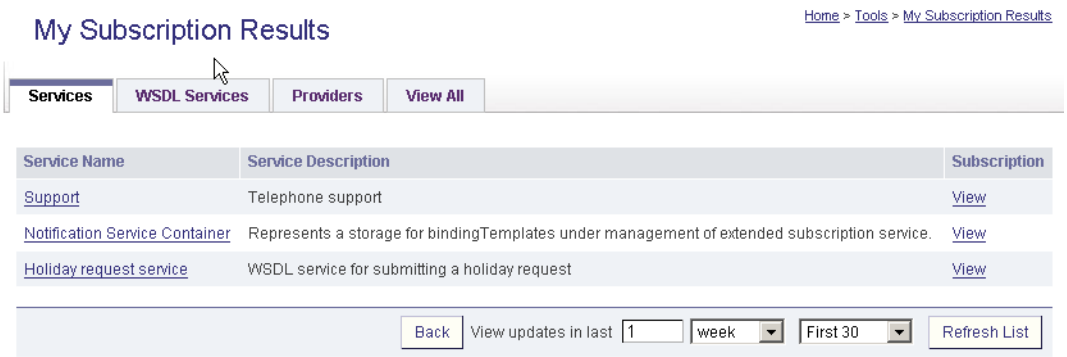


4.7.4. View Changed Entities

There are two options for viewing changed entities. If you have specified an email address during subscription creation, notification will be sent to you by email. The other option is to review changes under the **Tools** main menu tab. Click on

the **Providers** link. If the HR provider has been modified and you created the subscription described in this chapter, you will see the page shown in Figure 30

Figure 30. View Changes



4.8. Approval Process

The approval process includes two types of users:

- **requestor** - A user of the *publication* registry who can ask for the approval of data for promotion. Every user can ask for approval, but to have data considered for promotion, a user must have an administrator-assigned approver.
- An **approver** is a user or group given the ability to review published information on the *publication* registry and grant or deny approval to promote that information to the *discovery* registry. If the approver is a group then any of its members may approve or reject approval requests.

Note

We recommend reading Section 1.5, Approval Process in BEA AquaLogic Service Registry to become familiar with approval process.

In this chapter, we will describe:

- Requestor's actions
  - Create and submit request
  - Manage Requests
  - Cloning Requests
- Approver's actions
  - Approve/Reject request
  - View Approval History

4.8.1. Requestor's Actions

- Create and submit request

- Manage Requests
- Cloning Requests

Create and Submit Request

This section describes the steps to request approval of entities. This can be done in two ways:

1. When an entity is published on a publication registry, the final screen provides a button to request immediate approval of the entity and related entities for promotion to the discovery registry. See Section 4.4, Publishing;

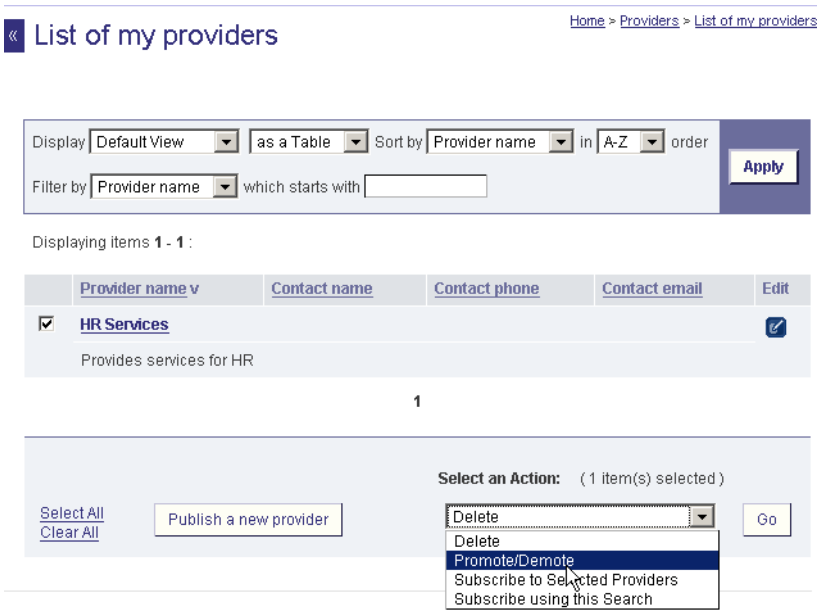
2. For published entities it is possible to request their promotion to the discovery registry or demotion from the discovery registry;

The procedure below uses the second of these as an example. The first case differs in that it is not possible to demote newly published entities and so the user is not presented with this option. The procedure has minor differences in the first few steps.

You need to publish entities before following this procedure. The first few steps request promotion of an existing provider as an example. The entities are those in sections Section 4.4.1, Publishing Providers and Section 4.4.2, Publishing Services. These sections also explain the other way of starting the procedure.

1. On the **Catalog** tab, click on an entity type such as **Providers** in the tree menu. To select an existing provider click **My providers** and the existing providers are displayed as in Figure 31.

Figure 31. Select Items for Promotion



2. Toggle the check box in front of the provider's name, select **Promote** from the action drop down list located in the bottom of the page, and then click **Go**. A page appears like that shown in Figure 32.

Figure 32. Add Items to Approval Request

3. You can see which entities will be added to the approval request including the requested entity and entities related to it. If the entity previously existed then you can specify whether you are requesting promotion to the discovery registry or demotion (deletion) from the discovery registry. You are not given this choice if you are requesting approval immediately following publication.

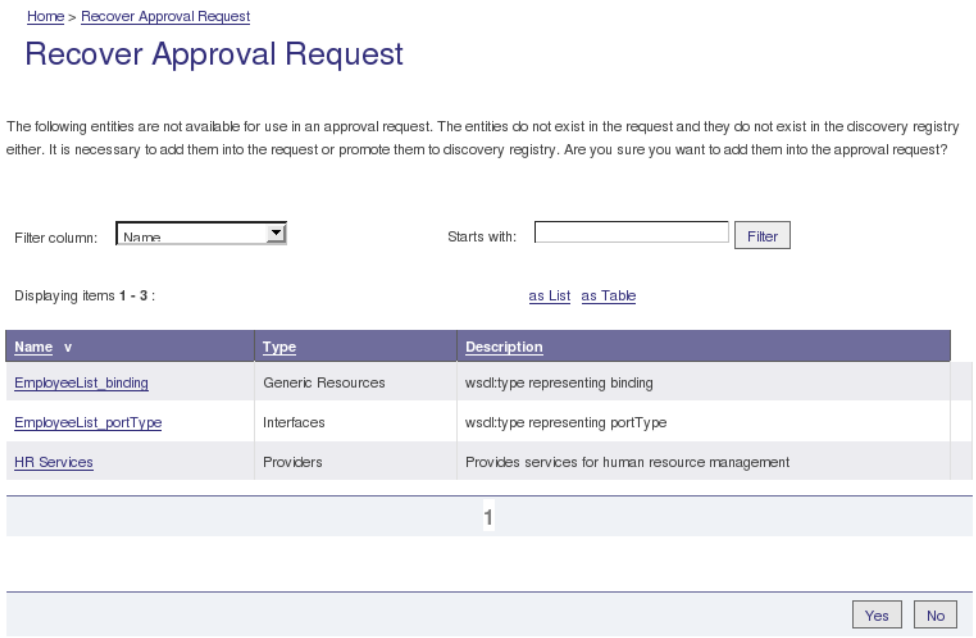
   In this example, an attempt to promote some entities and demote others would probably fail because they are related. However, you can select more than one entity in the catalog (Figure 31), promote entites related to some of them and demote the rest.

4. Further down you can choose to add the entities to a new approval request, in which case you must enter a name. Alternatively, if there are other pending requests, you can click **Add to existing request** and select one from the drop-down list.

5. Finally you can choose to **Submit for immediate approval**, in which case you can enter a message for the approver. Alternatively you can save the request. Then click **OK**.

6. If you have submitted the request for immediate approval, automatic context checking is performed. If it fails you may be presented with a page like Figure 33:

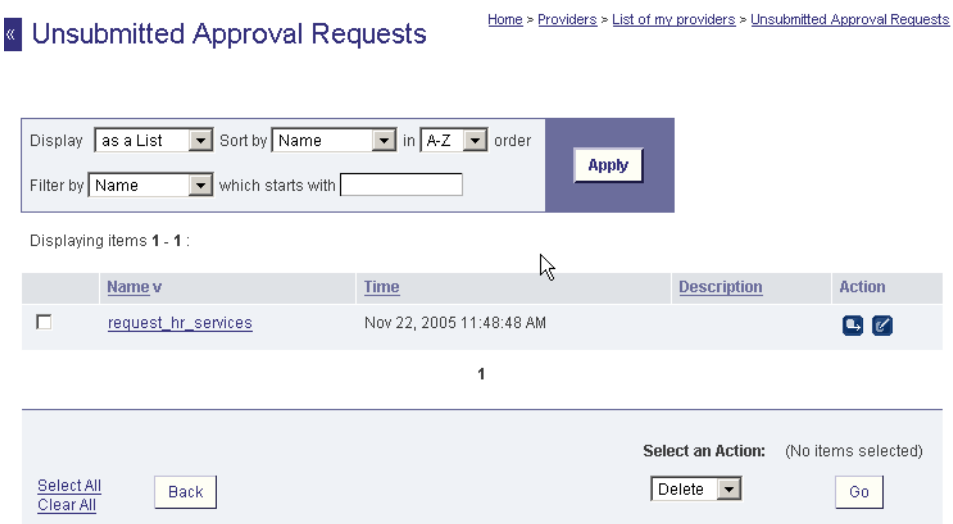
Figure 33. Recover Approval Request



You can then choose whether to recover by adding the suggested entities to the request.

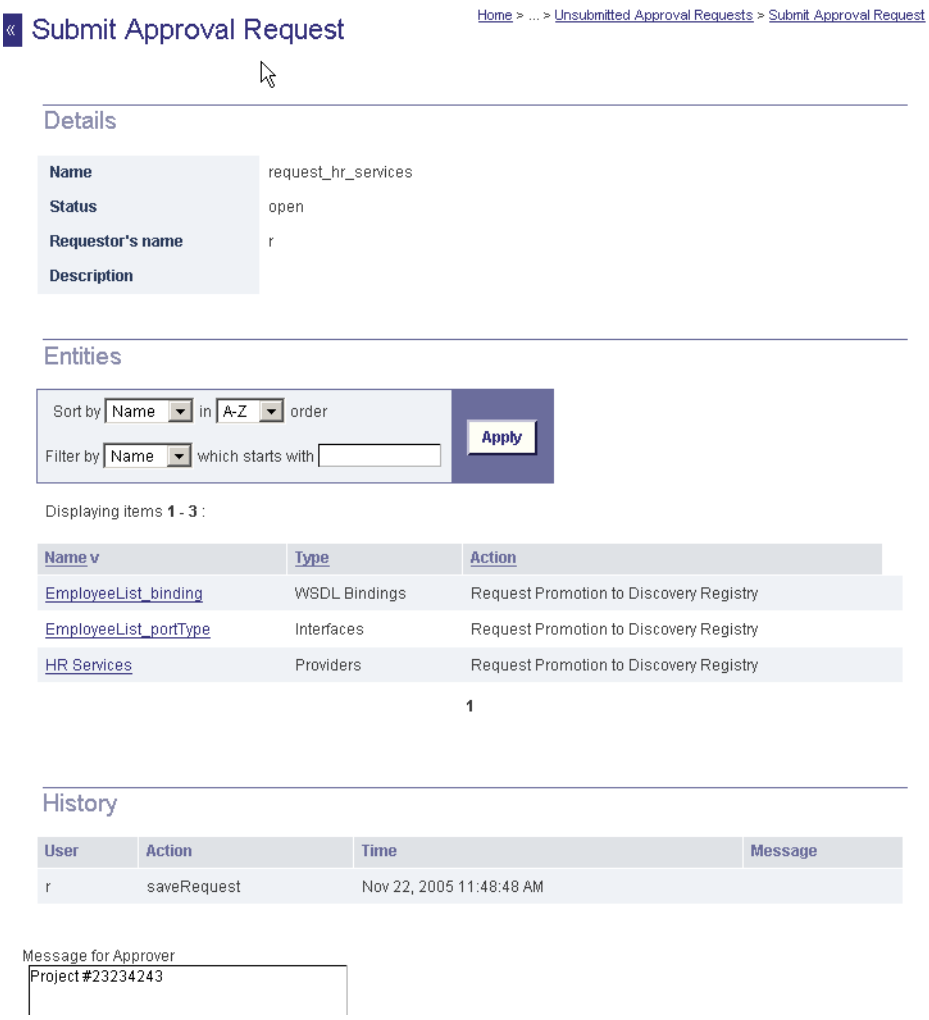
7. Otherwise you are presented with a page displaying unsubmitted approval requests as shown in Figure 34.

Figure 34. Unsubmitted Request



8. Click on a request in the list to display its details as shown in Figure 35. From here you can enter a message for the approver and click **Submit Request for Approval**. Click on the **Back** button to return to the list.

Figure 35. Enter Message for Approver



9. To view your submitted approval request, click on the **Submitted Approval Requests** link under the **Tools** main menu tab. A page similar to that shown in Figure 36 will appear.

Manage Requests

You can manage your approval requests under the **Tools** main menu tab. On this tab, there are the following links for managing your approval requests:

- **Unsubmitted Approval Requests** - The request work list holds requests that you have not yet submitted to an approver. You can add multiple types of entities into a single approval request. The work list is also a place to which you can restore canceled requests or requests for editing and re-approval. The request work list is persistent. You can work with requests in the work list after you log out of the Business Service Console.

- **Submitted Approval Requests** - The **Submitted Approval Requests** link will display a page where you can see your submitted approval request. These requests have been submitted but have not yet been approved or rejected. You can cancel a pending request or remind approver about the request. See Figure 36.

- **Completed Approval Requests** - The **Completed Approval Requests** link will display a page where you can see all your requests that have been approved or rejected. You can delete a request from this list or use the request for to create a new request. For more information, see Section Cloning Requests.

Figure 36. Submitted Approval Requests

Cloning Requests

You can create a new approval request from an existing approval request. We call this operation **cloning**.

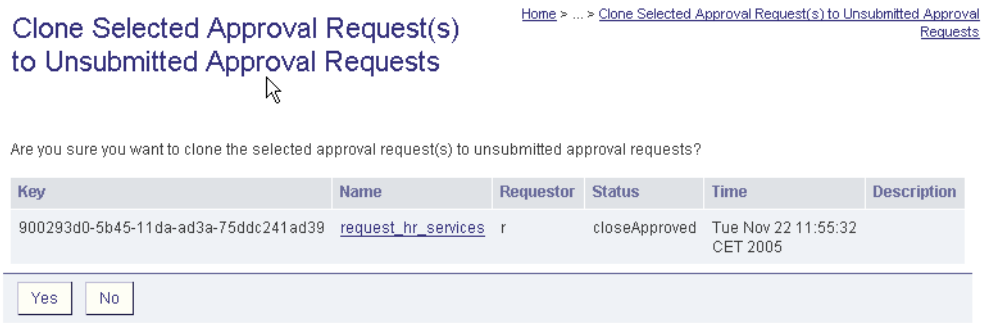
To clone a request, follow these steps:

1. Click on the **Completed Approval Request** link under the **Tools** main menu tab. The page shown in Figure 37 is returned.

Figure 37. Completed Approval Requests

2. Select the approval request and click on the **Clone** icon in the **Action** column. The page shown in Figure 38 will appear. Once you click on the **Yes** button, the new approval request will be created in your request work list with the name starting with `The clone. Name of the original request`. The cloned request contains the same entities as the original request.

Figure 38. Clone Request



4.8.2. Approver's Actions

An approver can perform the following actions:

- Approve/Reject request
- View Approval History

Approve/Reject Request

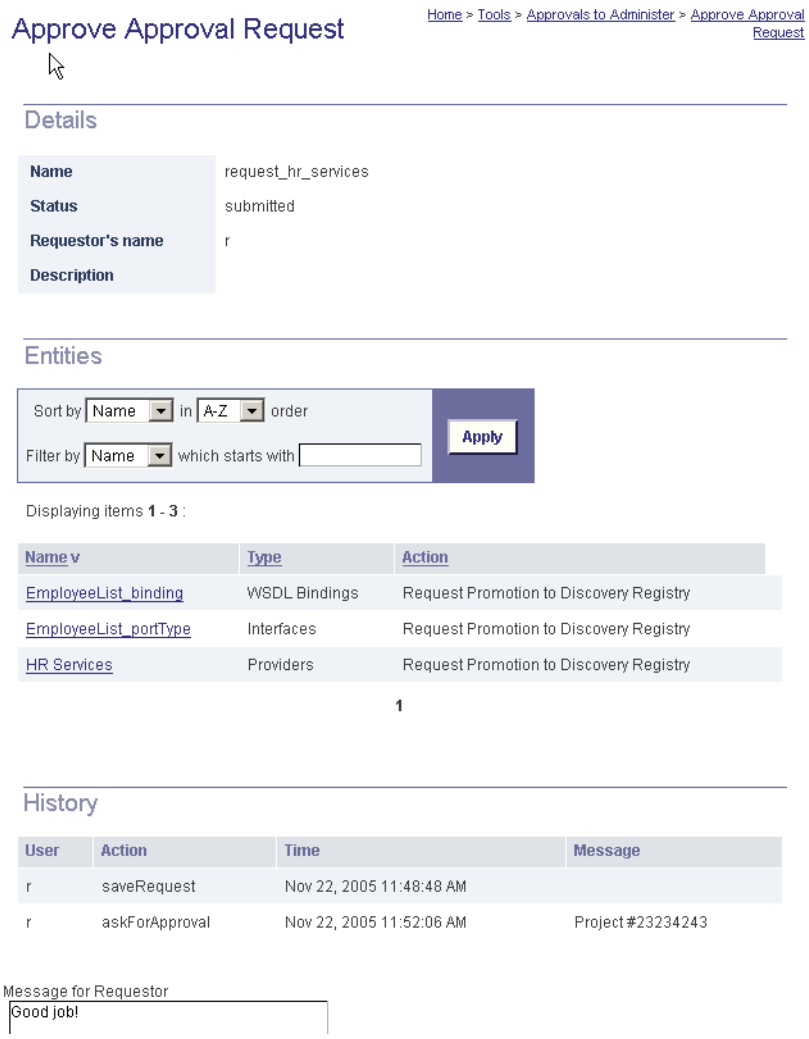
To approve or reject an approval request:

1. Click on the **Approvals to Administer** link under the **Tools** main menu tab. The page shown in Figure 39 will appear.

Figure 39. View Requests to Administer

2. If you click on the request name, you will see the request's detailed information including a list of entities the requestor wants to be promote. To approve or reject the request, click on an appropriate button icon in the **Action** column. If you click **Approve**, the page shown Figure 40 will appear.

Figure 40. Approve Request



3. You can review all entities in the request, see request history, and optionally enter a message for the requestor. Once you click **Approve**, an email notification will be sent to the requestor and entities listed in the request will be promoted to the discovery registry.

View Approval History

Approvers have the ability to see all approval requests they have approved or rejected. To access the approval history, click on the **Approval Admin History** link under the **Tools** main menu link. The **Approval Admin History** page shown in Figure 41 will appear.

Note

Approvers are not allowed to delete any approval requests, only requestors can delete their approval requests.

Figure 41. Approval Admin History

5. Advanced Topics

5.1. Data Access Control: Principles

This chapter describes the entity access control mechanism, which defines permissions for users and groups to access structures in BEA AquaLogic Service Registry

There are two types of user groups: **public** and **private**. Both public and private groups are visible to all users in the registry, meaning that all users are able to see which groups exist. Public and private groups differ in that members of public groups are visible to all users of the registry whereas members of private groups are visible only to the owner of the group.

Note

There are other permissions in BEA AquaLogic Service Registry used to control access to APIs and their operations. API permissions are relations between the user or group and operation only. Please see Section 5, Permissions: Principles in the Administration Guide for details.

Permission in this chapter is limited to Data Access Permission - ACL permission.

We use the following terms with regard to ACL permissions:

- **Party** A user or group of users
- **Core Structure** One of the major UDDI data structures: businessEntity, businessService, bindingTemplate or tModel
- **Action** An operation: "find", "get", "save", or "delete" on the entity plus special action "create", which means to save sub-entities. (For example, a user with the "create" permission on a businessService can save new bindingTemplates under the businessService, but can not update whole businessService.) Note that the "create" permission makes sense only on businessEntity and businessService, because bindingTemplates and tModels have no sub-entities.

Standard UDDI access control defines that only the owner of a UDDI core structure can update or delete it. Every user can find or get the structure (with the exception that deleted/hidden tModels are visible for `get_tModelDetail` but not for the `find_tModel` operation). ACLs (Access Control Lists) added to a UDDI entity can override standard UDDI access control as there are several cases in which standard access control is not sufficient.

Examples:

- When a Web service is under construction, its UDDI representation (businessService and bindingTemplate) should be visible only to members of the development team. Arbitrary users should not be able to obtain it in the result set

of `get_serviceDetail` or `find_service` operations. Moreover, a `get_businessDetail` or `find_business` operation result, which includes a superior businessEntity, should not give away the existence of the businessService.

- On the other hand when the server (where the service prototype is running) goes down, the administrator should be able to deploy the Web service on another server and repair the service endpoint in the accessPoint within its bindingTemplate, despite not being the owner of the bindingTemplate.

5.1.1. Explicit Permissions

Explicit permission gives (positive permission), or revokes (negative permission), access rights to a party to process an action on a specified entity.

Explicit permissions are saved with the entity as special keyedReferences in the categoryBag. For more information, please see Setting ACLs on UDDI v3 Structures and Setting ACLs on UDDI v1 and v2 Structures below.

5.1.2. Permission Rules

When no explicit permission is set for the find/get action on an entity, everyone can find/get it. When no explicit permission is set for the save/delete action on an entity, only owner of the entity can save/delete it. This is a standard UDDI access control. When an explicit Permission is set for an action, a completely different access control is used which is defined by the following rules:

1. **Owner always has full control** The owner can always process an operation over an owned entity, even if the permission is explicitly revoked.

2. **Negative permission for a user overrides positive permission for a user.** Example: User U has explicit *positive* permission on businessEntity BE for the `get` action. However, if U also has explicit *negative* permission on BE for action `get`, then an attempt to process `get_businessDetail` by user U on the BE will fail.

3. **Negative permission for group overrides positive permission for group.** Example: User U has belongs to groups G1 and G2. Group G1, has explicit *positive* permission on the BE for action `get`. Group G2, has explicit *negative* permission on the BE for action `get`. Because of this negative permission, any attempt to process `get_businessDetail` by user U on the BE will fail.

4. **Permission for user has more weight than permission for group** Example: User U has explicit *positive* permission on businessEntity BE for action `get`. Group G, to which U belongs, has explicit *negative* permission on the BE for action `get`. User U can process `get_businessDetail` on the BE, even though U belongs to group G.

5. **The owner of an entity can always process `get_XXX` on a direct sub-entity** Example: User U1 owns businessEntity BE. U1 (as owner) grants "create" permission to user U2. Then U2 saves new businessService BS with bindingTemplate BT under BE. When user U1 executes `get_businessDetail`, U1 obtains BE with BS but without BT, because BT is not a direct sub-element of the BE.

   Motivation: This rule ensures that the owner of an entity will see all direct sub-entities. The number of sub-entities is limited. By default, a user can save only one businessEntity, four businessServices per businessEntity, two bindingTemplates per businessService and 10 tModels. Suppose that user U1 has businessEntity BE. User U2 can save businessServices in BE (permission "create" on BE). If U2 has already saved four businessServices under BE, user U1 cannot, therefore, save a new businessService. Therefore, the owner of an businessEntity should see why the limit is reached.

6. **Delete and Save positive permissions are inherited from parent entities and override negative permissions on sub-entities** Example: User U has "delete" permission on businessEntity BE. Then U can execute the `delete_business` operation, which deletes the BE with all its businessServices and bindingTemplates, even if some of these sub-entities have negative permission for deletion by the user U.

Motivation: Sub-entities can not survive parent entity deletion. This rule ensures that a user who can save/delete an entity can do this despite not having sufficient privileges on sub-entities.

7. **To perform update by `save_XXX` operation, it is necessary to have both "save" and "get" permissions** Example: User U1 has "save" and "get" permissions on businessEntity BE, but he is not the owner. User U2 owns the BE and saves businessService BS1, which has "get" permission for U1, and businessService BS2 without any permissions. Both BS1 and BS2 are created under BE. U1 gets BE with only BS1 and updates BE in this way: U1 can add a category and save BE again without BS1. In fact, when BE is updated, BS1 is deleted but BS2 remains.

Example:

User U1 owns a businessEntity BE. The user U1 defines the explicit `get allowed` permission to user group G1. Everyone can find the BE, because there is no explicit permission for `find` and therefore the standard UDDI access control is used. On the other hand, only user U1 (as the owner) and all users from group G1 can get the BE.

5.1.3. Composite Operations

BusinessService BS can be moved from one businessEntity BE1 to other businessEntity BE2. By performing the `save_service` operation on BS, where BS has updated businessKey to point to the BE2. To perform this action, the party must have permission to save BE1, BE2, and BS, because all these entities are changed.

Similarly bindingTemplate BT can be moved from businessService BS1 to businessService BS2. The party who moves it must have save permission on BS1, BS2 and BT.

BusinessService BS hosted in businessEntity BE1 can be projected into businessEntity BE2. The party who projects BS must have save permission on BE2.

5.1.4. Pre-installed Groups

ACL logic considers some special pre-published abstract groups during permission evaluation. These abstract groups allow a publisher to give a permission to a specific set of BEA AquaLogic Service Registry users.

`system#everyone`

Holds all users of BEA AquaLogic Service Registry (both users who have and who do not have a BEA AquaLogic Service Registry account, authenticated and non-authenticated). If this group is used, all users always have the specified permission to the associated data.

`system#registered`

Holds all authenticated BEA AquaLogic Service Registry users. Every user who is authenticated (that is, who has an account and has logged into the registry) is a member of this group. If this group is used, all authenticated users always have the specified permission to the associated data.

`system#intranet`

Holds users who access BEA AquaLogic Service Registry via a local intranet. (This group is reserved for a future release. There is no implementation behind it as of BEA AquaLogic Service Registry 6.5)

5.1.5. ACL tModels

ACL permissions are represented as tModels as detailed below:

ACL Permission	v3 tModelKey	v2 tModelKey
find allowed	uddi:systinet.com:acl:find-allowed	uuid:aacfc8e0-dcf5-11d5-b238-cbbeaea0a8d4
find denied	uddi:systinet.com:acl:find-denied	uuid:ced3c160-dcf5-11d5-b238-cbbeaea0a8d4
get allowed	uddi:systinet.com:acl:get-allowed	uuid:f9977a90-dcf5-11d5-b238-cbbeaea0a8d4

ACL Permission	v3 tModelKey	v2 tModelKey
get denied	uddi:systinet.com:acl:get-denied	uuid:09e202d0-dcf6-11d5-b238-cbbeaea0a8d4
save allowed	uddi:systinet.com:acl:save-allowed	uuid:19885bd0-dcf6-11d5-b239-cbbeaea0a8d4
save denied	uddi:systinet.com:acl:save-denied	uuid:2a25e610-dcf6-11d5-b239-cbbeaea0a8d4
delete allowed	uddi:systinet.com:acl:delete-allowed	uuid:37f44ac0-dcf6-11d5-b239-cbbeaea0a8d4
delete denied	uddi:systinet.com:acl:delete-denied	uuid:4e51d8f0-dcf6-11d5-b239-cbbeaea0a8d4
create allowed	uddi:systinet.com:acl:create-allowed	uuid:5bc32980-dcf6-11d5-b239-cbbeaea0a8d4
create denied	uddi:systinet.com:acl:create-denied	uuid:6d0be7e0-dcf6-11d5-b239-cbbeaea0a8d4

5.1.6. Setting ACLs on UDDI v3 Structures

In UDDI v3, explicit ACL permission is saved in a special keyedReferenceGroup having the tModelKey `uddi:systinet.com:acl`. This keyedReferenceGroup can contain only keyedReferences to ACL tModels. Only the terms "user" and "group" are allowed in the included keyName, and the keyValue must contain the name of the user or group (according to keyName value).

For example, user demo_john can save (update) following businessEntity even if he is not the owner:

Example 1. Setting ACLs - v3

```
<businessEntity xmlns="urn:uddi-org:api_v3">
  ...
  <categoryBag>
    ...
    <keyedReferenceGroup tModelKey="uddi:systinet.com:acl">
      <keyedReference tModelKey="uddi:systinet.com:acl:save-allowed"
      keyName="user" keyValue="demo_john"/>
      ...
    </keyedReferenceGroup>
  </categoryBag>
</businessEntity>
```

5.1.7. Setting ACLs on UDDI v1/v2 Structures

Under versions 1 and 2 of UDDI, explicit ACL permission is saved as a special keyedReference in the categoryBag. This keyedReference refers to one of the tModels representing ACL permissions. Only the terms "user" and "group" are allowed in the included keyName and the keyValue must contain the name of the user or group (according to the keyName value).

For example, user demo_john can save (update) following businessEntity even if he is not the owner:

```
<businessEntity ...>
  ...
  <categoryBag>
    <keyedReference tModelKey="uuid:19885bd0-dcf6-11d5-b239-cbbeaea0a8d4"
                                          keyName="user" keyValue="demo_john"/>
    ...
  </categoryBag>
</businessEntity>
```

Note

ACL permissions cannot be set on the bindingTemplate structure because this structure has no categoryBag in UDDI v1/v2.

5.2. Publisher-Assigned Keys

Under UDDI v1 and v2, keys are generated automatically when a structure is published. Generated keys in these versions are in form `(uuid:)8-4-4-4-12` where the numbers indicate a count of hexadecimal values. For example, `uuid:327A56F0-3299-4461-BC23-5CD513E95C55`. Note that the prefix "uuid:" was only used in tModelKeys.

In UDDI v3 users may assign keys when saving a structure for the first time. These Keys can be 255 characters long and can contain numbers and Latin characters, so that the key itself describes what the UDDI structure means. For example, the key `uddi:systinet.com:uddiRegistry:demo:businessService` has the following elements:

- The prefix `uddi:` is a schema much like `http:` or `ftp:` and must be always present.
- `systinet.com` is an optional host name.
- The elements `uddiRegistry`, `demo`, and `businessService` represent a hierarchy of domains. The domain `demo` is a subdomain of `uddiRegistry`.

This description is sufficient for our purposes for now. For a more precise description of keys, please see the UDDI v3 Specification [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047261].

5.2.1. Generating Keys

The key generator tModel is a tModel with a key in the form `domain:keygenerator`. This tModel permits its owner to save structures with keys in the form `domain:string`. For example, the tModel `uddi:systinet.com:uddiRegistry:demo:keygenerator` allows its owner to publish structures with keys like:

- `uddi:systinet.com:uddiRegistry:demo:businessService`
- `uddi:systinet.com:uddiRegistry:demo:b52`

These are derived keys of the `uddi:systinet.com:uddiRegistry:demo` domain.

With one exception, the key generator tModel does *not* allow the user to save keys from subdomains such as `uddi:systinet.com:uddiRegistry:demo:businessService:exchangeRate`, that is, derived keys of `uddi:systinet.com:uddiRegistry:demo:businessService`.

The key generator tModel, however, permits the user to save the key generator for each direct subdomain. For example, the user can save `uddi:systinet.com:uddiRegistry:demo:businessService:keygenerator`. After creating this second key generator, the user is permitted to save structures with keys of the `uddi:systinet.com:uddiRegistry:demo:businessService` domain, such as `uddi:systinet.com:uddiRegistry:demo:businessService:exchangeRate`.

Important

To generate keys for a domain, the user must own the domain's key generator tModel. Only the administrator can save structures with assigned keys without having the key generator tModel. To enable this process for other users, the administrator must save the domain's tModel and then change its ownership to the user via custody transfer. For more information, please see Section Publish Custody Transfer.

5.2.2. Affiliations of Registries

The rules above ensure that two users can not create structures with the same key. A complicated situation arises when one user wants to copy UDDI structures from one registry to another while preserving the keys of those structures. There are two problems:

1. The key of the copied structure must not exist on the second registry. The key must be unique - this is required by the UDDI specification.
2. The user must be allowed to save a structure with a specified key on the second registry.

The **Affiliated registries** mechanism solves both problems. An affiliation is a relationship between two registries. The first registry gives up generation of keys for a certain domain and transfers this privilege to the second registry. This ensures that keys from both registries are unique.

Note

In the examples below we name the two registries 'master' and 'slave'. Moreover there are three people:

- The person 1 is an administrator of the master registry, this account is called **master-admin**.
- The person 2 is an administrator of the slave registry (account **slave-admin**) and a common user on the master registry (account **master-user2**).
- The person 3 is a common user on slave registry (account **slave-user3**) and a common user on master registry (account **master-user3**).

Affiliation Setup

To set up an affiliation:

1. The administrator of the slave registry (slave-admin) registers a user account on the master registry (master-user2).
2. Master-user2 requests a key generator tModel from the administrator of the Master registry.
3. This administrator, master-admin, creates the key generator tModel and transfers it to the master-user2 account using custody transfer.
4. Person 2 manually copies the key generator tModel to the slave registry (his slave-admin account has permission to assign any key) and sets up the slave registry to generate all keys based on this key generator. For more information, please see Section 2.7, Node in the Administrator's Guide.

All keys generated by the slave registry or its users will be from the domain or some subdomain defined by the key generator.

Copying Structures with Key Preservation

Given key should refer to the same structure no matter which registry the structure is in.

Suppose that slave-admin creates a key generator tModel for slave-user3 and this user uses the key generator to generate a key for a structure in the slave registry. To copy the structure to the master registry, this key generator tModel must exist on both registries.

To copy a structure from the slave to the master registry:

1. The slave-user3 must ask person 2 (slave-admin) to copy the second key generator, because only the holder of the account master-user2, as owner of the first key generator, can do this on the master registry.

2. Then master-user2 transfers ownership of the second key generator in the master registry to master-user3. Now master-user3 can copy the structure while preserving the generated keys.

5.3. Range Queries

BEA AquaLogic Service Registry's range queries functionality allows you to search UDDI entities with the ability to use comparative operators (>, <) for matching keyValues in keyedReferences. There must be a defined type of keyValues in the taxonomy which defines the ordering. The following ordering types are supported: `string`, `numeric`, and `custom`. KeyedReferences in `find_XXX` queries are extended by a list of find qualifiers. Do not mix with find qualifiers of the whole query. Find Qualifiers are used for specifying comparison operators.

See Section Find Business by Categories how to search UDDI data structures using range queries with Registry Console.

Note

The BEA AquaLogic Service Registry implementation of range queries goes beyond the current UDDI v3 specification since the specification does not define this functionality.

The following findQualifiers are supported:

- `equal` - the default find qualifier. If no one from the group of (equal, greaterThan, lesserThan qualifiers) is specified. This is done due to the backward compatibility with a standard UDDI. When used, the keyedReference from the request matches to the all keyedReferences from the database with the same tModelKey and the **same** keyValue.

- `greaterThan` - When used, the keyedReference from the request match to the all keyedReferences from the database with the same tModelKey and a **greater** keyValue.

- `lesserThan` - When used, the keyedReference from the request match to the all keyedReferences from the database with the same tModelKey and a **lesser** keyValue.

- `notExists` - This findQualifier has validity for the whole keyedReference (not just for keyValues). An entity matches the find request with `notExists` findQualifier if and only if the specific keyedReference does not exist in its categoryBag. This findQualifier can be arbitrarily combined with `greaterThan`, `lesserThan` and `equal` findQualifiers. If the `notExists` findQualifier is used alone, then the `equal` findQualifier is considered automatically.

Comparators can be combined:

- `greaterThan` and `equal` find qualifiers can be used together with the keyedReference match to the all keyedReferences with the same tModelKey and a **greater or equal** keyValue (>=).

- `lesserThan` and `equal` find qualifiers can be used together with the keyedReference match to the all keyedReferences with the same tModelKey and a **lesser or equal** keyValue (<=).

- `lesserThan` and `greaterThan` find qualifiers can be used together with the keyedReference match to the all keyedReferences with the same tModelKey and a **not equals** keyValue (<>).

- Combination of lesserThan, greaterThan and equal is not allowed.

5.3.1. Examples

The following examples demonstrate the usage of range queries. Suppose that the keyedReferences are placed in the category bag of the `find_business` request.

greaterThan Only business entities that have a keyedReference with tModelKey equal to tmKey, and a keyValue that is greater than kv, in their categoryBags are returned.

```
<keyedReference tModelKey="tmKey" keyValue="kv" keyName="kn">
   <findQualifiers>
      <findQualifier>greaterThan</findQualifier>
   </findQualifiers>
</keyedReference>
```

greaterThan and lesserThan Only business entities that have keyedReference with tModelKey that is equal to tmKey, and a keyValue not equal to kv, in their categoryBags are returned.

```
<keyedReference tModelKey="tmKey" keyValue="kv" keyName="kn">
   <findQualifiers>
      <findQualifier>greaterThan</findQualifier>
      <findQualifier>lesserThan</findQualifier>
   </findQualifiers>
</keyedReference>
```

notExists Only business entities that do not have a keyedReference with a tModelKey equal to tmKey, and a keyValue equal to kv, in their categoryBags are returned.

```
<keyedReference tModelKey="tmKey" keyValue="kv" keyName="kn">
   <findQualifiers>
      <findQualifier>notExists</findQualifier>
   </findQualifiers>
</keyedReference>
```

notExists and greaterThan Only business entities that do not have a keyedReference with a tModelKey equal to tmKey, and a keyValue greater than kv, in their categoryBags are returned.

```
<keyedReference tModelKey="tmKey" keyValue="kv" keyName="kn">
   <findQualifiers>
      <findQualifier>notExists</findQualifier>
      <findQualifier>greaterThan</findQualifier>
   </findQualifiers>
</keyedReference>
```

notExists, greaterThan, equal Only business entities that do not have a keyedReference with a tModelKey equal to tmKey, and a keyValue greater than or equal to kv, in their categoryBags are returned.

```
<keyedReference tModelKey="tmKey" keyValue="kv" keyName="kn">
   <findQualifiers>
      <findQualifier>notExists</findQualifier>
      <findQualifier>greaterThan</findQualifier>
      <findQualifier>equal</findQualifier>
   </findQualifiers>
</keyedReference>
```

See also Demos, Section 2.1, Advanced Inquiry - Range Queries.

5.4. Taxonomy: Principles, Creation and Validation

The UDDI Version 3 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3] provides tools for setting the context on all four major UDDI structures: businessEntities, businessServices, bindingTemplates and tModels. This document covers basic principles and management of this feature - the taxonomies.

5.4.1. What Is a Taxonomy?

A taxonomy, or value set in the terminology of the UDDI specifications, is a tModel which can be used in categoryBags, identifier bags, or Publisher Assertions. This tModel must be in a specific form, so that BEA AquaLogic Service Registry can recognize it as a taxonomy. The tModel must be categorized with the type of taxonomy and, optionally, with information concerning whether and how to validate the values in keyedReferences.

5.4.2. Taxonomy Types

The UDDI specification distinguishes four types of taxonomies: categorizations, categorizationGroups, identifiers, and relationships.

Categorizations

Categorizations can be used in all four main UDDI structures. They are used to tag them with additional information, such as identity, location, and what the taxonomy describes.

CategorizationGroups

New in UDDI version 3, CategorizationGroups group several categorizations into one logical categorization. For example, a geographical location comprised of two categorizations: longitude and latitude.

Identifiers

Used in businessEntities and tModels, Identifiers reference published information.

Relationships

Used only in Publisher Assertions, Relationships define the relation between two businessEntities.

5.4.3. Validation of Values

The publisher of a taxonomy can decide whether the values in keyedReferences within the taxonomy will be checked or not.

Unchecked Taxonomies

BEA AquaLogic Service Registry does not perform any checks on values used in keyedReferences associated with unchecked taxonomies. Unchecked taxonomies are those that are marked as such, *or* those that are not marked as checked. These two states are equivalent.

Checked Taxonomies

If a taxonomy is checked, BEA AquaLogic Service Registry executes its validation service for every keyedReference in which the checked taxonomy is used. The validation service may check the expected syntax of values, such as the format of a credit card or ISBN number. Taxonomies like the ISO 3166 Geographic taxonomy, which permits only existing countries, check the existence of the value against a list. A validation service may even permit or deny values depending on the context in which they are used.

BEA AquaLogic Service Registry Requirements

BEA AquaLogic Service Registry conforms to the technical note Providing A Value Set For Use In UDDI Version 3 [http://oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-valuesetprovider-20030212.htm]. To create a checked taxonomy, you must:

1. Prepare and deploy a validation service which implements the `Valueset_validation` API.
2. Publish the tModel categorized as a checked taxonomy and mark it as unvalidatable.
3. Publish the bindingTemplate that implements the `Valueset_validation` API and the taxonomy's tModel.
4. Republish the tModel, without the unvalidatable categorization, and with the categorization `uddi-org:validatedBy` pointing to the bindingTemplate.

BEA AquaLogic Service Registry requires that the bindingTemplate be published in the businessService of the Operational Business Entity. If this businessService is not part of the Operational Business Entity, the checked taxonomy will not be validatable and thus it may not be used in keyedReferences. This implies that only the BEA AquaLogic Service Registry administrator may publish checked taxonomies.

The bindingTemplate must contain an accessPoint with its `useType` attribute set to `"endPoint"`.

If the accessPoint starts with the prefix `class:`, then the remaining part is assumed to contain the fully qualified name of the class that implements interface and is accessible by the BEA AquaLogic Service Registry classloader.

If the accessPoint does not start with the prefix `class:`, it is assumed to be the URL of the Web service implementing the `Valueset_validation` API and a stub is created for this Web service.

Internal Validation Service

BEA AquaLogic Service Registry contains a special validation service called the Internal Validation Service. This service is used by checked taxonomies that declare a list of available values published using the Systinet Taxonomy API.

5.4.4. Types of keyValues

The creator of the taxonomy must specify types of keyValues by assigning the appropriate comparator reference (comparator tModel) of the `systinet-com:isOrderedBy` taxonomy to the categorization taxonomy you want to use to categorize a UDDI entity. The following types of key values types are supported:

- `string` - keyValues are treated as string values. If keyValues type is unknown then keyValues are treated as strings. The maximum length is 255 characters.
- `numeric` - keyValues are treated as decimal numbers. The value can have maximum 19 digits before the decimal point and maximum 6 digits after the decimal point.
- `custom` - keyValues must be transformed to string or numeric values using a transformation service. Please see Section Custom Ordinal Types for more information.

For example, the tModel of the categorization taxonomy with numeric key values must have the following keyedReference in its category bag:

```
<keyedReference tModelKey="uddi:systinet.com:isOrderedBy"
   keyValue="uddi:systinet.com:comparator:numeric"/>
```

Figure 42. Example of Numeric Categorization

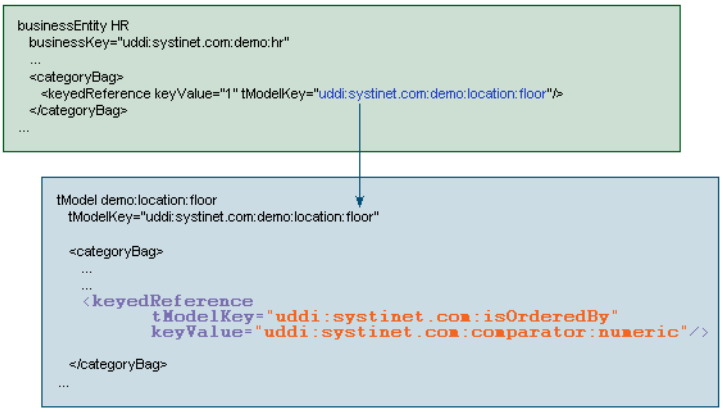


Figure 42 shows how the `demo:location:floor` taxonomy from Demo data can be assigned numeric key values.

Important

If you change type of keyValues of the taxonomy and there are entities in the BEA AquaLogic Service Registry that were already categorized with the taxonomy, the BEA AquaLogic Service Registry administrator must execute the task **Transform keyed references**. The button for executing this task is located in the Registry Console under the **Manage** tab, **Registry Management** link. See Administrator's Guide, Section 1.1, Accessing Registry Management

- To learn how to make this assignment using the Registry Console , see User's Guide, Section Adding a Category.
- See User's Guide, Section 5.5.5, Searching how to search UDDI data structures using range queries with Registry Console.
- See Administrator's Guide , Section 1.5.3, Editing Taxonomies how to edit taxonomy type.

Custom Ordinal Types

You can define your custom ordinal types. To demonstrate possible extensions, BEA AquaLogic Service Registry contains two demo comparators:

- `systinet-com:comparator:date`
- `systinet-com:comparator:stringToLowerCase`

Let's assume you want to create a taxonomy with date values in keyValues. You must mark the taxonomy tModel (that is, add the following keyedReference into its categoryBag) by `<keyedReference tModelKey="uddi:systinet.com:isOrderedBy" keyValue="uddi:systinet.com:comparator:date"/>`. It is quite easy because there is a demo comparator for date in the registry. Imagine the date comparator is not present. Take the following steps to create it in the registry:

1. Create a transformer service that transforms the date value into a string or numeric value. The transformer service must implement `TransformerKeyedReferenceApi` and add this class to the BEA AquaLogic Service Registry class path.
2. Create a new comparator tModel for date. The tModel must be categorized as a comparator using the `systinet-com:comparator` taxonomy. The comparator must refer to the transformer service. This reference is specified by

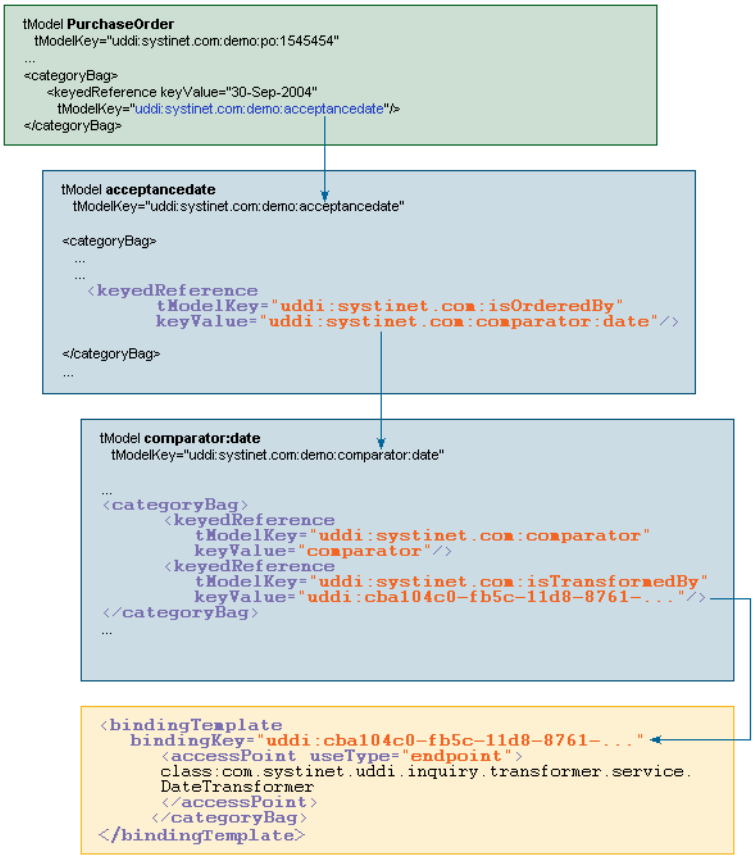
the taxonomy `IsTransformedBy` (where "uddi:cba104c0-fb5c-11d8-8761-eb2505508761" is the key of the bindingTemplate with the specification of the transformer service.

Important

If you change implementation of the of the transformer service of the taxonomy and there are entities in the BEA AquaLogic Service Registry that were already categorized with the taxonomy, the BEA AquaLogic Service Registry administrator must execute the task **Transform keyed references**. The button for executing this task is located in the Registry Console under the **Manage** tab, **Registry Management** link. See Administrator's Guide, Section 1.1, Accessing Registry Management

Figure 43 shows the tModel references for date categorization ordering. It describes a purchase order document that has been mapped to BEA AquaLogic Service Registry via XML-to-UDDI functionality, and then categorized by the `acceptancedate` taxonomy. The categorization taxonomy must refer to the comparator tModel `uddi:systinet.com:comparator:date` that references a bindingTemplate with the location of the date transformation service.

Figure 43. Example of Custom Categorization (date)



The transformer service is called whenever the appropriate keyedReference is processed. If any entity contains the keyedReference with a taxonomy tModel whose type is custom then the transformer service is called to discover the correct (that is, transformed) keyValue of the keyedReference. Such transformed values are stored into the database. If you want to find entities by this keyedReference (the keyedReference with the same taxonomy tModel), the service is called again to get the transformed value. Transformed values are used for the saving and searching of keyedReferences.

See Administrator's Guide , Section 1.5.3, Editing Taxonomies how to edit taxonomy type.

5.4.5. Taxonomy API

This section demonstrates the basics of taxonomy API and taxonomy persistence format. A comprehensive description of the Taxonomy API can be found in the Developer's Guide, Section 2.2.2, Taxonomy.

Note

For clarity, we use an XML representation, but you can achieve the same results with Java objects.

```
<taxonomy xmlns="http://systinet.com/uddi/taxonomy/v3/5.0"
        xmlns:uddi="urn:uddi-org:api_v3"
         check="false">
    <tModel tModelKey="uddi:systinet.com:demo:myTaxonomy">
        <uddi:name>My taxonomy</uddi:name>
        <uddi:description>Category system</uddi:description>
    </tModel>
    <compatibilityBag>
        <compatibility>businessEntity</compatibility>
    </compatibilityBag>
    <categorizationBag>
        <categorization>categorization</categorization>
    </categorizationBag>
</taxonomy>
```

Each taxonomy, in order to be saved, requires a valid tModel. While it must contain a tModelKey and a name, you do not need to set the content of the categoryBag.

- The Taxonomy attribute `check` determines whether the taxonomy will be checked or not.
- The compatibilityBag is an interface to Systinet's `uddi:systinet.com:taxonomy:categorization` taxonomy, which is used to limit usage of the selected taxonomy within the four main UDDI structure types. In this way you can enforce that your taxonomy can be used only within the UDDI structures of your choice and not in others.
- The categorizationBag is used to declare the type of the taxonomy, for example, whether it is a categorization, categorizationGroup, identifier or relationship taxonomy.

  Note that values may be combined.

Let's enhance the previous example and convert the taxonomy from unchecked to checked. Checked taxonomies must contain Validation. In this example, the taxonomy is checked by the Custom Validation Web service located at `http://www.foo.com/MyValidationService.wsdl`.

```
<taxonomy xmlns="http://systinet.com/uddi/taxonomy/v3/5.0"
        xmlns:uddi="urn:uddi-org:api_v3"
         check="true">
    <tModel tModelKey="uddi:foo.com:demo:myTaxonomy">
        <uddi:name>My taxonomy</uddi:name>
        <uddi:description>Category system</uddi:description>
    </tModel>
    <compatibilityBag>
        <compatibility>businessEntity</compatibility>
    </compatibilityBag>
```

```
    <categorizationBag>
        <categorization>categorization</categorization>
    </categorizationBag>
    <validation>
        <bindingTemplate bindingKey="" serviceKey="" xmlns="urn:uddi-org:api_v3">
            <accessPoint useType="endPoint">
               http://www.foo.com/MyValidationService.wsdl
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo
                        tModelKey="uddi:uddi.org:v3_valueSetValidation"/>
                <tModelInstanceInfo
                        tModelKey="uddi:systinet.com:demo:myTaxonomy"/>
            </tModelInstanceDetails>
        </bindingTemplate>
    </validation>
</taxonomy>
```

The `validation` element must hold the bindingTemplate identifying the validation Web service or categories structures. In this example we chose bindingTemplate. It must contain complete accessPoint and tModelInstanceDetails must hold the `Valueset_validation` API and tModelKey of the saved taxonomy. If the serviceKey is specified and if the businessService already exists, it must be part of the Operational Business Entity.

Important

Be aware that the service will be replaced during the `save_taxonomy` process.

If you can provide a list of allowed values, you do not need to implement your own validation Web service. Just provide the allowed values inside the `categories` structure (as shown below) and the Internal Validation Service will be responsible for validation of the keyedReferences.

```
<taxonomy xmlns="http://systinet.com/uddi/taxonomy/v3/5.0"
        xmlns:uddi="urn:uddi-org:api_v3"
         check="true">
    <tModel tModelKey="uddi:foo.com:demo:myTaxonomy">
        <uddi:name>My taxonomy</uddi:name>
        <uddi:description>Category system</uddi:description>
    </tModel>
    <compatibilityBag>
        <compatibility>businessEntity</compatibility>
    </compatibilityBag>
    <categorizationBag>
        <categorization>categorization</categorization>
    </categorizationBag>
    <validation>
        <categories>
            <category keyName="Value A" keyValue="A"/>
            <category keyName="Value B" keyValue="B">
                <category keyName="Value B1" keyValue="B1"/>
                <category keyName="Value B3" keyValue="B3" disabled="true" />
            </category>
            <category keyName="Value C" keyValue="C"/>
        </categories>
```

```
    </validation>
</taxonomy>
```

As you can see, you can arrange your values hierarchically. This is useful for the Registry Console that implements the drill-down pattern. If you really need, you can even specify bindingTemplate along with the `categories` structure, but its accessPoint must point to the Internal Validation Service.

5.4.6. Predeployed Taxonomies

BEA AquaLogic Service Registry comes with the following predeployed taxonomies:

- `uddi-org:types` is a UDDI Type Category System.

v3 UDDI key	`uddi:uddi.org:categorization:types`
v2 UUID key	`uuid:c1acf26d-9672-4404-9d70-39b756e62ab4`
Categorization	categorization
Compatibility	tModel
Checked	yes, Internal Validation Service

- `uddi-org:general_keywords` is a category system consisting of namespace identifiers and the keywords associated with namespaces.

v3 UDDI key	`uddi:uddi.org:categorization:general_keywords`
v2 UUID key	`uuid:A035A07C-F362-44dd-8F95-E2B134BF43B4`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes

- `uddi-org:entityKeyValues` is a category system used to declare that a value set uses entity keys as valid values.

v3 UDDI key	`uddi:uddi.org:categorization:entitykeyvalues`
v2 UUID key	`uuid:916b87bf-0756-3919-8eae-97dfa325e5a4`
Categorization	categorization
Compatibility	tModel
Checked	yes, Internal Validation Service

- `uddi-org:isreplacedby` is the identifier system used to point to the UDDI entity, using UDDI keys, that is the logical replacement for the one in which `isReplacedBy` is used.

v3 UDDI key	`uddi:uddi.org:identifier:isReplacedBy`
v2 UUID key	`uuid:e59ae320-77a5-11d5-b898-0004ac49cc1e`
Categorization	identifier
Compatibility	tModel, businessEntity
Checked	yes

- `uddi-org:nodes` is a category system for identifying the nodes of a registry.

v3 UDDI key	`uddi:uddi.org:categorization:nodes`
v2 UUID key	`uuid:327A56F0-3299-4461-BC23-5CD513E95C55`
Categorization	categorization
Compatibility	businessEntity
Checked	yes

- `uddi-org:owningBusiness_v3` is a category system used to point to the businessEntity associated with the publisher of the tModel.

v3 UDDI key	`uddi:uddi.org:categorization:owningbusiness`
v2 UUID key	`uuid:4064c064-6d14-4f35-8953-9652106476a9`
Categorization	categorization
Compatibility	tModel
Checked	yes

- `uddi-org:validatedBy` is a category system used to point a value set or category group system tModel to associated value set Web service implementations.

v3 UDDI key	`uddi:uddi.org:categorization:validatedby`
v2 UUID key	`uuid:25b22e3e-3dfa-3024-b02a-3438b9050b59`
Categorization	categorization
Compatibility	tModel
Checked	yes

- `uddi-org:wsdl:types` is a WSDL Type Category System.

v3 UDDI key	`uddi:uddi.org:wsdl:types`
v2 UUID key	`uuid:6e090afa-33e5-36eb-81b7-1ca18373f457`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes, Internal Validation Service

- `uddi-org:wsdl:categorization:protocol`

v3 UDDI key	`uddi:uddi.org:wsdl:categorization:protocol`
v2 UUID key	`uuid:4dc74177-7806-34d9-aecd-33c57dc3a865`
Categorization	categorization
Compatibility	tModel
Checked	yes

- `uddi-org:wsdl:categorization:transport`

v3 UDDI key	`uddi:uddi.org:wsdl:categorization:transport`
v2 UUID key	`uuid:e5c43936-86e4-37bf-8196-1d04b35c0099`
Categorization	categorization
Compatibility	tModel
Checked	yes

- `uddi-org:wsdl:portTypeReference` is a category system tModel that can be used to identify a relationship to a portType tModel.

v3 UDDI key	`uddi:uddi.org:wsdl:portTypeReference`
v2 UUID key	`uuid:082b0851-25d8-303c-b332-f24a6d53e38e`
Categorization	categorization
Compatibility	tModel
Checked	yes

- `systinet-com:taxonomy:compatibility` enhances a taxonomy tModel with additional information, in which structures the taxonomy can be used.

v3 UDDI key	`uddi:systinet.com:taxonomy:compatibility`
v2 UUID key	`uuid:cf68c700-f93d-11d6-8cfc-b8a03c50a862`
Categorization	categorization
Compatibility	tModel
Checked	yes, Internal Validation Service

- `systinet-com:dependency` creates link between two structures (may be different types). Both keyName and keyValue must be specified. KeyName must be one of businessEntity, businessService, bindingTemplate and tModel. KeyValue must be existing UDDI key of specified structure.

v3 UDDI key	`uddi:systinet.com:dependency`
v2 UUID key	`uuid:179e5540-f27b-11d6-9738-b8a03c50a862`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes

- `dnb-com:D-U-N-S` - Thomas Registry Suppliers

v3 UDDI key	`uddi:uddi.org:ubr:identifier:dnb.com:d-u-n-s`
v2 UUID key	`uuid:8609c81e-ee1f-4d5a-b202-3eb13ad01823`
Categorization	identifier
Compatibility	tModel, businessEntity, businessService, bindingTemplate

Checked	no

- `microsoft-com:geoweb:2000` - Geographic Taxonomy: GeoWeb (2000 Release)

v3 UDDI key	`uddi:297aaa47-2de3-4454-a04a-cf38e889d0c4`
v2 UUID key	`uuid:297aaa47-2de3-4454-a04a-cf38e889d0c4`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	no

- `ntis-gov:naics:1997` - Business Taxonomy: NAICS (1997 Release)

v3 UDDI key	`uddi:uddi.org:ubr:categorization:naics:1997`
v2 UUID key	`uuid:c0b9fe13-179f-413d-8a5b-5004db8e5bb2`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes, Internal Validation Service

- `ntis-gov:sic:1997` - Business Taxonomy: SIC (1997 Release)

v3 UDDI key	`uddi:70a80f61-77bc-4821-a5e2-2a406acc35dd`
v2 UUID key	`uuid:70a80f61-77bc-4821-a5e2-2a406acc35dd`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes, Internal Validation Service

- `ntis-gov:naics:2002` - Business Taxonomy: Business Taxonomy: NAICS (2002 Release

v3 UDDI key	`uddi:uddi.org:ubr:categorization:naics:2002`
v2 UUID key	`uuid:1ff729f2-1948-46cf-b660-31ec107f1663`
Categorization	categorization
Compatibility	tModel businessEntity businessService bindingTemplate
Checked	yes, Internal Validation Service

- `unspsc-org:unspsc:3-1` - Product Taxonomy: UNSPSC (Version 3.1)

v3 UDDI key	`uddi:db77450d-9fa8-45d4-a7bc-04411d14e384`
v2 UUID key	`uuid:db77450d-9fa8-45d4-a7bc-04411d14e384`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	no

- `unspsc-org:unspsc` - Product Taxonomy: UNSPSC (Version 7.3)

v3 UDDI key	`uddi:unspsc-org:unspsc`
v2 UUID key	`uuid:cd153257-086a-4237-b336-6bdcbdcc6634`
Categorization	categorization
Compatibility	tModel, businessEntity, businessService, bindingTemplate
Checked	yes, Internal Validation Service

- `unspsc-org:unspsc:v6.0501` - Product and Service Category System: United Nations Standard Products and Services Code (UNSPSC)

v3 UDDI key	`uddi:uddi.org:ubr:categorization:unspsc`
v2 UUID key	`uuid:4614C240-B483-11D7-8BE8-000629DC0A53`
Categorization	categorization
Compatibility	tModel businessEntity businessService bindingTemplate
Checked	yes, Internal Validation Service

- `ws-i-org:conformsTo:2002_12` is a category system used for UDDI entities to point to the WS-I concept to which they conform.

v3 UDDI key	`uddi:65719168-72c6-3f29-8c20-62defb0961c0`
v2 UUID key	`uuid:65719168-72c6-3f29-8c20-62defb0961c0`
Categorization	categorization
Compatibility	tModel
Checked	no

WSM Taxonomies

The following taxonomies are used for integration with a web service management system:

systinet-com:management:metrics:avg-byte

Average sum of incoming and outgoing message length

v3 UDDI key	`uddi:systinet.com:management:metrics:avg-byte`
v2 UUID key	`uuid:3c13a2e2-dfd0-30a2-bd58-c5de8c2ae3bb`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:avg-byte-input

Average input message length per hour

v3 UDDI key	`uddi:systinet.com:management:metrics:avg-byte-input`

v2 UUID key	`uuid:f18a50ad-ddb2-392a-b97c-1181c67b2817`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:avg-byte-output

Average output message length

v3 UDDI key	`uddi:systinet.com:management:metrics:avg-byte-output`
v2 UUID key	`uuid:7664723d-896a-3ed2-b7e9-46c9f38e7681`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:avg-hits

Average message hits per hour

v3 UDDI key	`uddi:systinet.com:management:metrics:avg-hits`
v2 UUID key	`uuid:bf010bf9-cafa-3f68-bf51-3cde3bd0f483`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:avg-response-time

Average response time in milliseconds

v3 UDDI key	`uddi:systinet.com:management:metrics:avg-response-time`
v2 UUID key	`uuid:099d67a9-eae6-3c30-8be9-48b44c5d9728`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:errors

Count of application failures in the last hour

v3 UDDI key	`uddi:systinet.com:management:metrics:errors`
v2 UUID key	`uuid:b074de10-e781-383a-bd00-248a1c42f0fa`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:hits

Count of hits in the last hour

v3 UDDI key	`uddi:systinet.com:management:metrics:hits`
v2 UUID key	`uuid:720689a4-dce4-398c-adba-e5c0f50d1eb2`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:median-byte

Median sum of incoming and outgoing message lengths

v3 UDDI key	`uddi:systinet.com:management:metrics:median-byte`
v2 UUID key	`uuid:0adefd4c-7624-3973-91a5-ea4971d6b0ef`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:median-byte-input

Median value of incoming message lengths

v3 UDDI key	`uddi:systinet.com:management:metrics:median-byte-input`
v2 UUID key	`uuid:c9c2fd87-f806-3ca0-819e-3f788cc8fd95`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:median-byte-output

Median output message length

v3 UDDI key	`uddi:systinet.com:management:metrics:median-byte-output`
v2 UUID key	`uuid:bdb4e8f8-1aba-3558-b1f5-cf89b5455529`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:median-response-time

Median response time in milliseconds

v3 UDDI key	`uddi:systinet.com:management:metrics:median-response-time`
v2 UUID key	`uuid:62f08146-1d3f-30e3-8c6a-1f2062c332d4`
Categorization	categorization

Compatibility	tModel
Checked	no

systinet-com:management:metrics:policy-violations

Count of policy violations in the last hour

v3 UDDI key	`uddi:systinet.com:management:metrics:policy-violations`
v2 UUID key	`uuid:be42511a-3c68-34d2-b137-d00e56bb4de4`
Categorization	categorization
Compatibility	tModel
Checked	no

systinet-com:management:metrics:reference

Reference to a tModel containing all metrics about the service. The keyValues in keyedReferences that refer to this tModel must be a tModelKey of the metric tModel.

v3 UDDI key	`uddi:systinet.com:management:metrics:reference`
v2 UUID key	`uuid:0d709256-b9f3-30a3-9aa1-51a1adb11324`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:proxy-reference

WSM Proxy Reference Taxonomy

v3 UDDI key	`uddi:systinet.com:management:proxy-reference`
v2 UUID key	`uuid:79bf6f6d-b0b7-3f08-b45e-9893b525de9b`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:server-reference

WSM Server Reference Taxonomy.

v3 UDDI key	`uddi:systinet.com:management:server-reference`
v2 UUID key	`uuid:1583604a-57a2-3887-9b1d-2549e270390c`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:state

WSM State Taxonomy

v3 UDDI key	`uddi:systinet.com:management:state`
v2 UUID key	`uuid:73c7ef28-6150-36a0-ba82-414424ede582`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:state-change-request-type

WSM State Change Request Taxonomy

v3 UDDI key	`uddi:systinet.com:management:state-change-request-type`
v2 UUID key	`uuid:64473cda-4a78-3ddb-b0c6-801533ce1943`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:system

WS Management System Taxonomy

v3 UDDI key	`uddi:systinet.com:management:system`
v2 UUID key	`uuid:e148d85e-cc08-32f6-8f00-db85e258e511`
Categorization	categorization
Compatibility	bindingTemplate
Checked	no

systinet-com:management:type

WSM Type Taxonomy

v3 UDDI key	`uddi:systinet.com:management:type`
v2 UUID key	`uuid:5d14645d-66ea-39ac-8122-49d06b09b492`
Categorization	categorization
Compatibility	bindingTemplate
Checked	yes

systinet-com:management:url

Endpoint URL Taxonomy

v3 UDDI key	`uddi:systinet.com:management:url`
v2 UUID key	`uuid:4897f99b-bd23-3889-af37-b80351cf8b52`
Categorization	categorization
Compatibility	bindingTemplate
Checked	no

5.5. Registry Console Reference

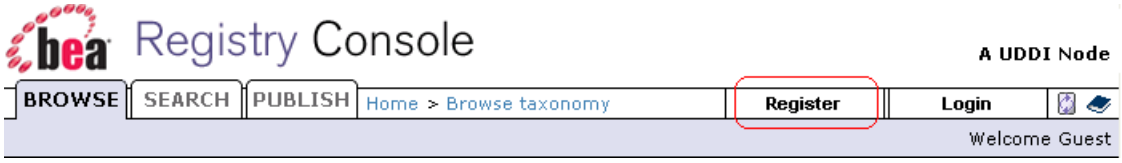
- Registry Console Overview
- Manage user account and user groups
- Browsing the registry;
- Searching the registry
- Publishing in the registry

5.5.1. Register/Create Account

Register

Before you can publish data to the registry, you must have a BEA AquaLogic Service Registry account. You can create an account via the web interface.

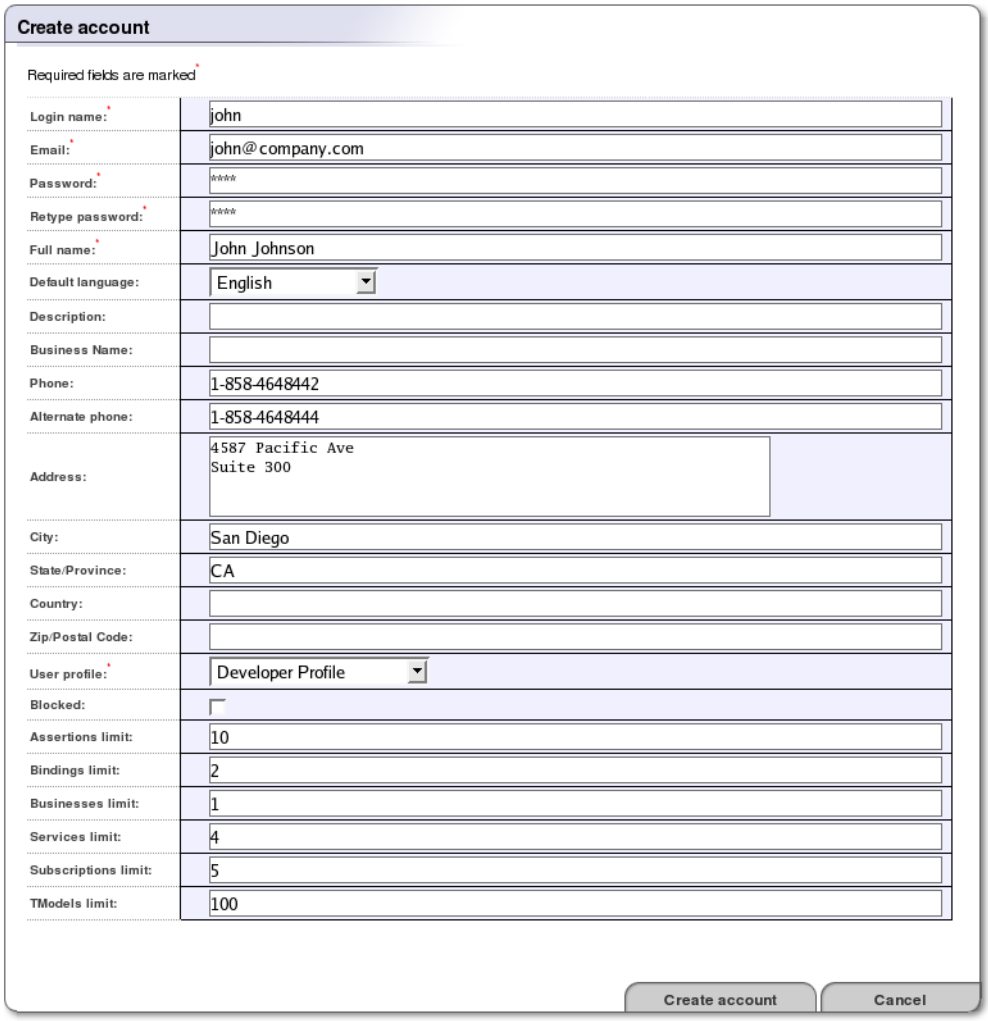
Figure 44. Register Account



Follow these steps to register a user account:

1. Click the **Register** link on the main Registry Console page. This returns the **Create account** page.
2. Fill in all fields. Those labeled with an asterisk (*) are required. Your email address may be used later for enabling your account.

Figure 45. Create Account



3. Click the **Create account** button.

The new account is now enabled.

Note

BEA AquaLogic Service Registry may be configured to require email confirmation in order to enable the user account. In this case, the registry sends an email confirmation. Follow the instructions in this email to enable your account.

Login

To log on, click the **Login** link on the upper part of the Registry Console, and enter your username and password.

Figure 46. Login Tab

Once logged into the registry, you are able to publish, delete, and update the various UDDI structures. Users have access to their own account information. Administrators also have account administration access; that is, the ability to delete and edit accounts and produce account audit reports.

5.5.2. Registry Console Overview

Registry Console is comprised of the following objects:

A: Main Menu Tabs

Browse

This tab allows you to browse UDDI entities using taxonomies.

Search

This tab allows you to search the registry. You can perform inquiry on UDDI entities, you can find business entity, service, bindings, tModels, and related businesses. The menu option also allows you to browse taxonomies and directly get information from BEA AquaLogic Service Registry when you know a key of UDDI data types (business, service, binding, and tModel)

Publish

This tab allows you to publish UDDI structures (businessEntities, businessServices, bindingTemplates, and tModels). On this tab, you can also assert relationships between business entities, subscribe interest in receiving information about changes made to a registry, transfer ownership of selected UDDI structures (Custody Transfer), and publish WSDLs to the registry.

Profile

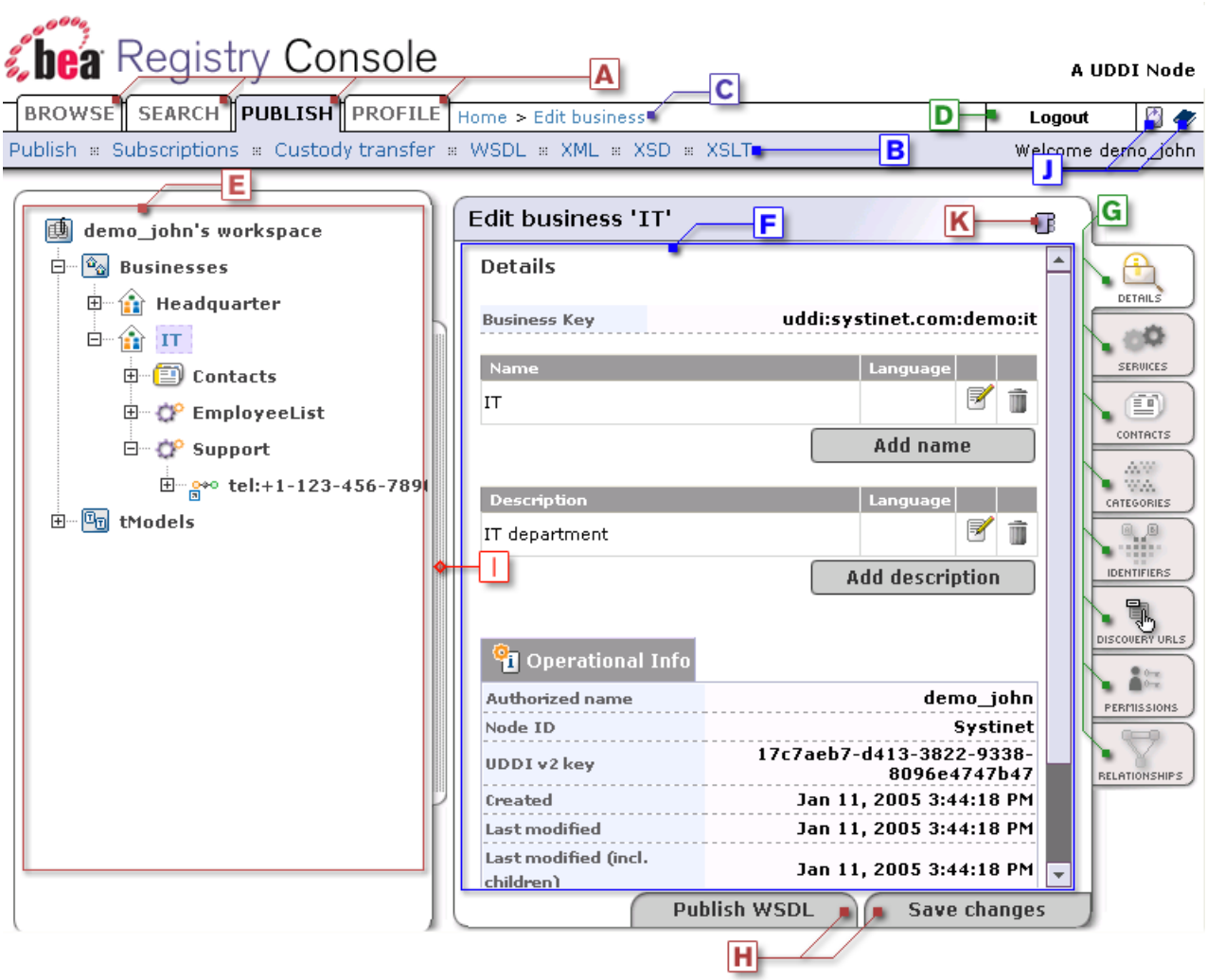
Here you can manage your user account properties, account groups and favorite taxonomies.

Manage

This tab is used by the BEA AquaLogic Service Registry administrator to perform management tasks. See Administrators Guide for more information.

B: Menu Bar Sub menu options are located here.

Figure 47. Registry Console Overview



C: History path (breadcrumbs) This area displays the log of your recent actions. You can return to any of these previous actions by clicking on the hyperlinks.

D: User Actions This area contains several control elements that enable a user to:

- Create an account
- Log On
- Log Out

E: Tree Display Area A tree of available objects displays whenever applicable. It is displayed when viewing a business entity and its child objects and when the user may want a hierarchical overview of the UDDI workspace (such as when publishing).

F: Main Display Area Information chosen from the tabs and the tree display is made available in the Main Display Area.

G: Display Tabs These tabs allow the user to control the main area's display based on information type. A plain listing of all business properties would be very long and very difficult to read. Dividing the properties into tabs reduces the amount of information and improves page readability. The displayed information changes with the context.

H: Action Buttons The action buttons allow you to perform operations on the contents of the main display.

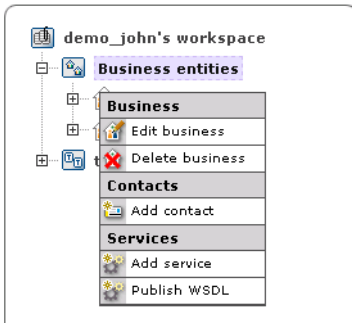
I: Resizing Bar This feature includes several enhancements to common web-based GUIs. If the content is too wide, you can drag and resize the tree window to fit all contents. The scrolling logic also applies to the main display area and makes it possible to keep all control elements of the interface on screen in any situation.

J: Action Icons There are two icons in this area. The first one allows you to refresh the page content, second one will open the product documentation page.

K: Action Icons Icons from this area allow you to switch on/off display tabs and open the current page in the printer friendly mode.

L: Context Menu The context menu displayed in Figure 48 is available by right mouse click on a node's icon in the tree display area.

Figure 48. Context Menu



For more information, please see Figure 47.

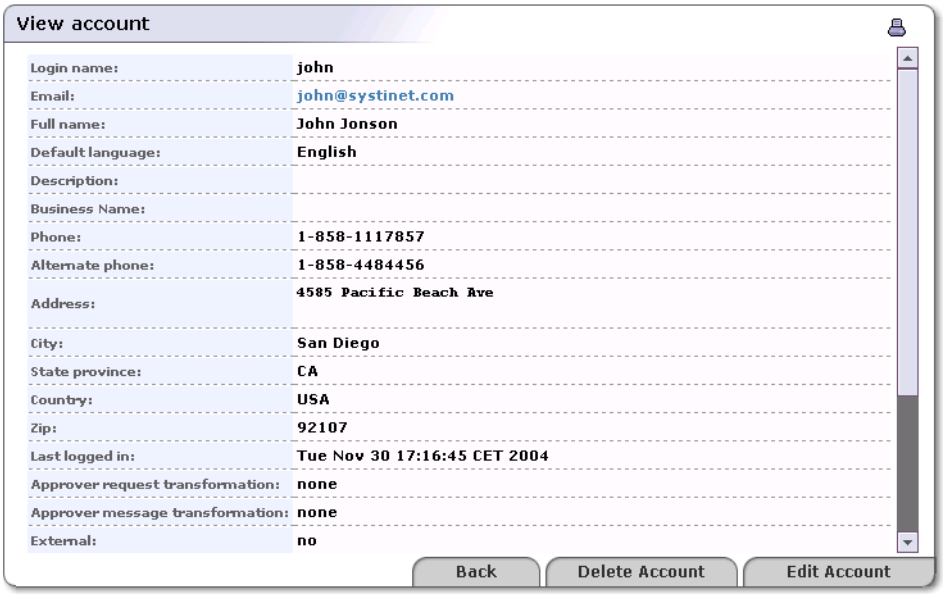
5.5.3. User Profile

You can manage your user account, user groups, and favorite taxonomies under the **Profile** menu tab.

Figure 49. Profile Menu Tab

To update your account properties, select **My account** and click the **Edit Account** button

Figure 50. View Account



Field descriptions (self-explanatory fields are omitted):

Default Language Code

Set the default language code. Used when publishing, it is the language code associated with a particular field when the language is not specified.

Use the following profile

Profile preference - Select your preferred predefined user profile from this drop down list

To maintain user groups, click the **Groups** link. From the Groups screen, you can:

- Create and manage your own groups
- Manage group membership

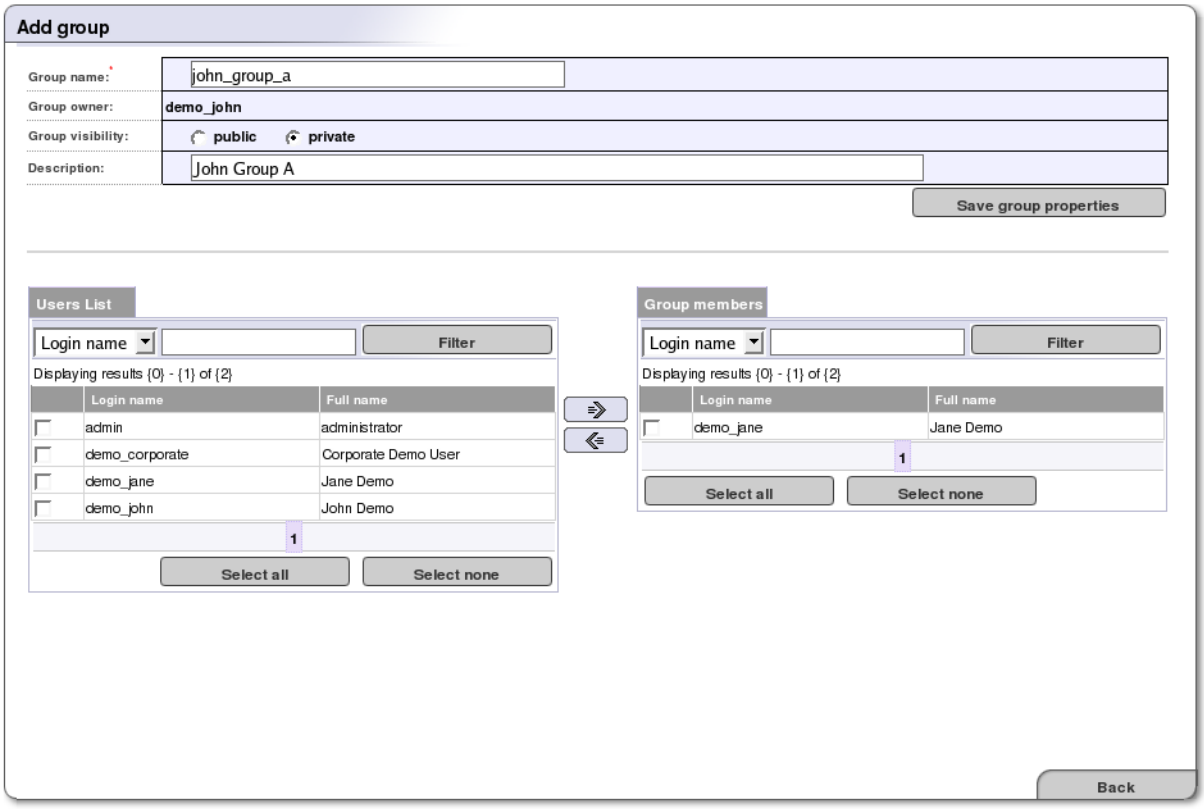
Figure 51. View User Groups

Create and Manage Groups

To create a new group:

1. Click on the **Profile** menu tab, and select the **Groups** link. This returns the Group list shown in Figure 51.
2. Click the **Add Group** button.

Figure 52. Edit Group Membership



3. In the edit box labeled **Group name**, type the name of your group.

4. Use the radio buttons labeled **public** and **private** to establish whether this group should be visible to all members (public) or visible only to the group owner (private).

5. Click **Filter** to display a list of the registry's users.

6. Check the boxes for all members you wish to include, then click the right-pointing arrow to move them to the **Group members** table.

7. Once users are added, click **Save Group** to update BEA AquaLogic Service Registry

Manage Group Membership

To add or remove members from a group:

1. Click on the **Profile** menu tab.

2. Click on the **Groups** link. This returns the Group list shown in Figure 51.

3. Click on the **Edit** button.

4. Use arrow buttons to add and remove users as shown in Figure 52

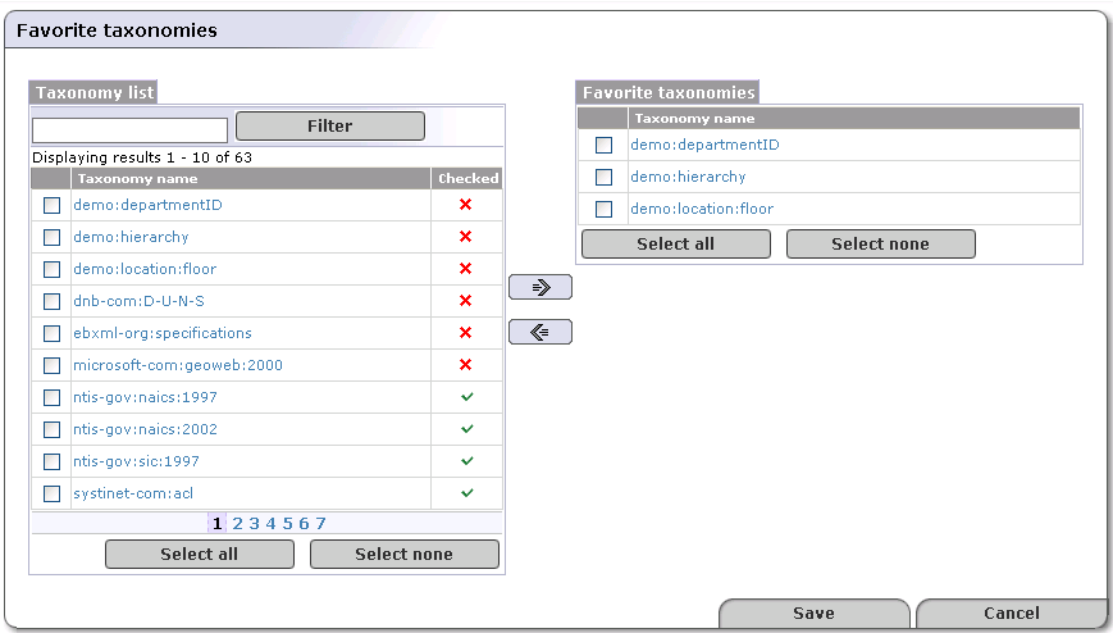
favorite Taxonomies

You can manage your favorite taxonomies under the **Profile** tab. You can define which taxonomies will be present in the list of your favorite taxonomies. Favorite taxonomies help you to search and categorize UDDI entities.

To manage your list of favorite taxonomies:

1. Click on the **Profile** menu tab. Click on the **favorite taxonomies** link. This returns the list of your favorite taxonomies shown in Figure 53.
2. Click **Filter** to search taxonomies by name.
3. Check the boxes for all taxonomies you wish to include, and click the right-pointing arrow to copy them to the favorite taxonomies table.
4. Once taxonomies are added, click the **Save** button to update the registry.

Figure 53. Manage favorite Taxonomies



5.5.4. Browsing

In this section, we will show you how to browse taxonomy structures to discover UDDI entities categorized or identified by taxonomies. You can also define a taxonomy filter and put your search criteria to a query. We present a demo data set that is installed with BEA AquaLogic Service Registry. This demonstration set is designed to help familiarize you with the registry.

To browse taxonomies and UDDI entities:

1. Click on the **Taxonomies** link under the **Browse** main menu tab.
2. The page shown in Figure 54 will appear.

Figure 54. Browse Menu Tab

On this page, you can use the drop down list to switch the taxonomy list to **favorite taxonomies**, **enterprise taxonomies**, and a defined **filter**.

Note

The **favorite taxonomies** option appears in the drop down list only if your list of favorite taxonomies is not empty. To add a taxonomy to your favorites, follow the direction in Section favorite Taxonomies. The list of enterprise taxonomies is defined by an administrator. For more information, see Section 1.5, Taxonomy Management in the Administrator's guide.

Initially, the **filter** contains all taxonomies except system taxonomies. Icons next to the drop down list serve to show/hide categorized entities, and show all/suppress empty categories.

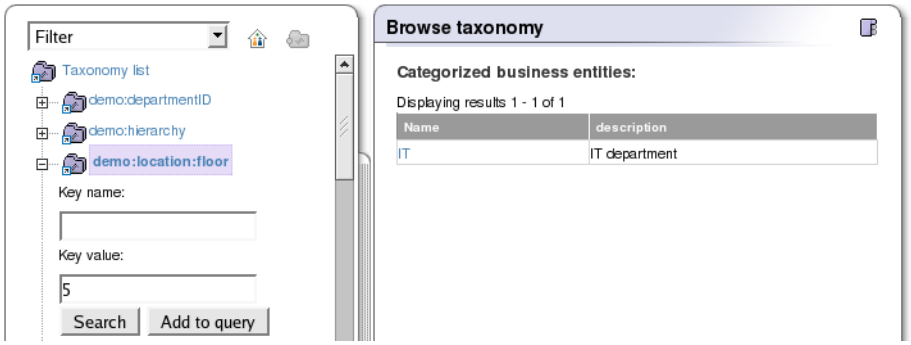
Drill down through the taxonomy tree to see all taxonomy categories. Those with sub-categories can be expanded and collapsed.

When you browse internally checked taxonomies you can see their value set to see UDDI entities categorized by these key values. For unchecked or externally checked taxonomies, you can search UDDI entities by key values. We will show you how to browse an unchecked taxonomy from the demo data.

To browse the demo data using **demo:location:floor** taxonomy:

1. Switch the drop down list shown in Figure 54 to the **filter** option.
2. Click on the **demo:location:floor** taxonomy. Expand the taxonomy by clicking on the plus sign in front of the taxonomy name. The key name and key value field pair appears.
3. Enter key value as 5, then click **Search** button.
4. You will get a list of UDDI entities categorized by this taxonomy with matching key value (IT in this case) as shown in Figure 55.

Figure 55. Browse Demo

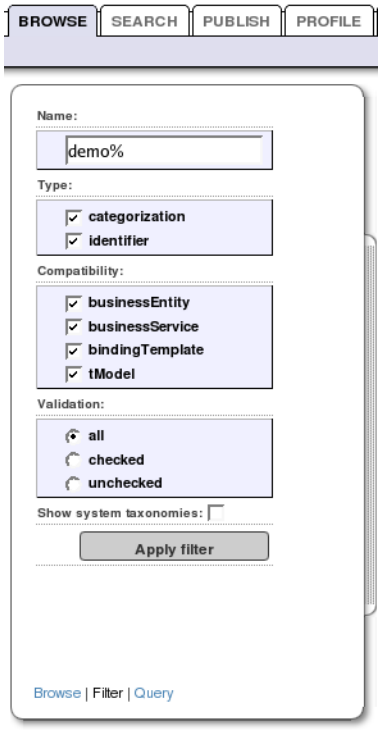


You can also add this search criterion to a query.

Define Filter

You can reduce the number of taxonomies in the taxonomy list by defining a taxonomy filter. To switch from taxonomy browsing to filter definition, click on the **filter** link in the lower left corner. The page shown in Figure 56 will appear.

Figure 56. Taxonomy Filter

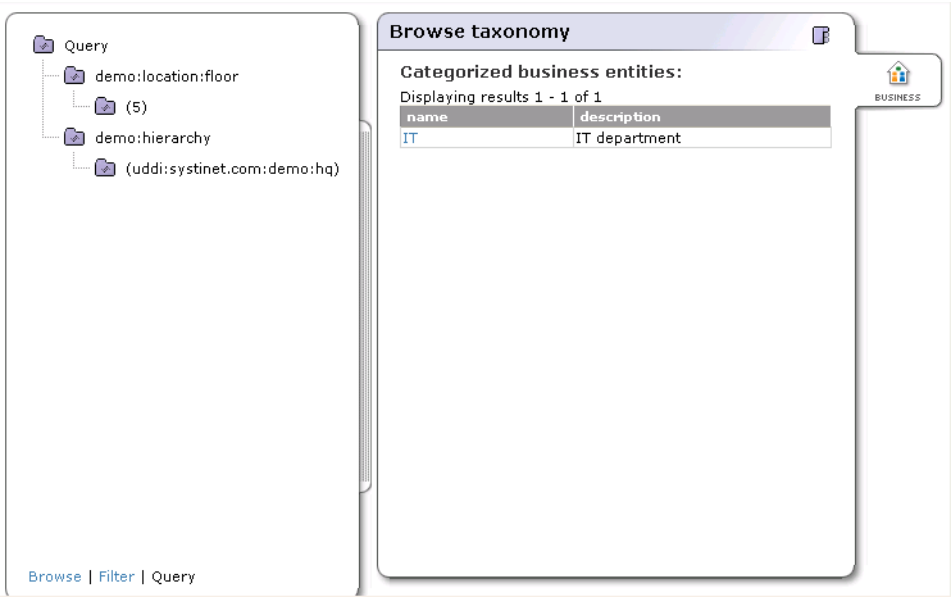


You can filter taxonomies by name using the wild card characters % and _. You can specify taxonomy type, compatibility, and a validation type. Once you define the filter criteria, click **Apply filter**. This will return you to the browse taxonomy page.

Define Query

You can also combine search criteria in a query. To add a search criterion to a query, use the button **Add to query** shown in Figure 55. Then, you can expand another taxonomy and specify a new criterion. The page shown at Figure 57 presents the query displaying business entities located on the 5th floor (demo:location:floor taxonomy) having Headquarter department as the superior department (demo:hierarchy taxonomy).

Figure 57. Query



To remove a category from the query, right-click on the query and select **remove from query** from the context menu.

Note

The query definition is not persistent. Once you leave the **Browse** menu tab, the query will disappear.

5.5.5. Searching

BEA AquaLogic Service Registry search function allows you to perform the following searches:

Find UDDI data structures

You can search for business entities, services, bindings, and tModels using names and categories in combination with find qualifiers including range queries.

- Find Business
- Find Services
- Find Binding
- Find tModel

Direct Get

You can retrieve data from BEA AquaLogic Service Registry when you know the key of the UDDI entity you want to retrieve.

Find Resources

You can search for resources:

- Find WSDL
- Find XML
- Find XSD
- Find XSLT

In the Search section, we present a demonstration data set that is installed with BEA AquaLogic Service Registry. This demonstration set is designed to help familiarize you with the registry.

Note

BEA AquaLogic Service Registry supports the use of wildcard characters. You can use both `%` and _.

Use `%` in place of any number of characters and spaces. For example, if you wish to find all business beginning with A, type `A%`. Use the underscore wildcard (_) in place of any single character. For example, to find Dan or Dane, type `Dan_`.

See Section Find Business by Categories how to use range queries functionality.

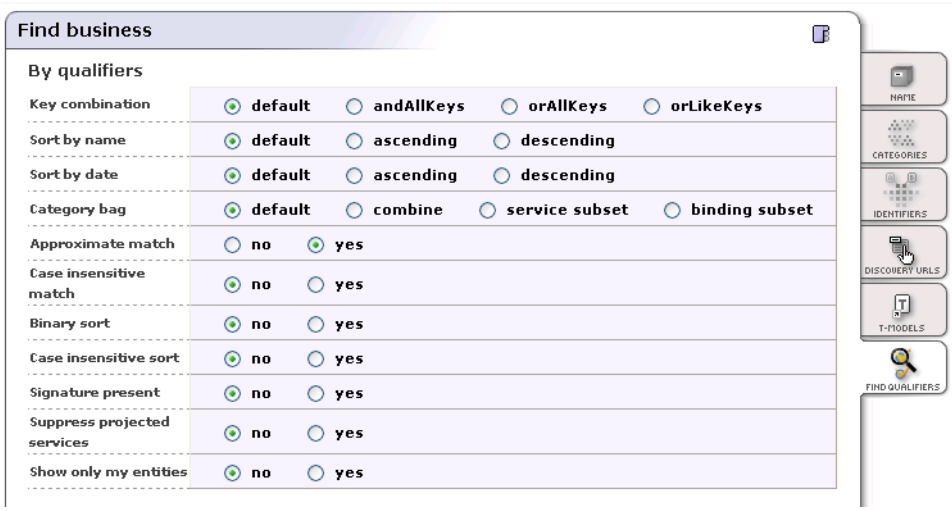
Find Business

In this section, we cover locating business entities using a number of different methods. You can locate business entities by:

- Name
- Categories
- Identifiers
- Discovery URL
- tModel

For each find method, you can specify qualifiers located on the **Find Qualifiers** tab of the **Search** panel.

Figure 58. Find Qualifiers



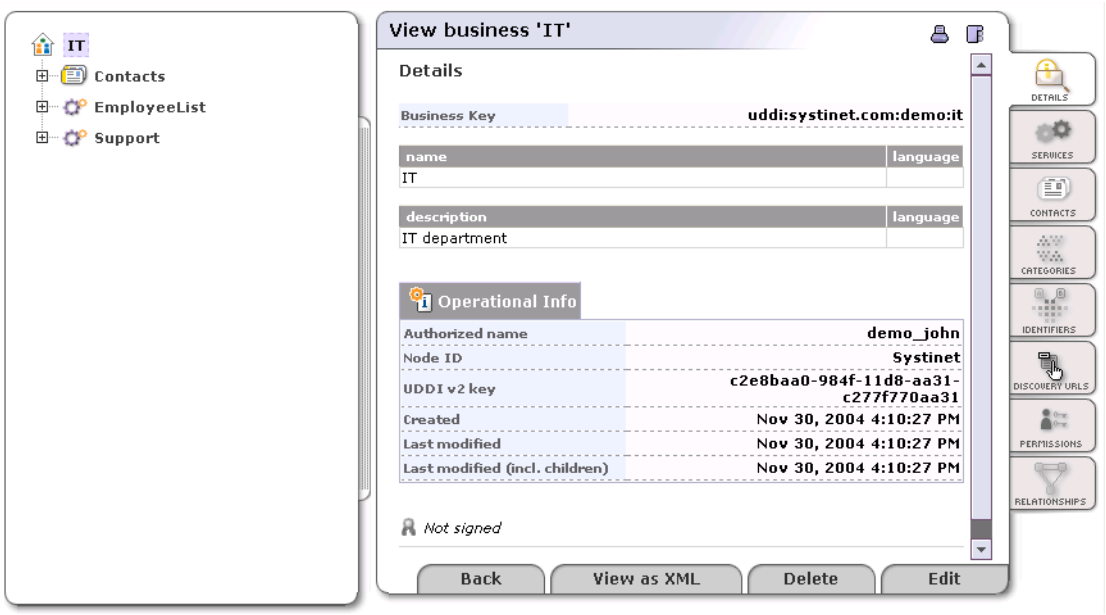
Find Business by Name

To find a business by name:

1. Under the main **Search** tab, click the **Businesses** link.
2. Click the **Add Name** button in the **Search** panel.
3. Type in the business name, such as `IT` from the pre-installed demo data. Then click the **Find** tab at the bottom right corner.

   To see all businesses, type the wildcard `%` and click **Find**.
4. The search result will appear on the **Results** panel. Click on the link with the business name, this opens the page shown at Figure 59.

Figure 59. View Business Detail



Find Business by Categories

In this section we will show you how to search for business entities by categories. We will use demo data to demonstrate how to find all departments located on specific floors. Also, an example how to use range queries will be shown.

To find a business by category:

1. Under the main **Search** tab, click the **Businesses** link

2. Click the **Categories** tab, then click the **Add category** button. This returns a list of available taxonomies.

   You can switch the **Show** drop down list from **favorite taxonomies** to see **all taxonomies**. To manage **favorite taxonomies** see Section 5.5.3, User Profile.

3. Click on the desired taxonomy.

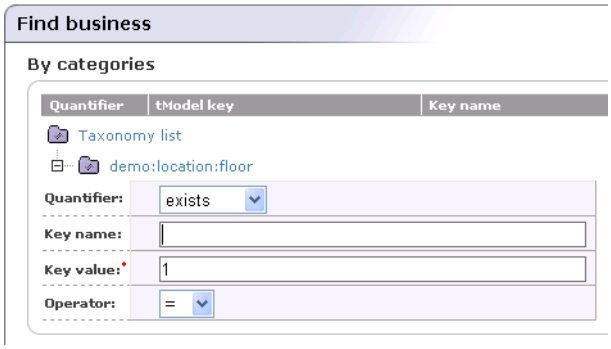
   The taxonomy is shown as a tree; its sub-branches include categories.

   Select `demo:location:floor` from our demo data.

4. Now you can enter **Key name** and **Key value**.

   Type `1` in the box labeled **Key value** and then click the **Add category** icon.

Figure 60. Find Business by Category

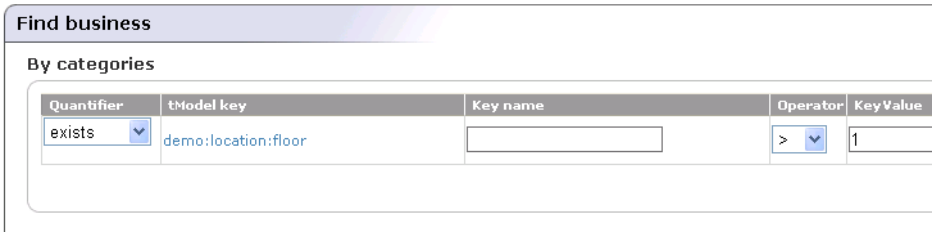


5. Once a category is added as your search criteria, click **Find**.

You will get the department with that is located on the first floor. If you want search for all departments located on higher floors you must use range queries functionality. We will continue with the previous search.

1. Click the tab **Search** to return to the Find business by categories page.
2. Click the **Edit category** icon. The page shown in Figure 61 is returned.

Figure 61. Find Business by Range Category



3. From the **Operator** drop down list, select the > operator, and click the **Update** icon.
4. Click **Find**. You will get all departments located higher than the first floor.

Find Business by Identifier

In this section we will show you how to find a business entity by identifier. We will use demo data to demonstrate how to find departments by their department number identifiers.

To find a business by identifier:

1. Under the main **Search** tab, click the **Businesses** link
2. Click the **Identifiers** tab. Then click the **Add identifier** button. This returns a list of available taxonomies.
3. Click on the desired taxonomy

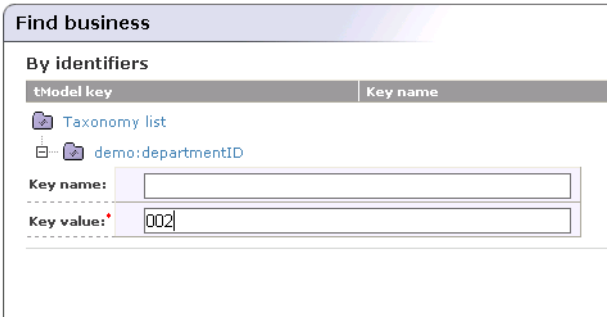
   The taxonomy is shown as a tree with its sub-branches including categories.

   Select `demo:departmentID` from the demo data.

4. Now you can enter **Key name** and **Key value**.

   Type `002` in the box labeled **Key value**, and click **Add identifier**.

Figure 62. Find Business by Identifier



5. Once the Identifier is added as your search criteria, click **Find**.

Find Business by Discovery URL

To find a business entity by discovery URL:

1. Under the main **Search** tab, click the **Businesses** link.
2. Select the **Discovery URLs** tab.
3. Type in the discovery URL and click **Find**.

Find Services

You can find services using a number of different methods including by:

- Name
- Category
- tModel

Search principles for finding services are the similar to those used for finding business entities.

Find Binding

You can find bindings using a number of different methods including by:

- Parent service
- Category
- tModel

The search principles for finding bindings are similar to those used for finding business entities.

Find tModel

You can find tModels using a number of different methods including by:

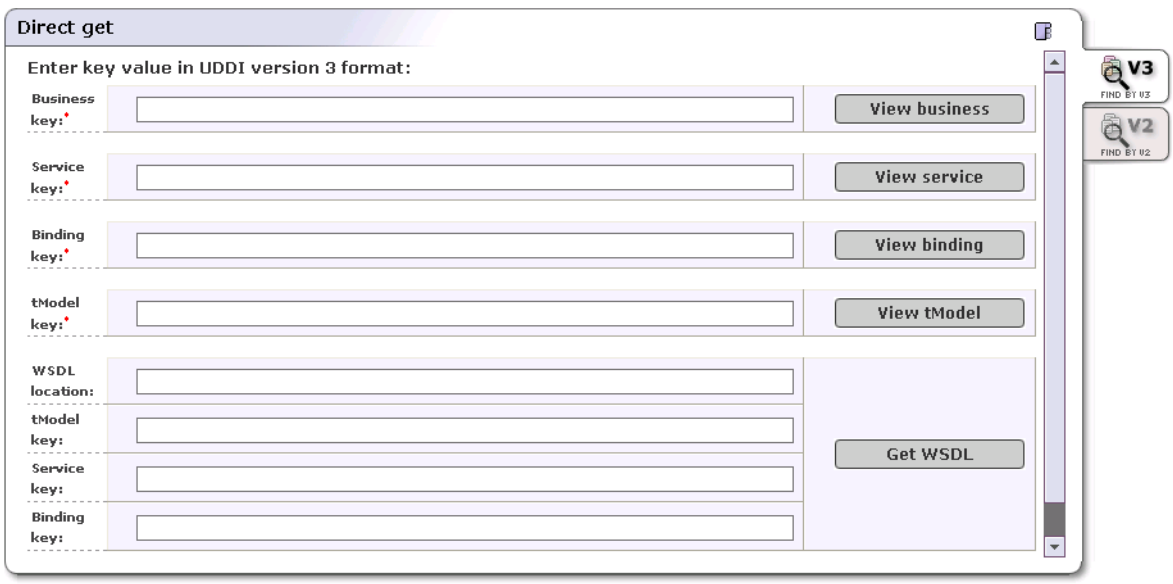
- Name
- Category
- Identifiers

The search principles for finding tModels are similar to those used for finding business entities.

Direct Get

You can also use **Direct get** from the **Search menu** tab to retrieve data from BEA AquaLogic Service Registry when you know the key of the UDDI structure you want to retrieve. BEA AquaLogic Service Registry allows you to specify keys for both UDDI version 2 and UDDI version 3. Click the **Find by v2** tab if you want to search using UDDI v2 keys.

Figure 63. Direct Get



Direct Get of XML Structures

You can also acquire the XML form of businesses, services, bindings, and tModels for use in automated processing by entering the key of the structure into a URI.

The form of the URI is:

```
http://<hostname>:<port>/uddi/web/directGetXml?<structureKey>=<key>
```

URI Examples Note that UDDI v3 is assumed by default.

- `http://localhost:8080/uddi/web/directGetXml?businessKey=uddi:systinet.com:uddinodebusinessKey`
- `http://localhost:8080/uddi/web/directGetXml?serviceKey=...`
- `http://localhost:8080/uddi/web/directGetXml?bindingKey=...`
- `http://localhost:8080/uddi/web/directGetXml?tModelKey=...`

Example with Login This URI includes username and password.

- `https://localhost:8080/uddi/web/directGetXml?businessKey=uddi:systinet.com:uddinodebusiness-Key&userName=admin&password=changeit`

Example with UDDI Version Specification Use this format when getting information associated with v1 and v2 structures.

- `http://localhost:8080/uddi/web/directGetXml?businessKey=8f3033d0-c22f-11d5-b84b-cc663ab09294&version=2`

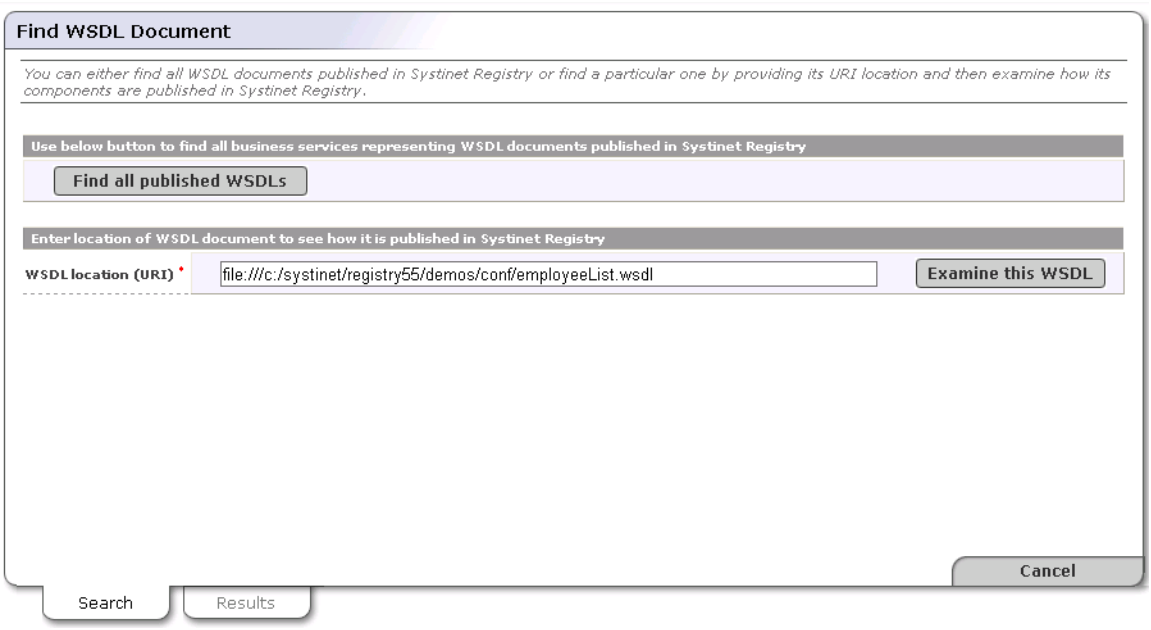
Find WSDL

You can find all WSDL documents published in BEA AquaLogic Service Registry. When you supply the WSDL location URI, you can review how artifacts of the WSDL document are published in BEA AquaLogic Service Registry. The following criteria: a WSDL document location, a tModel key, a business service key, and a binding template key can be used. To search for a WSDL document in BEA AquaLogic Service Registry:

1. Select the **Search** menu tab and click the **WSDL** link. The page shown in Figure 64 will appear.
2. Click the **Find all published WSDLs** button, or

   Enter **WSDL location URI** , then click **Examine this WSDL** button.

Figure 64. Find WSDL



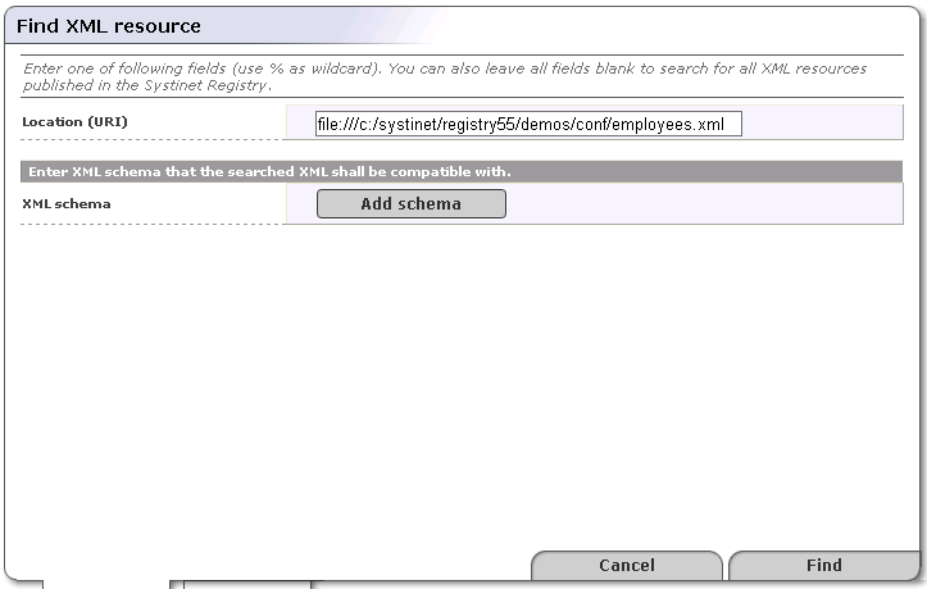
Find XML

You can search for an XML document in BEA AquaLogic Service Registry according to location URI of the XML document.

To search an XML document:

1. Select the **Search** menu tab and click the **XML** link. The page shown in Figure 65 will appear.
2. Enter a location and click **Find**.

Figure 65. Find an XML Document



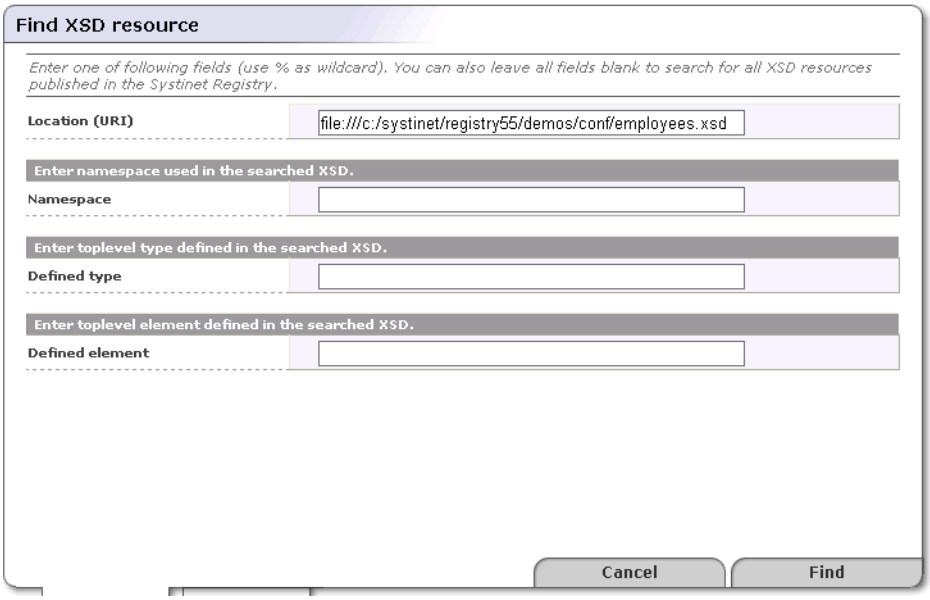
Find XSD

You can search for an XML Schema in BEA AquaLogic Service Registry according to location URI of the XML document.

To search an XML document:

1. Select the **Search** menu tab and click the **XSD** link. The page shown in Figure 66 will appear.
2. You can search by the location of the XML Schema document, namespaces, and by xsd:elements and xsd:types defined in the XML Schema document. Once you specify the search criteria, click **Find**.

Figure 66. Find XSD

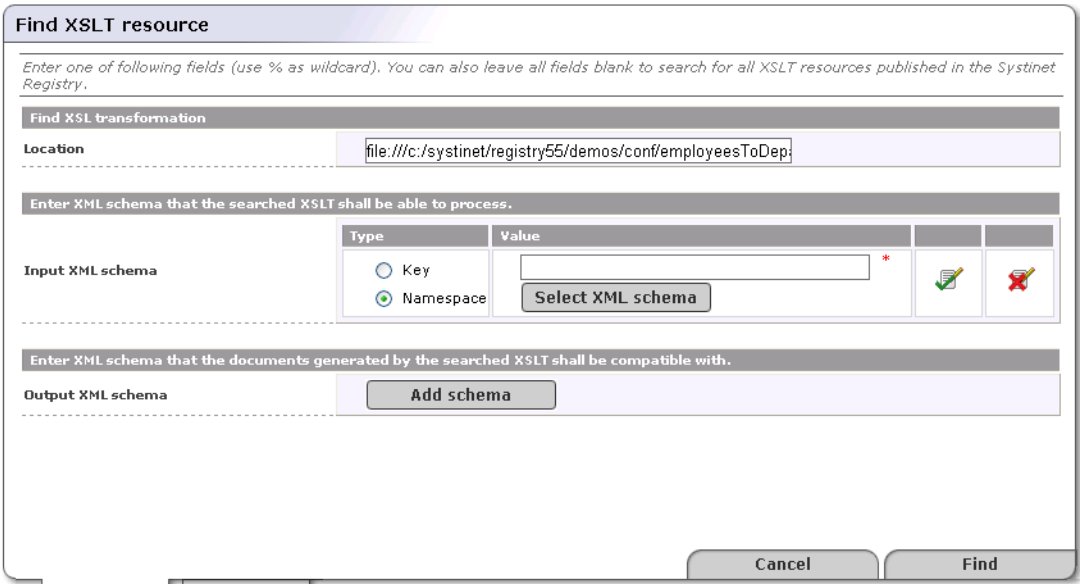


Find XSLT

To search an XSL transformation:

1. Select the **Publish** menu tab and click the **XML** link. The page shown in Figure 67 will appear.
2. You can enter the location of the XSLT. You can also search according to input and output XML schemas Search criteria for an XML schemas can be specified by tModel key or namespace. If you click on **Select XML Schema** you can specify additional criteria for the XML Schema, then select an XML Schema from the XML Schema list.
3. Before you click **Find**, click the **Update** icon if you specified to be search according to an XML Schema.

Figure 67. Find XSLT



5.5.6. Publishing

Publishing in BEA AquaLogic Service Registry has several components:

- Publish UDDI core structures:
  - Section Publishing a Business
  - Section Publishing a Service
  - Section Publishing a Binding Template
  - Section Publishing a tModel
  - Section Publishing Assertions - Asserting relationships between business entities.
- Section Publishing Subscriptions - Subscribing interest in receiving alerts regarding changes made to a registry.
- Section Publish Custody Transfer - Transferring ownership of selected UDDI structures.
- Publish Resources

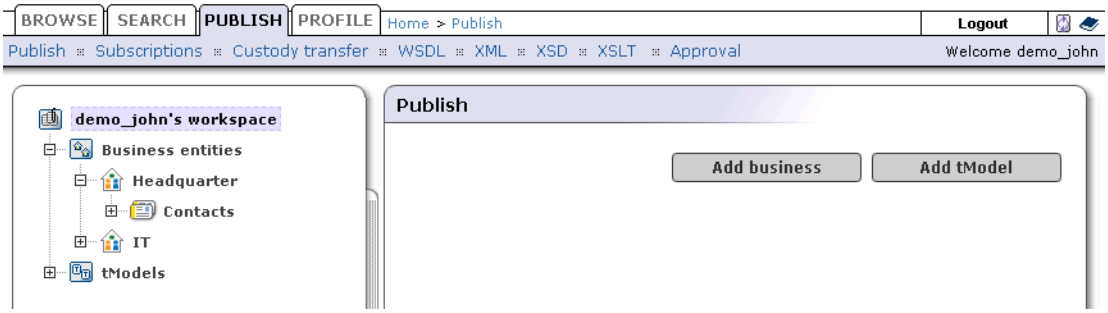
- Section Publishing WSDL Documents - Publishing Web Services Description Language documents (WSDL) to BEA AquaLogic Service Registry.
- Section Publish XML - Publishing XML Documents.
- Section Publish XSD - Publishing XML Schema Definition (XSD) Documents.
- Section Publish XSLT - Publishing Extensible Stylesheet Language Transformation (XSLT) Documents.

Note

You must be logged into BEA AquaLogic Service Registry to publish to it. There is a limitation of how many UDDI structures a user can store. See Administrator's Guide, Section Account Limits

The main Publish page is divided into two panels. The left panel displays UDDI data structures that belong to the logged-in user or to which this user has access permissions. The panel on the right displays details about the data structure selected in the left panel. As you can see, if no structures are selected, buttons for adding businesses and tModels are displayed.

Figure 68. Publish Page



Publishing a Business

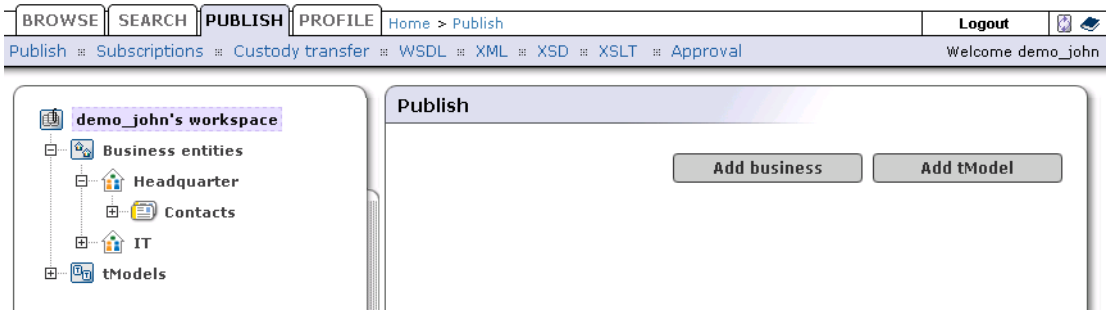
This section explains how to publish a businessEntity and edit businessEntity-related structures:

- Add business name and description
- Add Contact
- Add a Discovery URL
- Add a Category
- Add an Identifier
- Add Business Services
- Add Projected Services
- Assert Business Relationships

To publish a business:

1. Click the **Add Business** button in the right-hand panel of the publish page, or select **Add Business** from the context menu that appears when you right-click the **Business Entities** node.

Figure 69. Add Business



2. Enter the business name and a description, then click **Add Business**.

3. The business will appear in the left tree panel under the **Business entities** node

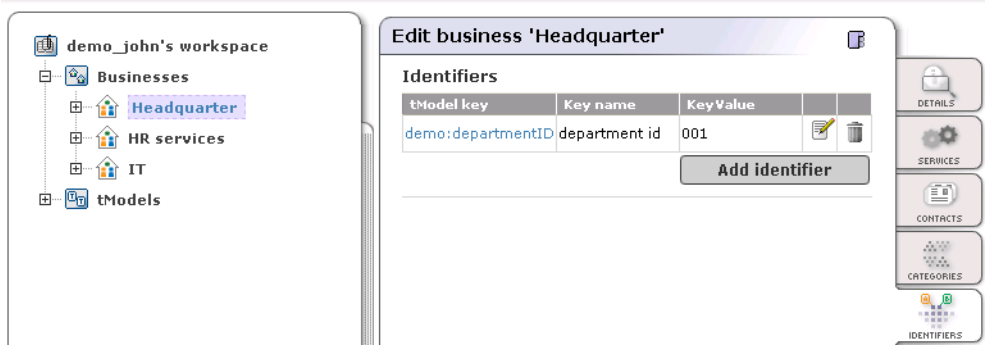
To edit a business entity:

1. Select the **Publish** menu tab.

2. Click the **Publish** link.

3. In the left tree panel, click on the business entity node you wish to edit.

Figure 70. Edit Business



4. After you modified the business entity, click the **Save changes** button.

Adding a Contact

The contact structure provides you with a space where you can list the people associated with the business entity. It is comprised of six properties: name, phone, email, address, description, and use type.

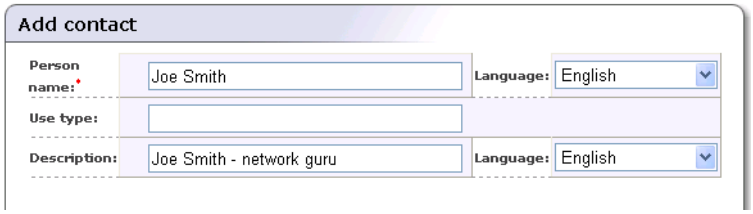
It is recommended that you use the description field to give a brief explanation of how the contact should be used.

Use types can be used to indicate the expected way in which the contact should be used. For example, "New Franchises", "Sales contact", "Technical Questions".

To add a contact:

1. On the **Contacts** tab of the Edit business or View business page, click the **Add contact** button. This displays the Add contact page where you can specify the contact's name and use type, as shown in Figure 71:

Figure 71. Add Contact



2. Click **Add contact**.

3. Build your lists of information for descriptions, phone numbers, and addresses. Each collection page, with the exception of Address collection, functions in the same manner. Click the **Add** button for the element you want to add. You will see two or more edit fields to be completed.

   Important

   Once the fields have been edited, you must click the **Update** icon on the right.

   For addresses, click the **Addresses** tab. On this tab, add, edit, or delete existing address structures by clicking through the appropriate buttons.

   When you add or edit an address, fill in the desired fields, add the data to your list, and click **Update** when finished.

4. Once you have updated all of the contact's information, click **Save changes** at the bottom of the Edit contact page. You will see the name and use type of your new contact entry in the contacts list.

Adding a Discovery URL

To add a Discovery URL:

1. On the Edit business page click on the **Add discovery URL** button at the bottom of the **Discovery URLs** tab.

2. Complete the **Discovery URL** and **Use Type** edit fields with the relevant data.

3. When the fields are complete, click **Update** on the right to add this information to the list.

4. Click **Save changes**

Adding a Category

With categories you can make your business more visible to searches by associating it with a number of accepted taxonomies. These taxonomic categories identify a business and its services by location, product or service line, and industry.

BEA AquaLogic Service Registry comes with keys for three basic checked taxonomies by default: These are the `ISO 3166` geographical classification system and the `NAICS` and `SIC` industry and product classifications.

A key is also provided for `Microsoft GeoWeb 2000`, but as this is an unchecked taxonomy, key names and key values must be entered by hand.

To add a category to your list:

1. On the **Categories** tab of the Edit business page, click the **Edit** button. If there are already categories associated with this business entity, a list of them will be returned along with the **Add category** button. Otherwise, only the button will be displayed.

2. Click the **Add category** button beneath the **Categories** tab. This returns a list of available taxonomies from which you can choose categories to add to the list.

3. Click on an available taxonomy. Checked taxonomies will expand to a tree of categories valid for that model. You can type a known key name in the search box for faster retrieval. Note that larger branches are limited to ten items per page.

4. You can also search for the name of the taxonomy through the search box at the top of the taxonomy form. Use the **starts with**, **contains**, and **exact match** radio buttons as necessary. Like standard wild cards, these buttons search for the entered string as specified. For example, The pattern `Cana`, when used with the **starts with** button and a geographic taxonomy, returns the set {"Canada" "Canarias"}. The result set is limited to a maximum of 250 items.

   Note

   If you provide too broad a search pattern, the resulting list will be truncated to 100 items.

   With unchecked taxonomies (for example, Microsoft's `GeoWeb` taxonomy), it is possible to supply the key name and value through edit fields.

5. To add multiple categories, for example Albania and Armenia from the `uddi-org:iso-ch:3166:1999` taxonomy, check the boxes to the right of those key names, and click **Add category**. If you would like to add categories from different pages, you must click **Add category** on the first page before continuing to the next page containing your selections. For example, to choose Albania and Kazakhstan:

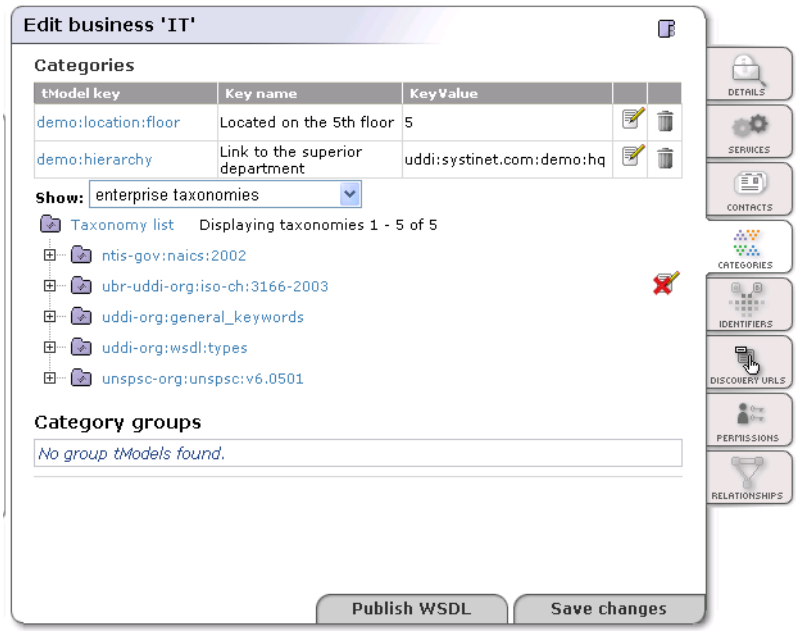
   a. Select Albania and click **Add category**.

   b. Click **Add category** on the Find service page.

   c. Click the link for page 8 on the expanded Find service page.

   d. Check the box next to Kazakhstan and click **Add category**.

Figure 72. Add Category



6. When you find the taxonomic classification you want, click the **Add category** button for checked taxonomies. For unchecked taxonomies, click **Add category** once the edit fields have been completed.

Adding an Identifier

You can also make your organization more visible by supplying any of your public or private identifiers, such as D-U-N-S, Tax, or Geographical Locator numbers to the registry. UDDI identifier structures are composed of the following elements:

tModel Key
: Identifies a namespace or service in which the key name and key value have significance

keyName
: The name or description of the key being used

keyValue
: The value of the key

To add an identifier to your list:

1. On the Edit business page, switch to the **Identifiers** tab.
2. Click the **Add identifier** button at the bottom of the Identifiers list.
3. Choose the identifier type from the displayed list of available taxonomical tmodels. This returns a field in which you enter key names and key values.
4. When you have filled in the fields, click the **Add identifier** button to the right to add the new identifier to the list.

Important

If you use a tModel for a checked identifier, the key value must be of a recognizable form and value. For example, if you want to use a `uddi-org:isReplacedBy` key, you must supply the valid business entity UUID key in the

keyValue field. Failure to do so will generate an error when you attempt to submit your business data to the database.

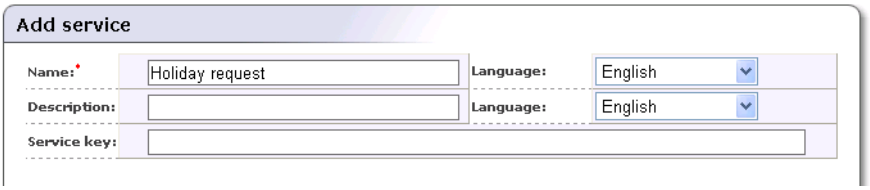
Publishing a Service

To publish a service:

1. Select the **Publish** menu tab and click the **Publish** link
2. In the left panel, click on the business to which you want to add a service. The right display area will show business details.
3. Select the **Services** tab, and click the **Add Service** button.

   Alternately, right-click on the business node to which you want to add a service, and select **Add Service** from the context menu.

   Figure 73. Add Service



4. Enter the service name and description and click **Add service**.

   The service is added to the left panel tree.

Publishing a Binding Template

Once you have declared and defined a business service, you must establish how current and potential business partners can access that service, a technical description of the service including where it can be found. This is accomplished through bindingTemplates.

A bindingTemplate represents a Web service instance where you obtain (among other things) the access point of an instance of the parent business service. Every bindingTemplate has a unique bindingKey for identification. (An access point contains contact information such as a URL, email address, or telephone number used to locate the service.)

The AccessPoint in a bindingTemplate structure can contain a URL of the endpoint of the web service. If there is more than one businessEntity that provides the same business service we recommend you reuse this information in a bindingTemplate. Create a bindingTemplate on the businessService that holds technical information. Other businessServices should contain bindingTemplates with accessPoints containing the key of the first technical bindingTemplate. These accessPoints should also contain useTypes with the value `hostingRedirector`.

Note

Alternatively, reference to another bindingTemplate can be stored in a hostingRedirector structure instead of in an accessPoint. However the hostingRedirector structure (not the hostingRedirector value of useType) is a relic of UDDI v2 and is deprecated in UDDI v3.

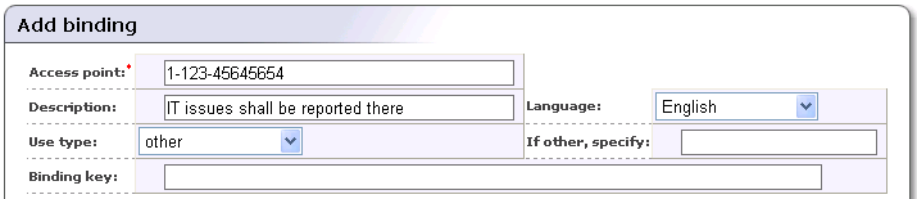
To add a bindingTemplate:

1. Select the **Publish** menu tab and click the **Publish** link

2. In the left panel, click on the service to which you want to add a binding. The right display area will show service details. Select the **Bindings** tab and click the **Add Binding** button.

   Alternatively, right-click the service node to which you want to add a binding, and select **Add Binding** from the context menu.

Figure 74. Add Binding



Publishing a tModel

The tModel is a structure that takes the form of keyed metadata (data about data). In a general sense, the purpose of a tModel within BEA AquaLogic Service Registry is to provide a reference system based on abstraction. Among the roles that a tModel plays in UDDI is the ability to provide and to describe compliance with a specification or concept, to a taxonomy, for example.

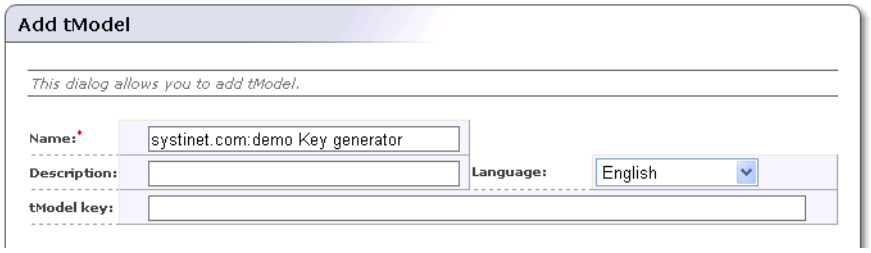
To publish a tModel:

1. Select the **Publish** tab, and click the **Publish** link.

2. On the right Publish panel, click the **Add tModel** button.

   Alternatively, right-click on the tModels node in the left panel and select **Add tModel** from the context menu.

Figure 75. Add tModel



3. Enter tModel name and description, and click the **Add tModel** button.

Note

If you delete an unused tModel, the tModel will be deleted from the database. The BEA AquaLogic Service Registry Administrator can change this behavior that tModels will be only marked as deleted. See Administrator's Guide, Section 2.7, Node.

Adding a Category

In this section we will show you how to assign `demo:location:floor` taxonomy to the numeric ordering as show at Figure 42.

1. Log on as `demo_john` user. (password is the same as the username).
2. Click the **Publish** tab in the main menu. Click on the tModel `demo:location:floor` item in the tree in the left part of the page. Edit tModel 'demo:location:floor' page will appear.
3. Click **Add category** button. A taxonomy list will appear.
4. Select the taxonomy `systinet-com:isOrderedBy`, enter **Key value** `uddi:systinet.com:comparator:numeric`.
5. Click the button **Add category** , then **Save changes** button.

Publishing Assertions

You can assert relationships that businesses under your BEA AquaLogic Service Registry custody have with others under your custody or with those under the custody of another user registered at the same operator node. The success of the latter assertion depends upon the approval of the user to whom the assertion is made.

When making an assertion you must supply:

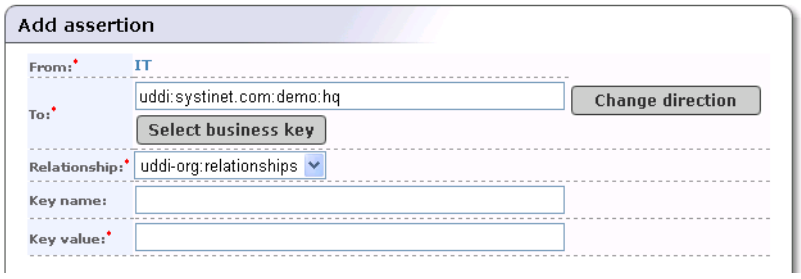
- The identity of the business from which the assertion is being made
- The identity of the business to which it is making a claim. BEA AquaLogic Service Registry specifies these business identities through their UUID keys.
- A reference explaining the nature of the relationship. References about the nature of the asserted relationship are derived from your own tModels or from the `uddi-org:relationships` tModel.

Adding an Assertion

To add a new assertion:

1. On the Edit business panel, switch to the **Relationships** tab. This displays the Relationship assertions page. If you have already set assertions you will see a list of those previously published. If not, you will see the message "No assertions found."
2. Click the **Add new assertion** button to display the Add assertion page shown in Figure 76.

Figure 76. Add Assertion



3. If the business for which you are making an assertion will assume the "To" role, click the **Change Direction** button.
4. Find the business with which you want to assert a relationship in the same way you would on the inquiry side of UDDI. The difference is that, along with the business name, you will see the business descriptions in the retrieved record set and a **Select business key** icon next to each record.

When you locate the target business among the records, click its **Select business key** icon. This returns you to the Add assertion page with the UUID key of the selected business as the previously missing role.

Important

A Keyed Reference will be required for the assertion to be valid. Click the **Set** button on the right of the Keyed Reference line. The Set keyed reference page displays.

5. Locate a tModel for your reference in the same way you would on the inquiry side of UDDI. The difference is that there are edit fields for Key Names and Key Values next to the tModel names and a **Set** button at the end of each row. Pertinent tModels include `uddi-orgs:relationship` and those you have published yourself.

   a. Enter the key value and the key name or description. For `uddi-orgs:relationship`, the key value may be `parent-child`, `peer-peer`, or `identity`.

   b. Click the **Set** value. This returns you to the Add assertion page. The tModel, key name, and key value added to the Keyed Reference record are displayed there.

6. Click the **Add assertion** button.

7. If the assertion is made to a business of which you have custody, the assertion will be completed automatically. If it is made to a business in the custody of another user, that user will need to review the assertion and complete it through his or her own account. This process is described below.

Accepting an Assertion

Assume that you have been notified by a parent company, a subsidiary, a peer, or a cooperative member that they have asserted a relationship with your company. Now you must review that assertion and, if you are in agreement, complete it.

To accept the assertion:

1. On the Edit business page, switch to the **Relationships** tab.

2. View the incomplete assertions made toward your business in the **Requested assertions** list. Each assertion will have a **Complete assertion** button next to its status message.

3. Click the **Complete assertion** button to accept the assertion.

4. If you wish to refuse, leave the assertion incomplete by omitting step 3. Return to the Publisher assertions page by clicking the link at the top of the page. Contact the business making the assertion to resolve the details of your relationship. Incomplete assertions will not appear when users query for related businesses.

Publishing Subscriptions

Subscriptions give you the ability to register interest in receiving information about changes made to BEA AquaLogic Service Registry. It allows the monitoring of new, changed, and deleted UDDI structures. Each subscription has a filter that limits the subscription scope to a subset of registry entities.

You can establish a subscription based on a specific query or set of entities in which you are interested. Query-based subscriptions notify the user if the result set changes within a given time span; entity-based subscriptions notify the user if the contents of the specified entities change.

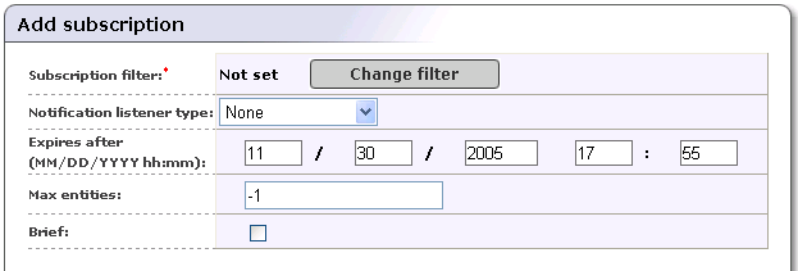
Subscriptions enable:

- notification of the registration of new businesses or services
- monitoring of existing businesses or services
- acquiring registry information for use in a private registry
- acquiring data for use in a marketplace or portal registry

This filter should be one of the following ordinary UDDI inquiry calls:

- find_business
- find_relatedBusinesses
- find_service
- find_binding
- find_tModel
- get_businessDetail
- get_serviceDetail
- get_bindingDetail
- get_tModelDetail

Figure 77. Add Subscription



Adding Subscriptions

To add new subscription:

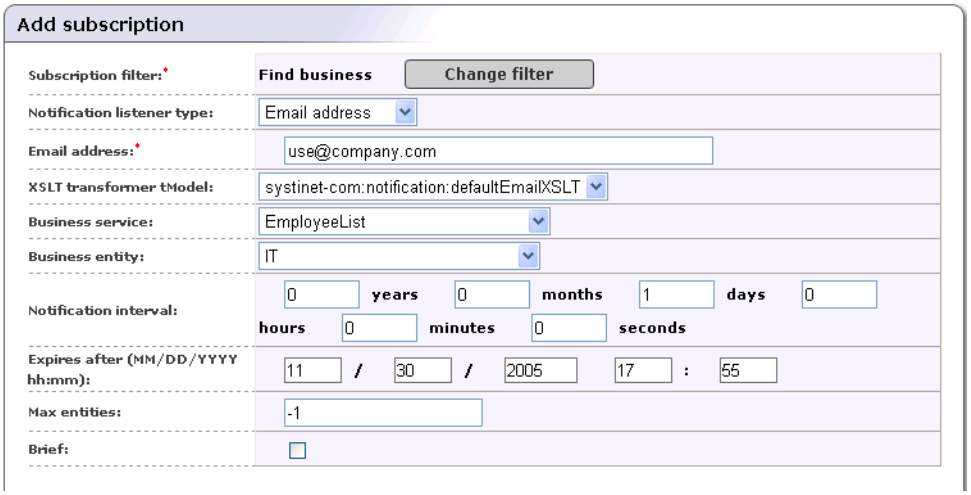
1. Click on the **Subscriptions** link under the **Publish** menu tab to display the Subscriptions page.
2. Click the **Add subscription** button to display the Add subscriptions page shown in Figure 77.
3. Click **Change filter** to specify a filter for your subscriptions. This returns the **Subscription filter type** page.
4. Select the filter type from the drop down list labeled **Subscription filter type**.
5. Click **Select filter**.
6. Set the filter properties in the same way you would for ordinary search calls.
7. Click the **Preview results** button to check filter results.

8. Click **Save filter** to return to the page with the filter settings shown in Figure 77.

9. Fill in the other subscription fields if needed. These are described below.

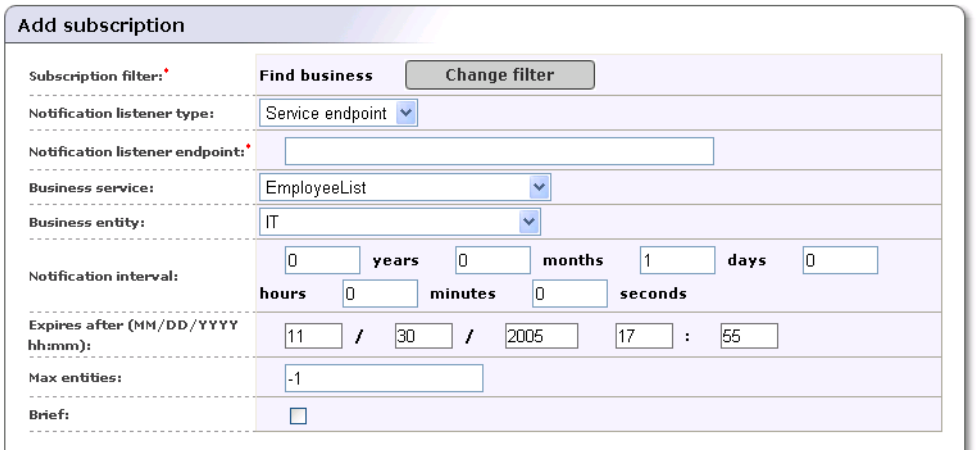
Notification Listener Types

Figure 78. Add Subscription - Email Notification Listener Type



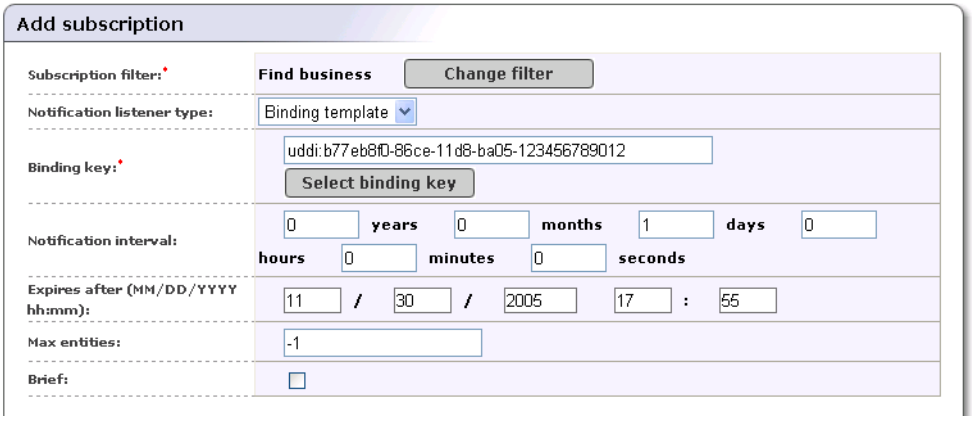
- **Subscription filter** - Specifies on which UDDI structure change the notification will occur.
- **Notification listener type** - Select notification listener type
  - Email address
  - Service endpoint
  - Binding template
- **Email address** - Email address to which notifications will be sent
- **XSLT transformer tModel** - tModel that references XSLT
- **Business service** and **Business entity** - Business service and business entity to which the bindingTemplate representing the notification listener service will be saved. These drop down lists lists only business entities and business services under which you have the permission to create the binding template.
- **Notification interval** - Specifies how often change notifications are to be provided to a subscriber. Required only for asynchronous notifications.
- **Expires after** - Specifies the period of time for which the administrator would like the subscription to exist.
- **Max entities** - Contains the maximum number of entities in a notification returned to a subscription listener.
- **Brief** - Controls the level of detail returned to a subscription listener.

Figure 79. Add Subscription - Service Endpoint Listener Type



- **Subscription filter** - Specifies on which UDDI structure change the notification will occur.
- **Notification listener type** - Select notification listener type here.
  - Email address
  - Service endpoint
  - Binding template
- **Notification listener endpoint** - URL to which the notification will be sent
- **Business service** and **Business entity** - business service and business entity to which the bindingTemplate representing the notification listener service will be saved. These drop down lists lists only business entities and business services under which you have the permission to create the binding template.
- **Notification interval** - Specifies how often change notifications are to be provided to a subscriber. Required only for asynchronous notifications.
- **Expires after** - Specifies the period of time for which the administrator would like the subscription to exist.
- **Max entities** - Contains the maximum number of entities in a notification returned to a subscription listener.
- **Brief** - Controls the level of detail returned to a subscription listener.

Figure 80. Add Subscription - Binding Template Listener Type



- **Subscription filter** - Specifies on which UDDI structure change the notification will occur.
- **Notification listener type** - Select notification listener type here.
  - Email address
  - Service endpoint
  - Binding template
- **Binding Template** - The bindingTemplate representing the notification listener service.
- **Notification interval** - Specifies how often change notifications are to be provided to a subscriber. Required only for asynchronous notifications.
- **Expires after** - Specifies the period of time for which the administrator would like the subscription to exist.
- **Max entities** - Contains the maximum number of entities in a notification returned to a subscription listener.
- **Brief** - Controls the level of detail returned to a subscription listener.

Editing Subscriptions

To edit an existing subscription:

1. Click on the **Subscriptions** link under **Publish** menu tab to display the Subscriptions page.
2. Click the **Edit** button beside the subscription you want to edit. This returns the Edit subscription page. Here you can edit all subscription arguments except Subscription filter.

Deleting Subscriptions

To delete subscription:

1. Click on the **Subscriptions** link under **Publish** menu tab to display the Subscriptions page.
2. Check the boxes beside subscriptions you want to delete.
3. Click the **Delete selected** button. This returns a confirmation page.

4. The confirmation page contains a list of subscriptions marked for deletion. If it is correct, press the **Yes** button to delete subscriptions permanently.

Publish Custody Transfer

Custody transfer is a service used to transfer ownership of a selected structure (business entity, business service, binding template or tModel) from one user to another. It consists of two steps: selecting structure(s) to transfer and generating a custody transfer token. When the potential new owner receives the transfer token (by a secure transport such as encrypted email), that user may accept or reject the custody transfer.

Important

This token must be kept secret, as it is sufficient information to transfer custody of the structure to any user!

If you decide to cancel the request (for example the transfer token has been compromised), use the **Discard transfer token** button.

Requesting Custody Transfer

To request custody transfer:

1. Click on the **Custody** link under **Publish** menu tab to display the Custody transfer page.
2. Click the **Request transfer token** link. This returns a list of UDDI data structures you own.
3. Check the box next to the UDDI structure(s) you wish to transfer, and click **Request transfer token**.
4. The next page will generate the transfer token. Copy the text of the transfer token to a file and send this file to the user who shall become the new owner of selected structures. Keep the token secret, as anyone who knows it can use it to transfer custody of that structure. Unencrypted email, for example, is not good data transfer choice.

Accepting Custody Transfer

To accept custody transfer:

1. Click on the **Custody** link under **Publish** menu tab to display the Custody transfer page.
2. Click on the **Transfer custody** link.
3. Open the file with the transfer token, copy its contents to clipboard and paste it to the edit area on the **Transfer structures** page.
4. Click **Transfer** button.

Publishing WSDL Documents

BEA AquaLogic Service Registry WSDL to UDDI (WSDL2UDDI) mapping is compliant with OASIS's technical note Using WSDL in a UDDI registry Version 2.0 [http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v200-20031104.htm]. It enables the automatic publishing of WSDL documents to UDDI, enables precise and flexible UDDI queries based on specific WSDL artifacts and metadata, and provides a consistent mapping for UDDI v2.

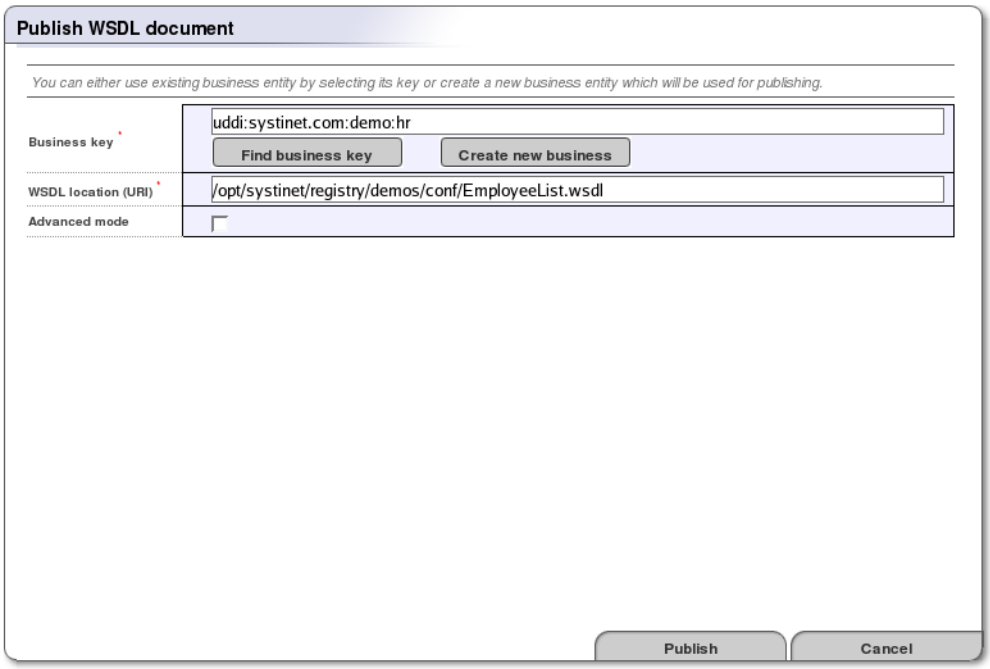
Publish WSDL

To publish a WSDL document:

1. Click on the **WSDL** link under the **Publish** main menu tab.

2. The page shown at Figure 81 will appear.

Figure 81. Publish WSDL



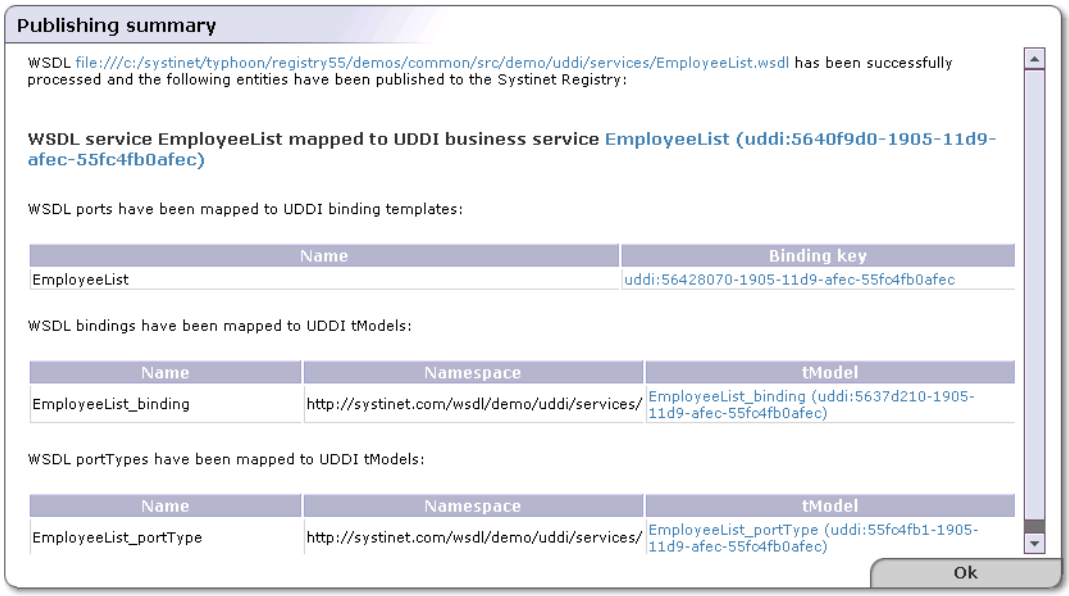
3. Enter the **Business key** of the business where services from WSDL document will be published. You can find a business key by clicking on the **Find business key** button.

4. Enter a **WSDL location**. You can try the WSDL document from BEA AquaLogic Service Registry demos from `REGISTRY_HOME/demos/conf/employeeList.wsdl`.

5. Leave the **Advanced mode** check box unchecked, then click **Publish** button.

The WSDL document will be published to BEA AquaLogic Service Registry. You can review how WSDL artifacts of the document have been mapped to BEA AquaLogic Service Registry at Figure 82.

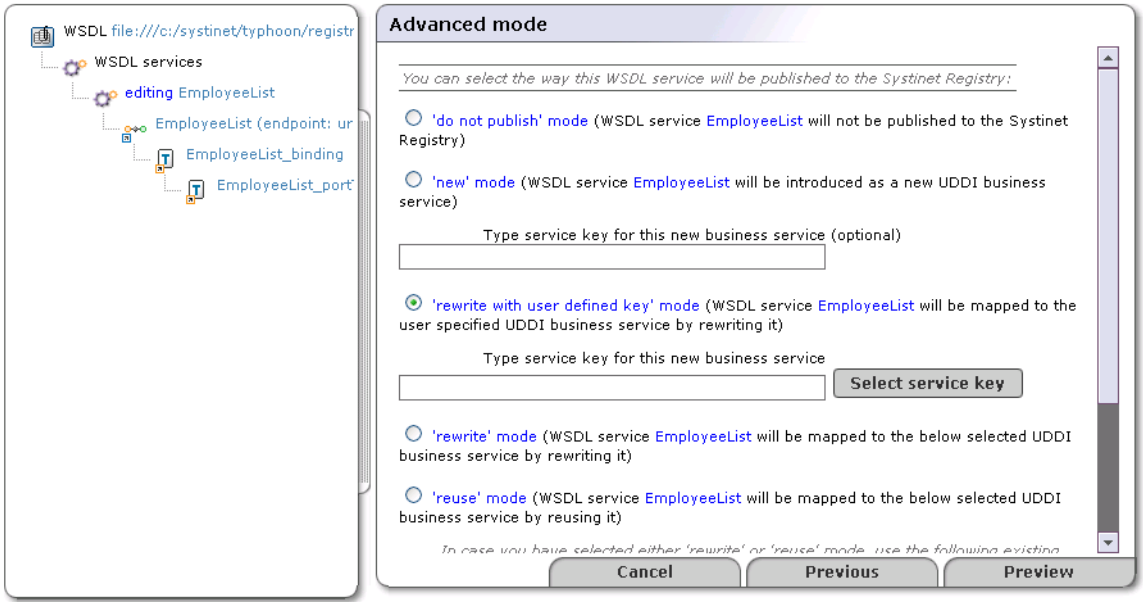
Figure 82. Publish WSDL Summary



Publishing WSDL Documents (Advanced Mode)

The advanced publishing mode allows you to specify certain details of how the WSDL document will be mapped to the UDDI registry. To publish in this mode, follow the steps from the previous section, and toggle the **Advanced mode** check box on. Once you click on the button **Publish** the Advanced Mode Publish page shown in Figure 83 will appear.

Figure 83. Publish WSDL (Advanced Mode)



In the left tree panel, you can see how artifacts of the WSDL document will be published. Click on a tree branch to edit how WSDL artifacts will be mapped to BEA AquaLogic Service Registry. Explanatory instructions in the right panel describe the mapping options. Click **Preview** to see how each part of the WSDL document will be mapped to the registry. From the Preview page, you can go back to adjust the WSDL mapping.

The wizard's default selection in Figure 83 is based on the following rules:

- If a possible mapping of a WSDL artifact already exists in the registry, and the user owns this UDDI structure, the wizard will suggest rewriting that mapping in the registry.
- If a possible mapping of a WSDL artifact already exists in the registry, and the user does not own this UDDI structure, the wizard will suggest reusing that UDDI entity.
- If no mapping of the WSDL artifact exists in the registry, the wizard will suggest creating a new UDDI entity to represent the mapping.

BEA AquaLogic Service Registry applies these rules automatically when you publish a WSDL document without the **Advanced mode** option.

Note

Publishing of WSDL operations and WSDL messages is not implemented in this BEA AquaLogic Service Registry release.

Unpublish WSDL

To unpublish a WSDL definition:

1. Search for the WSDL document in the registry.
2. In the result view, click on a business service.
3. The page with business service details will appear, click the **Unpublish** button at the page.
4. The **Unpublish WSDL document** wizard will appear.

Publish XML

BEA AquaLogic Service Registry XML to UDDI (XML2UDDI) mapping enables the automatic publishing of XML documents to UDDI, enabling precise and flexible UDDI queries based on specific XML artifacts and metadata

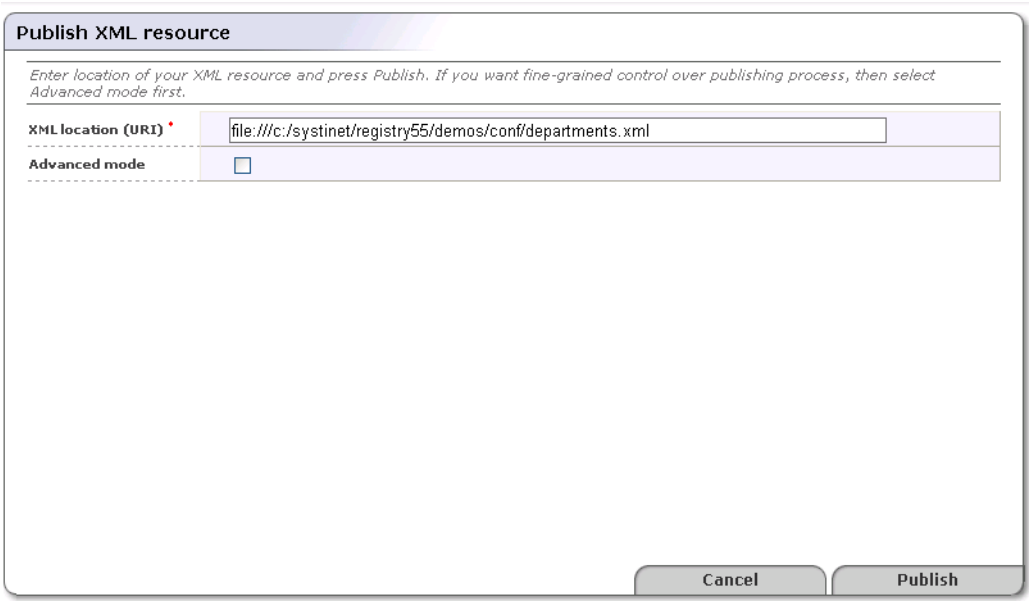
If you want to unpublish an XML document, use the **Find XML** button, then click the **Unpublish** button in the search result page.

Publishing an XML Document

To publish an XML document:

1. Click on the **XML** link under the **Publish** main menu tab.
2. The page shown in Figure 84 will appear.

Figure 84. Publish XML Document



3. Enter an **XML location**. To demonstrate, choose the file `REGISTRY_HOME/demos/conf/employees.xml` from the BEA AquaLogic Service Registry demos.

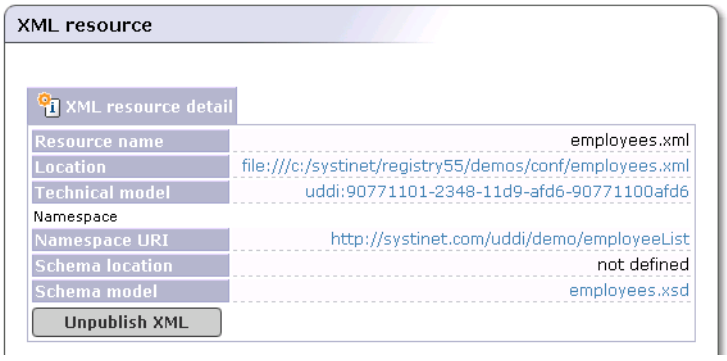
4. Leave the **Advanced mode** check box unchecked, and click **Publish**.

The XML document will be published to BEA AquaLogic Service Registry You can review how the XML document has been mapped to BEA AquaLogic Service Registry at Figure 85.

Note

The content of the XML document is not copied into the registry

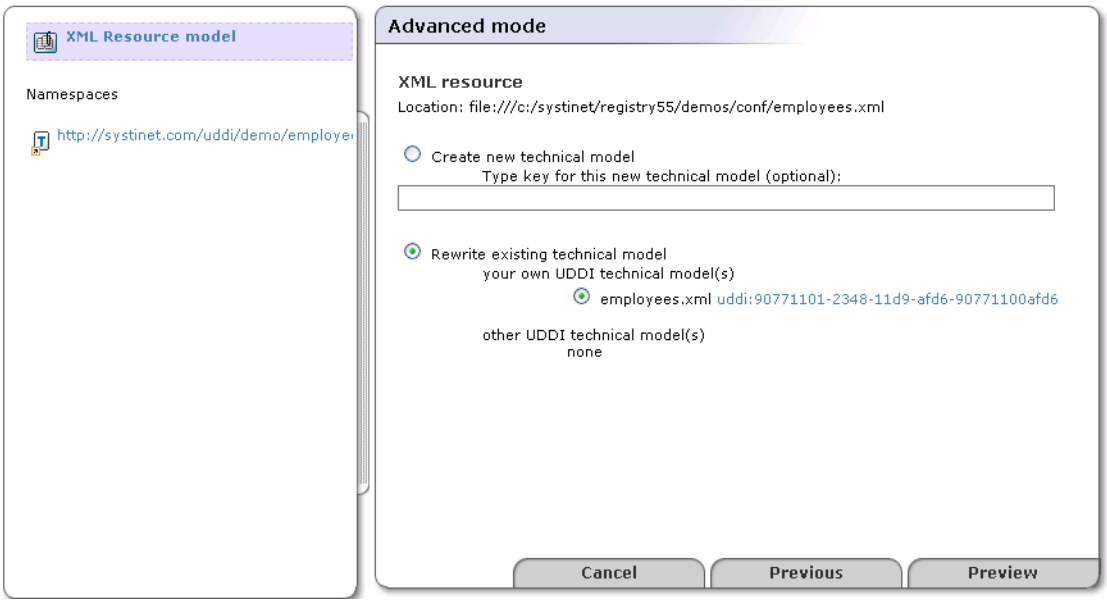
Figure 85. Publish XML Document Summary



Publishing an XML Document - Advanced Mode

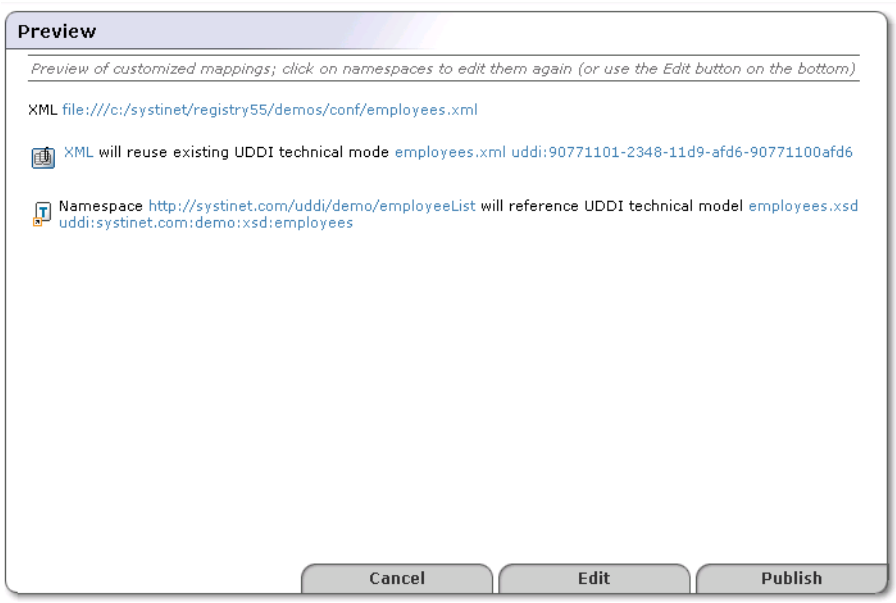
The advanced publishing mode allows you to specify certain details of how the XML document will be mapped to the UDDI registry. To publish in this mode, follow the steps from the previous section, check the box labeled **Advanced mode**, and click **Publish**. This returns the Advanced Mode Publish page shown in Figure 86 will appear.

Figure 86. Publish XML Document - Advanced



In the left tree panel, you can see how Namespaces of the XML document will be published. Click on a Namespace to edit how the Namespace will be mapped to BEA AquaLogic Service Registry. Explanatory instructions in the right panel describe the mapping options. Click **Preview** to see how the XML document and its Namespaces will be mapped to BEA AquaLogic Service Registry. From the Preview page, you can go back to edit the XML mapping.

Figure 87. Publish XML Document - Preview



Unpublish an XML Document

The Unpublish XML operation allows you to delete an XML mapping from BEA AquaLogic Service Registry To unpublish an XML document, you must search for the XML document first.

Publish XSD

BEA AquaLogic Service Registry XSD to UDDI (XSD2UDDI) mapping enables the automatic publishing of XML schema documents to UDDI, enabling precise and flexible UDDI queries based on specific XML schema artifacts and metadata.

If you want to unpublish an XML schema document, use the **Find XSD** button and click the **Unpublish** button in the search result page.

Publishing an XML Schema

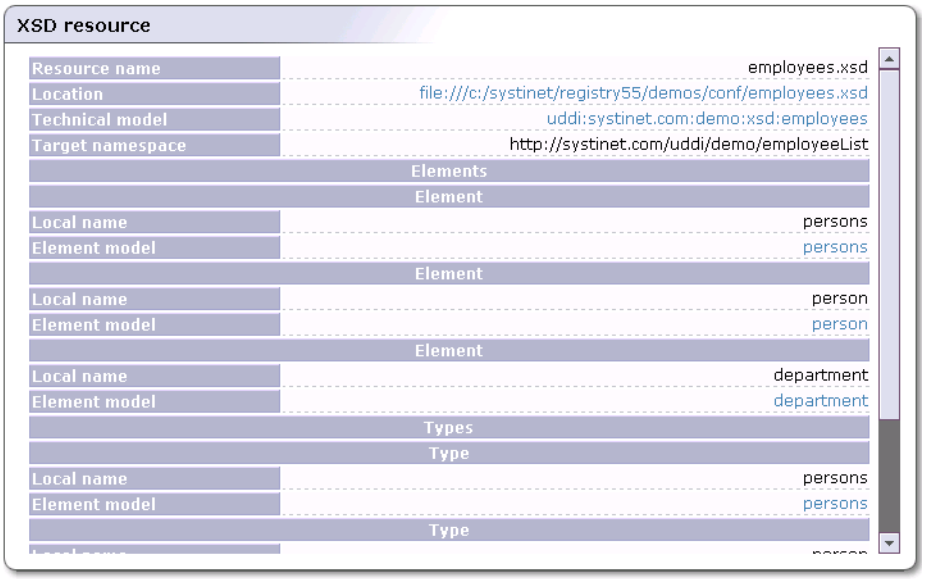
To publish an XML Schema document:

1. Click on the **XSD** link under the **Publish** main menu tab.
2. The page shown in Figure 88 will appear.

Figure 88. Publish XSD

3. Enter an **XML Schema location**. To demonstrate, use the file `REGISTRY_HOME/demos/conf/employees.xsd` from the BEA AquaLogic Service Registry demos.
4. Leave the **Advanced mode** check box unchecked, then click **Publish**.
5. The XML Schema document will be published to the registry. You can review mappings of the XML Schema document itself and its elements at Figure 89.

Figure 89. Publish XSD Summary

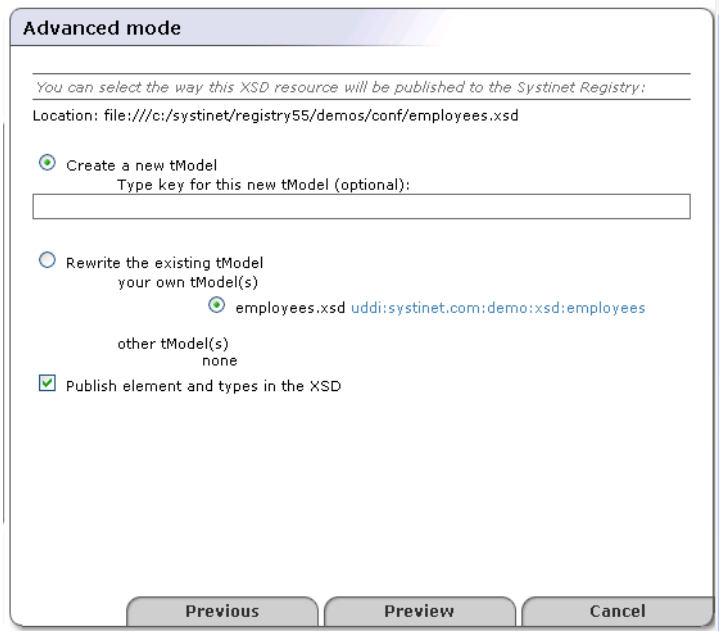


Publishing an XML Schema (Advanced Mode)

The advanced publishing mode allows you to specify certain details of how the XML Schema document will be mapped to the UDDI registry. To publish in this mode:

1. Follow the steps from the previous section, but check the **Advanced mode** box
2. Click **Publish**. This returns the Advanced Mode Publish page shown in Figure 90.

Figure 90. Publish XSD - Advanced



3. In the left tree panel, you can see how the XML Schema and its possible XML Schema imports will be published. Click on an XML Schema model node to edit how the parts of the XML Schema will be mapped to the BEA AquaLogic Service Registry. The explanatory instructions in the right panel describe the mapping options.

4. Click the **Preview** to see how the XML Schema document will be mapped to BEA AquaLogic Service Registry. From the Preview page, you can go back to edit the XML Schema mapping.

Unpublish an XML Schema

The Unpublish XML operation allows you to delete the XML Schema mapping from BEA AquaLogic Service Registry. To unpublish an XML Schema document, you must search for the XML Schema document first.

Publish XSLT

BEA AquaLogic Service Registry XSLT to UDDI (XSLT2UDDI) mapping enables the automatic publishing of XSL Transformations to UDDI, enabling precise and flexible UDDI queries based on specific XSLT artifacts and metadata.

If you want to unpublish an XSL transformation, click the **Find XSLT** button, then click the **Unpublish** button in the search result page.

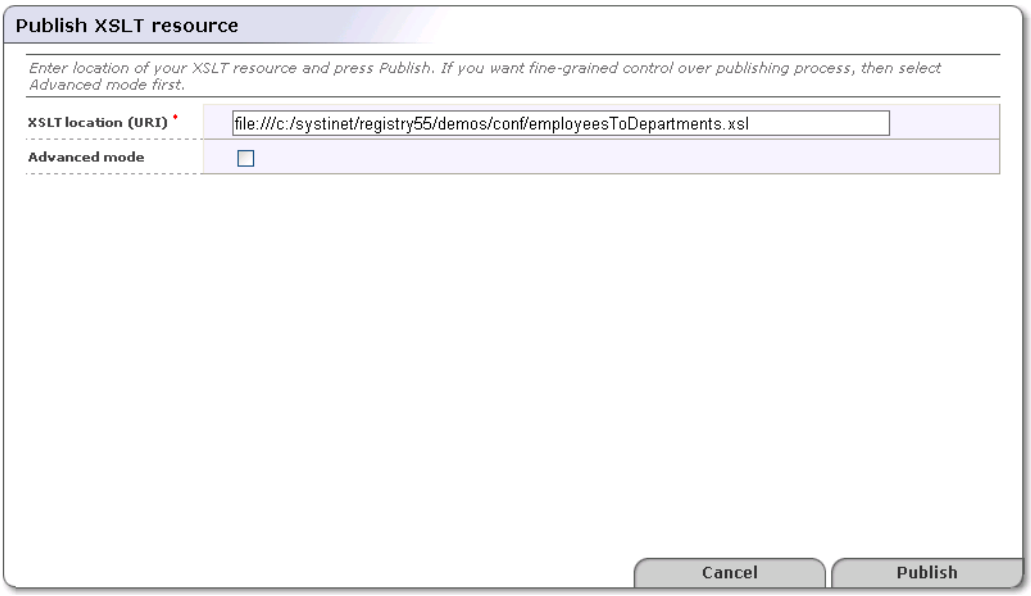
Publishing an XSL Transformation

To publish an XSL transformation:

1. Click on the **XSLT** link under the **Publish** main menu tab.

2. The page shown in Figure 91 will appear.

Figure 91. Publish XSLT

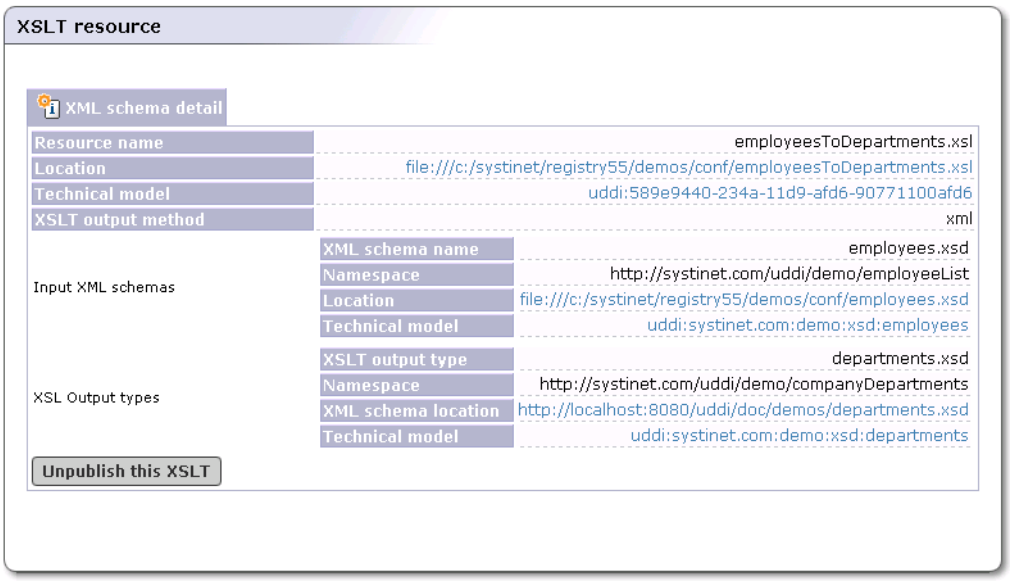


3. Enter an **XSLT location**. To demonstrate, use the `REGISTRY_HOME/demos/conf/employeesToDepartments.xsl` file from the BEA AquaLogic Service Registry demos.

4. Leave the **Advanced mode** check box unchecked, then click **Publish**.

The XSL transformation will be published to BEA AquaLogic Service Registry. You can review how XSLT artifacts have been mapped to BEA AquaLogic Service Registry at Figure 92

Figure 92. Publish XSLT Summary

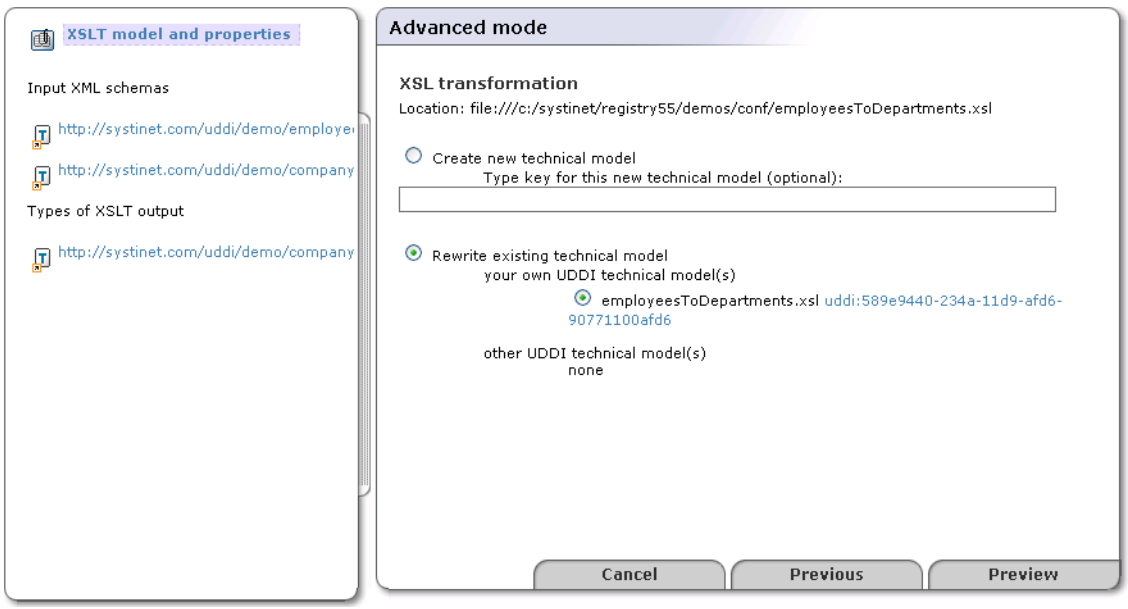


Publishing an XSL Transformation (Advanced Mode)

The advanced publishing mode allows you to specify certain details of how the XSL transformation will be mapped to the UDDI registry. To publish in this mode:

1. Follow the steps from the previous section, but check the **Advanced mode** box.
2. Click **Publish**. This returns the Advanced Mode Publish page shown in Figure 86.

Figure 93. Publish XSLT- Advanced



In the left tree panel, you can see how XSLT and its input and output schemas will be published.

3. Click on an XSLT node itself, its input XML Schemas, and types of XSLT output to edit how these artifacts will be mapped to BEA AquaLogic Service Registry. Explanatory instructions in the right panel describe the mapping options.

4. Click **Preview** to see how the XSLT will be mapped to BEA AquaLogic Service Registry. From the Preview page, you can go back to edit the mapping.

5.6. Signer Tool

One of the most important advantages of UDDI version 3 is its support for digital signatures. Without signatures you cannot verify whether the publisher of a business entity is really who that publisher claims to be. But if the publisher has signed the UDDI structure, anyone can verify that the information is unmodified by any means (including by UDDI registry operators) and to confirm the publisher's identity.

The BEA AquaLogic Service Registry Signer tool simplifies signature manipulation. You can find this tool's script in the `bin` directory of your BEA AquaLogic Service Registry installation. The Signer is a graphical application that can be used to add, remove, and verify the signatures of UDDI structures you have published.

Note

If you are using IBM Java, you must install Bouncy Castle security provider. See Installation Guide [http://edocs.bea.com/alsr/docs21/install/].

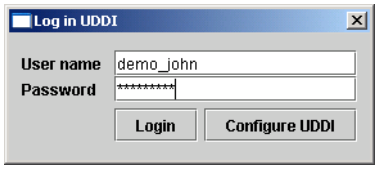
5.6.1. Starting the Signer

1. To start the Signer tool, first ensure that BEA AquaLogic Service Registry is running, then execute the following script from the `bin` subdirectory of your BEA AquaLogic Service Registry installation:

Windows:	signer.bat
UNIX:	./signer.sh

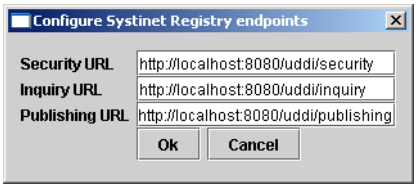
2. When the tool starts, you must first authenticate yourself against the selected UDDI version 3 registry. Simply provide your user name and password. If your registry is not running on a local machine, you must configure its endpoints. This can be accomplished via the **Configure UDDI** button.

Figure 94. Login Dialog



3. On the returned screen, set the endpoints of the Security, Inquiry, and Publishing Web services. For help, ask the administrator of your registry.

Figure 95. Configure Dialog

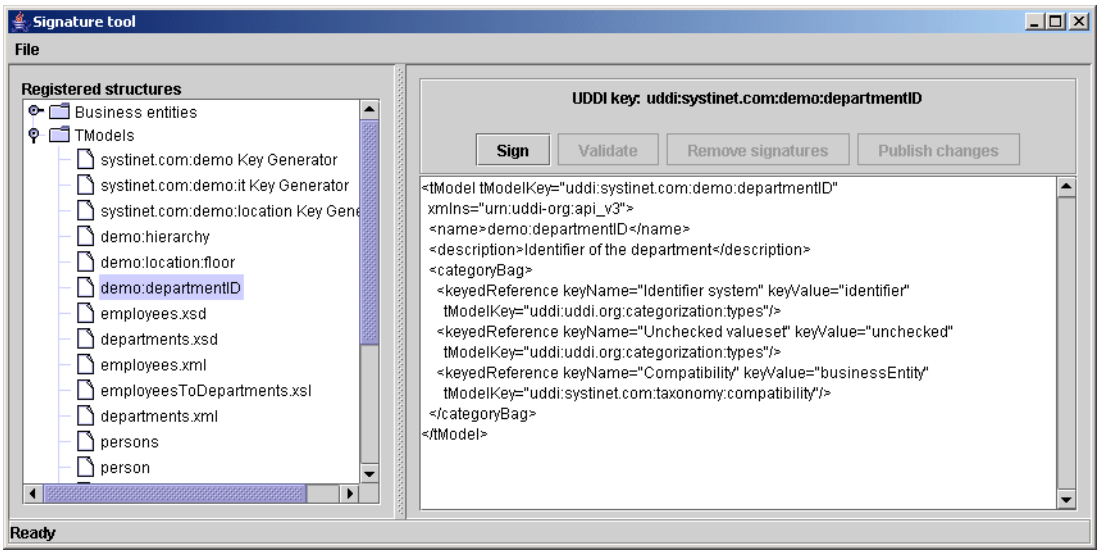


4. Once you have entered your user name and password, click the **Login** button. The Signer tool will attempt to authorize you at the selected registry. If authorization fails, you can correct your login information. Once it succeeds, the **Login dialog** disappears and the Signer tool asks BEA AquaLogic Service Registry for your registered information (businessEntities and tModels that you have published).

5.6.2. Main Screen

In the Signer tool's interface, the left part of the main screen consists of a tree containing all your businessEntities and tModels. If you wish to add or remove a digital signature, select the structure to sign from this tree. The Signer will fetch it from the registry. When the structure is fetched, its XML representation is displayed in the right panel. The **Sign** button is unblocked. If the structure has been already signed, the **Remove signatures** button is unblocked as well.

Figure 96. Signature Tool - Main Screen



The status bar at the bottom of the application informs the user of current action progress and results.

5.6.3. Sign

To sign a UDDI structure, you must set up the Java keystore. Use JDK tool **keytool** to generate the keystore. Please, see your JDK documentation for more information how to use **keytool**. The Signer tool has been tested with keystores in JKS and PKCS12 formats.

Note

To generate the certificate issue the following command

keytool -genkey -keyalg RSA -storetype JKS -alias demo_john -keystore test_certificate.jks

Example of the dialog:

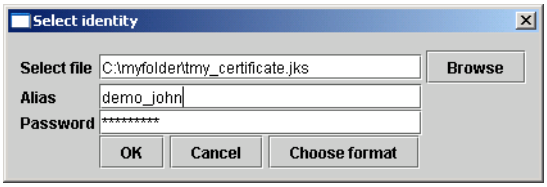
```
 Enter keystore password:  changeit
What is your first and last name?
  [Unknown]:  John Johnson
What is the name of your organizational unit?
  [Unknown]:  UDDI
What is the name of your organization?
  [Unknown]:  Myorg
What is the name of your City or Locality?
  [Unknown]:  San Diego
What is the name of your State or Province?
  [Unknown]:  California
What is the two-letter country code for this unit?
  [Unknown]:  CA
Is CN=John Johnson, OU=UDDI, O=Myorg, L=San Diego, ST=California, C=CA correct?
  [no]:  yes
Enter key password for <demo_john>
        (RETURN if same as keystore password):
```

To sign a UDDI structure, you must set the Java keystore file, alias, and password as follows:

1. Click on the **Sign** button. This returns the **Select identity** dialog.
2. In the box labeled **Select identity**, type the path to the file with your Java keystore.
3. In the box labeled **Alias**, type the alias located in the identity.
4. In the box labeled **Password**, type the password used to encrypt the private key.

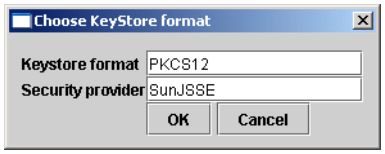
Important

If you enter the wrong value for the alias or the password, the tool will not be able to open the identity.



5. If the keystore is in the Sun JKS format, you do not have to click on **Choose format** button. You can leave default values there. If the keystore is not in the Sun JKS format, you can specify the format by clicking the **Choose format** button. In the returned dialog window, set the keystore format and its provider. For example, to use the PKCS12 format, set the format to PKCS12 and the provider to SunJSSE.

Figure 97. KeyStore Format Dialog



6. When the signing operation succeeds, the selected UDDI structure will have a digital signature and its XML representation will be updated. For security reasons, the signing process takes place on your computer so as not to risk compromise to your private key.

7. Finally the **Publish changes** and **Remove signatures** buttons are enabled.

5.6.4. Validation

The **Validate** button is used to perform validity check of UDDI structures that contain XML digital signatures. The result of this operation is displayed in the status bar.

5.6.5. Remove Signatures

The **Remove signatures** button is used to remove all digital signatures from the selected UDDI structure. When this operation is complete, the XML representation of the UDDI structure is updated. If the **Publish changes** button had been disabled, it is enabled.

5.6.6. Publish Changes

If you have signed the selected UDDI structure or removed digital signatures from it, you can select the **Publish changes** button to publish the changes to the registry. Its invocation uses standard UDDI publishing methods (`save_tModel`, etc.) to update this UDDI structure on the registry. The private key is not used during this operation.

5.6.7. Signer Configuration

The Signer tool automatically remembers the actual configuration such as registry endpoints or keystore location and format. The config file is saved in the user's home directory with the name `signer.conf`. You can change the location (and filename) by using the signer script's `-c` option. If you do not want this feature, use `-n`. The list of valid options can be obtained with `-h` option.

Administrator's Guide

The BEA AquaLogic Service Registry Administrator's Guide contains information necessary for the management of BEA AquaLogic Service Registry. It is aimed at the user responsible for configuring the registry and managing permissions, approval, and replication. This guide is divided into the following sections:

Section 1, Registry Management Registry management includes also management of user accounts and permissions, taxonomy management, and management of the approval process.

Section 2, Registry Configuration How to configure the Registry Console.

Section 3, Business Service Console Configuration How to configure the Business Service Console.

Section 4, Registry Console Configuration This section covers setting the URLs, directories, contexts, timeouts and limits associated with the BEA AquaLogic Service Registry interface.

Section 5, Permissions: Principles This section discusses the mechanism BEA AquaLogic Service Registry provides for the management of users' rights; permissions allow the administrator to manage or make available different parts of the registry to different users.

Section 6, Approval Process Principles This section describes Approval, a process by which control is exercised over the data published to BEA AquaLogic Service Registry.

Section 7, PStore Tool Describes a tool for management of protected stores for certificates and security identities.

Note

Make sure BEA AquaLogic Service Registry is running before attempting to use its consoles for configuration. To start it change to the `bin` subdirectory of `REGISTRY_HOME` and run:

Windows:	serverstart.bat
UNIX:	./serverstart.sh

The Registry Console can be found at `http://<hostname>:<port>/uddi/web` and the Business Service Console can be found at `http://<hostname>:<port>/uddi/bsc/web`.

Hostname and port are defined when BEA AquaLogic Service Registry is installed. The default port is 8080.

Log on as administrator. Initially, the administrator's user name is set to *admin* and the password to *changeit*.

Note

We strongly advise you to change the password for user admin once you have logged in.

Important

Be very careful when editing the `Operational business entity`, or editing the taxonomy `uddi-org:types`. Modification of these structures can lead to registry instability.

1. Registry Management

1.1. Accessing Registry Management

Registry Management is a set of tasks that the administrator can address through the Registry Console. These tasks are listed in Figure 1

To access the Registry Management console:

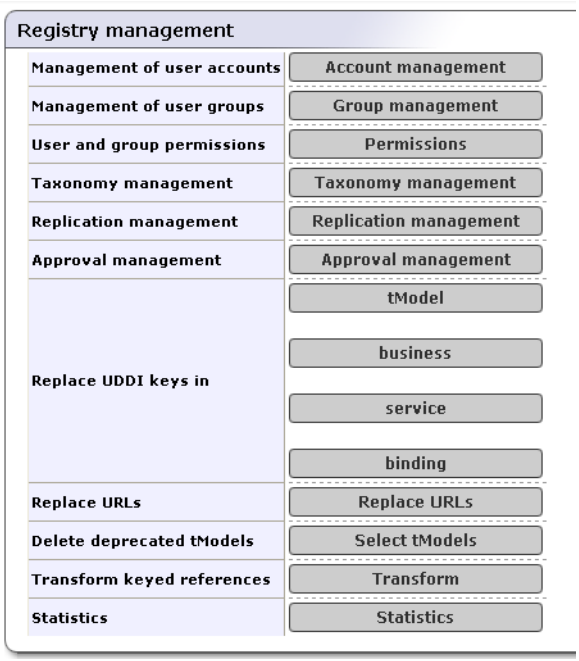
1. Log on as administrator or as a user with privilege to display **Manage** tab as described in Rules to Display the Manage Tab.
2. Click the **Manage** main menu tab.
3. Select the **Registry management** link under **Manage** tab. This returns the screen shown in Figure 1.

Rules to Display the Manage Tab

The **Manage** tab is available if at least one of the following conditions is satisfied:

- You have ApiManagerPermission to all methods (*) of one or more APIs (Account,Group,Permission,Taxonomy,ApprovalManagement,Statistics,Administration Utils).
- You have ConfiguratorManagerPermission to all operations (*) and all configurations (*).
- You have ApiManagerPermission to all methods (*) of ReplicationApi and ConfiguratorManagerPermission to all operations (*) for replication configuration.
- You have ConfiguratorManagerPermission to all operations (*) for web configuration.

Figure 1. Registry Management



- Account Management - Create, edit, and delete user accounts.
- Group Management - Create, edit, and delete accounts groups.
- Permissions - Set up permissions using the Registry Console
- Taxonomy Management - Build and maintain taxonomies via the Registry Console.
- Replication Management - Set up a subscription-based replication mechanism under which a slave registry receives notification from a master registry regarding updates and changes. (For more information on replication, please see Section 1.6, Replication Management.)
- Approval Management - set up requestors and approvers. This button is available only if the approval process is installed. See Installation Guide [http://edocs.bea.com/alsr/docs21/install/].
- Replace UDDI keys - Replace the UDDI keys of businessEntities, businessServices, tModels, and bindingTemplates.
- **Replace URLs** - Replace URL prefixes in the following entities:
  - tModel - OverviewDoc URL
  - tModelInstanceInfo - overviewDoc URL and DiscoveryURL
  - binding template - accessPoint URL
- **Delete deprecated tModels** - This option lets the administrator permanently delete deprecated tModels. A tModel is considered deprecated when it is marked as deleted by its owner. By default, tModels are deleted permanently by users. See Section 2.7, Node how to change this behavior.
- **Transform keyed references** - This operation is necessary when the type of taxonomy keyValues or the implementation of the taxonomy transformation service have been changed. For more information see, User's Guide, Section 5.4, Taxonomy: Principles, Creation and Validation.
- Statistics - This option displays two statistics tabs:
  - The first tab displays information about the number of accesses made to the various UDDI interface methods. One column displays the total request counts and a count of calls that fail and therefore return exceptions.
  - The second one contains counts of the main data structures (businessEntities, businessServices, tModels, bindingTemplates) in the database.

1.2. Account Management

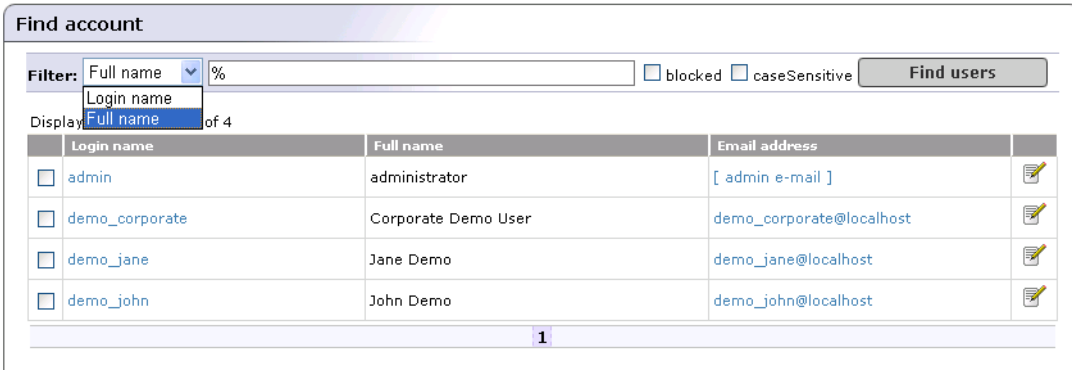
The BEA AquaLogic Service Registry administrator manages user accounts using the Registry Console. Use this console whenever you want to disable an account, change limits for a particular account, or take care of general housekeeping.

To access the Account management console:

1. Log on as administrator.
2. Click the **Registry management** link under the **Manage** tab.
3. Click the **Account management** button.

   This displays a list of all accounts, as shown in Figure 2.You can search accounts using the **Find users** button.

Figure 2. Find Account

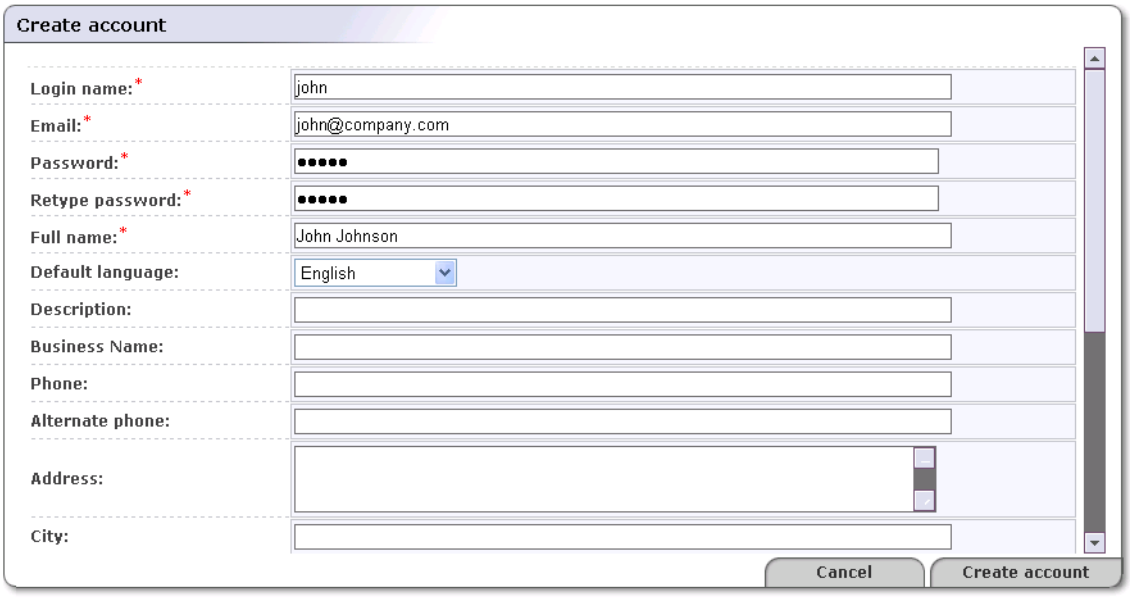


1.2.1. Create Account

To create an account:

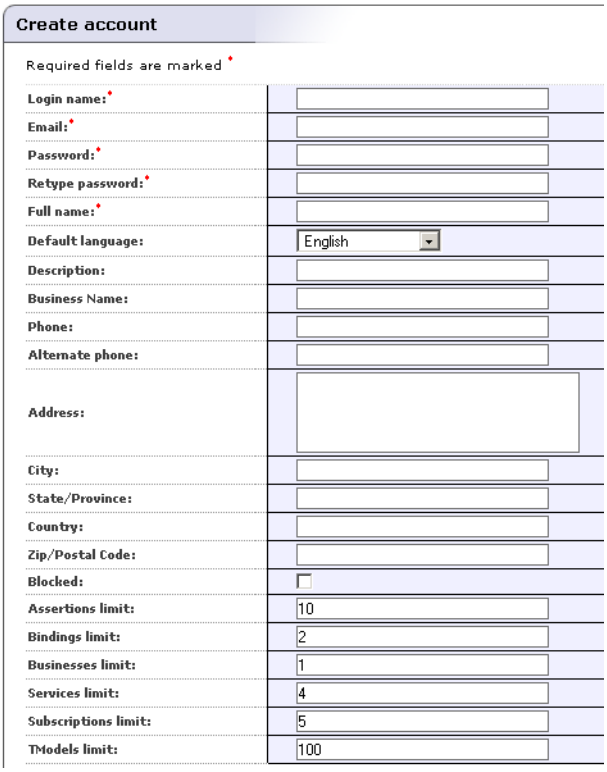
1. On the **Find Account** page, click **Create Account** button. This returns the Create account page shown in Figure 3.

Figure 3. Create Account



2. Provide the information shown in . Fields marked with a red asterisk (*) are required.

Figure 4. New Account - All Fields



Field descriptions (self-explanatory fields are omitted):

Default Language Code

Set the default language to be used during publishing when the language code associated with a particular field is not specified.

Use the following profile

Profile preference - Select your preferred predefined user profile from this drop down list

Blocked

Here you can enable/disable a user account. This is the account flag which prevents/permits a user from successfully logging onto the server.

Limits

These fields (**Assertions limit**, **Bindings limit**, **Businesses limit**, **Services limit**, **Subscriptions limit**, and**TModels limit**) indicate the number of these items allowed by the user. Changing default user limits is discussed in the Accounts section of Registry Configuration.

3. When finished, click **Create account**. This returns the Find account page. Note that the list of accounts now includes the account you have just created.

Account Limits

Each user account has the following limits for data saved under the account:

- Businesses limit - maximum number of businessEntities the account can hold. (1 by default).
- Services limit - maximum number of businessServices in the same businessEntity (4 by default).

- bindings limit - maximum number of bindingTemplates in the same businessService (2 by default).
- tModels limit - maximum number of tModels the account can hold. (100 by default).
- Assertions limit - maximum number of publisherAssertions the account can hold (10 by default).
- Subscriptions limit - maximum number of subscriptions an account can hold. (5 by default)

Common users can not change these limits. Only the administrator can change limits for a user or change default limits for newly created users.

The number of businessServices/bindingTemplates are checked against the limit on the user account owning the parent structure, not against the limit of the user processing the save_XXX call. For example, a user U1 owns a businessEntity BE_U1 and provides `create` ACL right to the user U2. The user U2 saves a new businessService under the BE_U1, total count of businessServices under the BE_U1 (no matter who is the owner) is checked against the service limit of the BE account.

Limit checking is skipped if a user who performs the operation has an `ApiManagerPermission` with the appropriate permission name and action:

- API (permission name)
  - `org.systinet.uddi.client.v3.UDDI_Publication_PortType` for skipping limit tests on Publishing V3 API.
  - `org.systinet.uddi.client.v2.Publish` for skipping limit tests on Publishing V2 API.
  - `org.systinet.uddi.client.v1.PublishSoap` for skipping limit tests on Publishing V1 API.
  - `org.systinet.uddi.client.subscription.v3.UDDI_Subscription_PortType` for skipping limit tests on Subscription API.

- operation (action)
  - `save_business` for skipping businesses limit test on Publishing V1/V2/V3 API
  - `save_service` for skipping services limit test on Publishing V1/V2/V3 API
  - `save_binding` for skipping bindings limit test on Publishing V1/V2/V3 API
  - `save_tModel` for skipping tModels limit test on Publishing V1/V2/V3 API
  - `add_publisherAssertions` for skipping assertions limit test on Publishing V2/V3 API
  - `set_publisherAssertions` for skipping assertions limit test on Publishing V2/V3 API
  - `save_subscription` for skipping subscriptions limit test on Subscription API

For more information see Section 5, Permissions: Principles. By default, only the administrator has these permissions, and therefore the administrator has an *unlimited* account.

1.2.2. Edit Account

To edit an account:

1. On the **Find account** page shown in Figure 2, click the **Edit Account** icon ( ) associated with the account you want to edit.

   This returns the Edit account page.

2. On the Edit account page, provide or change the information in the various fields. These are the same as the fields shown in Figure 4.

   Field descriptions (self-explanatory fields are omitted):

   Default Language Code

   Set the default language to be used during publishing when the language code associated with a particular field is not specified.

   Blocked

   Here you can enable/disable a user account. This is the account flag which prevents/permits a user from successfully logging onto the server.

   Limits

   These fields (**Assertions limit**, **Bindings limit**, **Businesses limit**, **Services limit**, **Subscriptions limit**, and**TModels limit**) indicate the number of these items allowed by the user. These are described in detail in the Accounts section of Registry Configuration.

3. When finished, click the button labeled **Save Changes**. This returns the Find account page.

1.2.3. Delete Account

To delete an account:

1. On the Find account page, check the box next to the **Login name** of the account you want to delete.
2. Click the **Delete Selected** button.
3. If you are certain you want to delete the account, click **Yes** when prompted. Note that on publication registries and standard installations of BEA AquaLogic Service Registry, all published information associated with the user will be lost.

Note

If you are using LDAP for storing users, the user account will not be deleted from the LDAP store, because LDAP stores are treated as read-only. The `delete account` operation will delete an account only from the registry database.

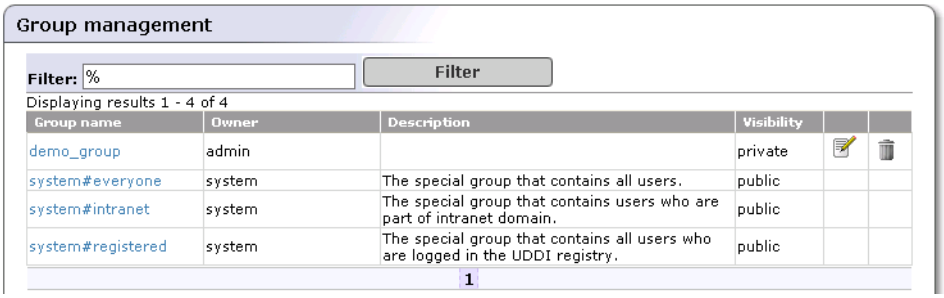
1.3. Group Management

User groups simplify management of access rights to each UDDI data structure. You can use groups to group users with similar rights.

The administrator can:

- Create and manage user groups
- Manage group membership

Figure 5. View User Groups



1.3.1. Create and Manage Groups

To create a new group:

1. Click on the **Manage** menu tab. On the Manage tab, select the **Registry management** link, and then click the **Group management** button. This returns the Group Management page.

2. To display all groups on the registry, click **Filter**. This returns a Group list like the one shown in Figure 5.

3. Click the **Add Group** button. This returns a blank Add group page much like the one shown in Figure 6.

Figure 6. Add Group Page

4. In the edit box labeled **Group name**, type the name of your group. In the edit box labeled **Group owner**, type the owner of the group. The default owner is Admin. These two fields are required.

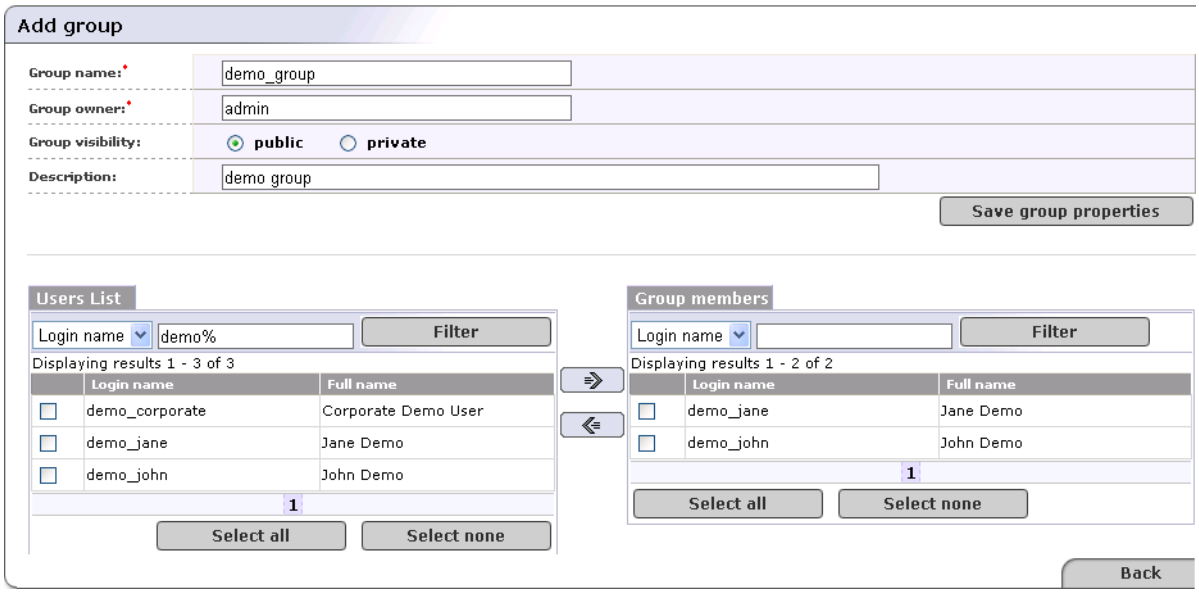
5. Use the radio buttons labeled **public** and **private** to set group visibility.

   Both public and private groups are visible to all users in the registry, meaning that all users are able to see which groups exist. Public and private groups differ in that members of public groups are visible to all users of the registry whereas members of private groups are visible only to the owner of the group.

6. Optionally, Enter a description of the group in the box labeled **Description**.

7. Click the **Save group properties** button. This returns the **Users list** and **Group members** sections shown in Figure 5.

Figure 7. Edit Group Membership



8. In the **Users list** section, click **Filter** to display a list of all of the registry's users.

   Use the drop down list in this section to sort users by **Login name** or **Full name**.

   Use the text box to further filter users. You can use % as wildcard in this field.

9. Check the boxes next to all members you wish to include, and click the right-pointing arrow ( ) to move them to the **Group members** table.

   Group members are updated in the database once you click the arrow buttons.

1.3.2. Manage Group Membership

To add or remove members from a group:

1. Click on the **Manage** main menu tab.
2. Click on the **Registry management** link. This returns the main Registry Management page.

   Click the **Group management** button. This returns the Group list shown in Figure 5.
3. Enter your search criteria, then click the **Filter** button. Click **Filter** without search criteria to return a list of all groups.
4. Click the **Edit** button ( ) in the row with the group you want to manage. This returns the Edit Group page. Specify search criterion for user accounts, then click the **Filter** button.
5. Use the arrow buttons (and) to add and remove users as shown in Figure 7

1.4. Permissions

This chapter describes how you can set permissions using the Registry Console. Before you start to work with permissions, we recommend reading Section 5, Permissions: Principles to become familiar with permissions principles.

BEA AquaLogic Service Registry uses the same interface for managing both user permissions and group permissions. In this section we discuss user permissions, but group permissions are handled the same way.

1.4.1. Accessing Permission Management

To access permission management:

1. Log on as Administrator or as a user who has permission to set permissions, as described in Section 5.1, Permissions Definitions.

2. Click the **Manage** main menu tab. On the **Manage** tab, select the **Registry management** link, and then click the **Permissions** button.

3. On the initial Select Principal screen, click **Filter**, without changing the default settings, to view a list of all users (principals).

   Tip

   Use the drop down list in this section, labeled **Filter:** to sort users by **Login name** or **Full name**.

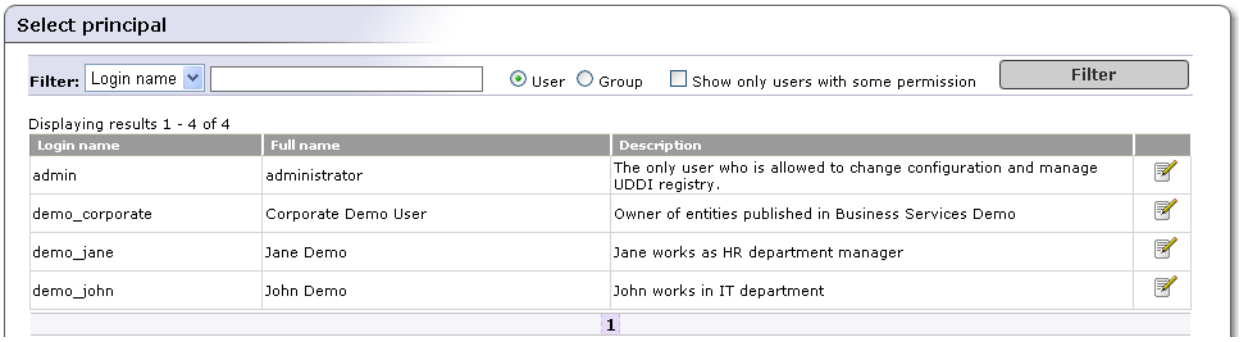
   Use the text box to further filter users by name. You can use % as wildcard in this field.

   Select the radio button labeled **User** to manage permissions for individual users. Select the button labeled **Group** to manage group permissions.

   Check the box labeled **Show only users/groups with some permission** to filter out principals who have not already been granted permissions.

   This returns the list of users shown in Figure 8.

   Figure 8. Select Principal



4. Click the **Edit** icon ( ) associated with the user or group whose permissions you wish to set.

1.4.2. Add Permission

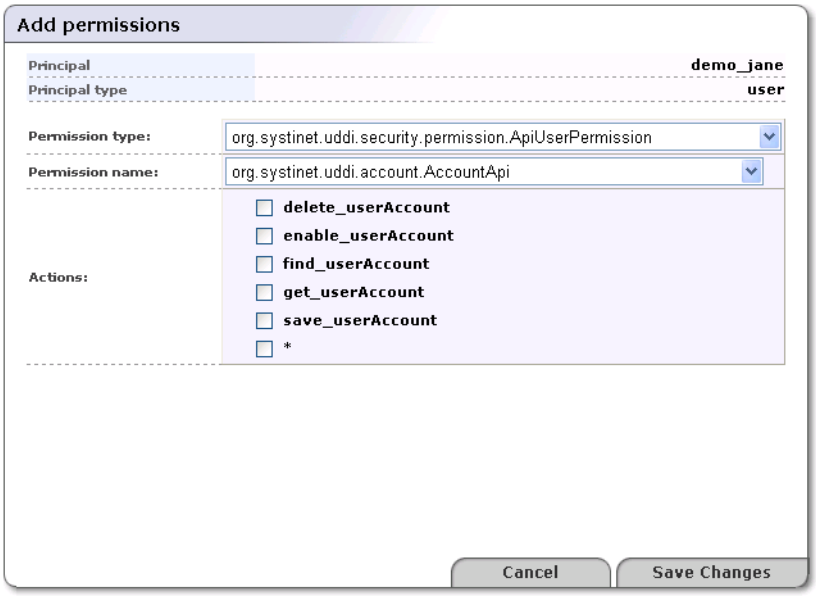
To add permissions:

1. Access permission management as described above in Section 1.4.1, Accessing Permission Management.

2. On the principal list page shown in Figure 8, click the **Edit** icon ( ) associated with the group or user to whom you wish to add permissions. On the returned Permissions page, click **Add permission**.

3. An Add permissions page much like the one shown in Figure 9 will appear.

Figure 9. Add Permission



4.
   - Select the type of permission from the drop down list labeled **Permission type**.
   - From the drop down list labeled **Permission name**, select the name of the permission to add.
   - Check the box(es) next to the actions associated with the permission name in order to grant permission to perform those actions. Check the box next to the asterisk (*) to permit all the actions on the list.

5. Click **Save Changes** to save the permission.

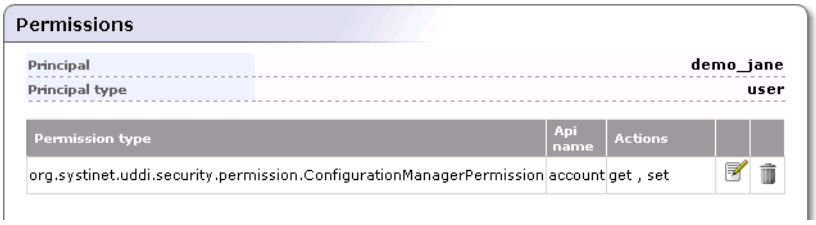
1.4.3. Editing and Deleting Permissions

To edit a permission:

1. On the principal list page shown in Figure 8, click the **Edit** icon ( ) associated with the user whose permissions you want to edit or delete.

2. If the principal has permissions defined, a permission list like the one shown in Figure 10 will appear.

Figure 10. Permissions List



3. Click the **Edit** or **Delete** icon ( ) associated with the permission you want to address.

1.4.4. Assigning Administrator's Permission

If you want to give administrator's permissions to an existing user, you must assign the following permissions types to the user:

- `org.systinet.uddi.security.permission.ApiManagerPermission`
- `org.systinet.uddi.security.permission.ApiUserPermission`
- `org.systinet.uddi.security.permission.ConfigurationManagerPermission`

For each **Permission type** set all **Permission name**s and all **actions** using the asterisk (*)

1.5. Taxonomy Management

This chapter describes how administrators can build and maintain taxonomies using the Registry Console. Before you start to manage taxonomies, we recommend reading User's Guide, Section 5.4, Taxonomy: Principles, Creation and Validation to become familiar with taxonomy principles.

The following tasks are described in this chapter:

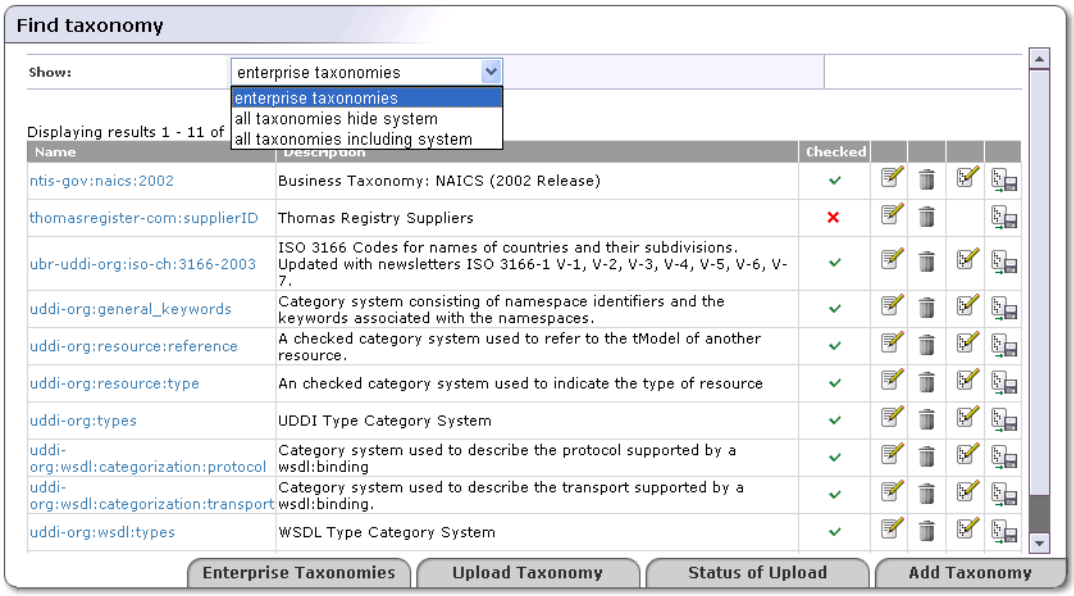
- Adding a taxonomy - How to add taxonomies BEA AquaLogic Service Registry.
- Finding taxonomies - How to locate taxonomies in BEA AquaLogic Service Registry.
- Editing taxonomies - How to change taxonomy categorization, compatibilities, and a taxonomy type that is important in range queries comparison.
- Editing a taxonomy structure - How to add categories, disable nodes, edit node values, and delete nodes.
- Uploading a taxonomy
- Downloading a taxonomy

To view the Taxonomy management page:

1. Log on as administrator.
2. Click the **Manage** tab under the Main menu, and then click on the **Registry management** link under the **Manage** menu tab.
3. Click **Taxonomy management**. This returns a blank Taxonomy management page. To view a selection of taxonomies, select a filter from the drop down list labeled **Show**. Possible filters are:
   - Favorite taxonomies
   - Enterprise taxonomies
   - All taxonomies hide system
   - All taxonomies including system

   This returns a list of taxonomies similar to that shown in Figure 11.

Figure 11. Find Taxonomy (Enterprise Taxonomies)



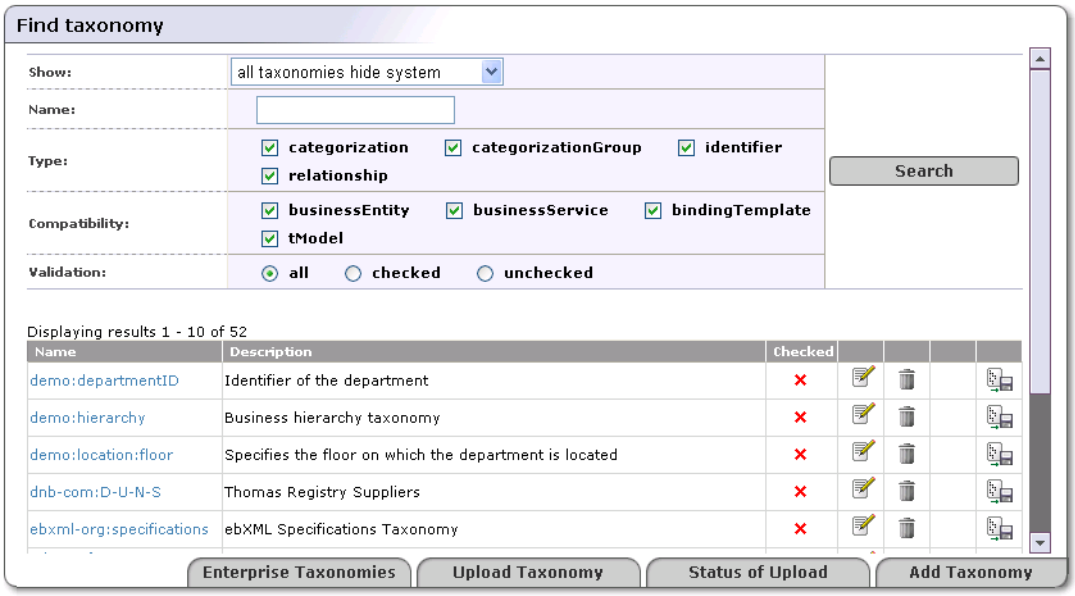
Use the page shown in Figure 11 to search enterprise taxonomies. You can classify taxonomies according to the following overlapping groups:

- Enterprise taxonomies - The BEA AquaLogic Service Registry administrator can define which taxonomies will be present in the enterprise taxonomies list. The **Enterprise taxonomies** button located in the bottom part of Figure 11 allows you to manage a list of enterprise taxonomies for all registry user accounts.
- Favorite taxonomies - All registry users can define their list of favorite taxonomies. See User's Guide, Section favorite Taxonomies for more information on how to manage your list of favorite taxonomies.
- System taxonomies - When you edit a taxonomy you can assign whether the taxonomy is a system taxonomy using the check box **System taxonomy**.

The reason for this taxonomy classification is to make taxonomy management and UDDI entity categorization easier.

If you want to manage taxonomies which are not in the enterprise taxonomy list, select **see all taxonomies including system taxonomies** from the drop down list labeled **Show**. The page shown in Figure 12 will appear. You can search taxonomies using the following criteria: taxonomy name, type, compatibility, and validation.

Figure 12. Find Taxonomy



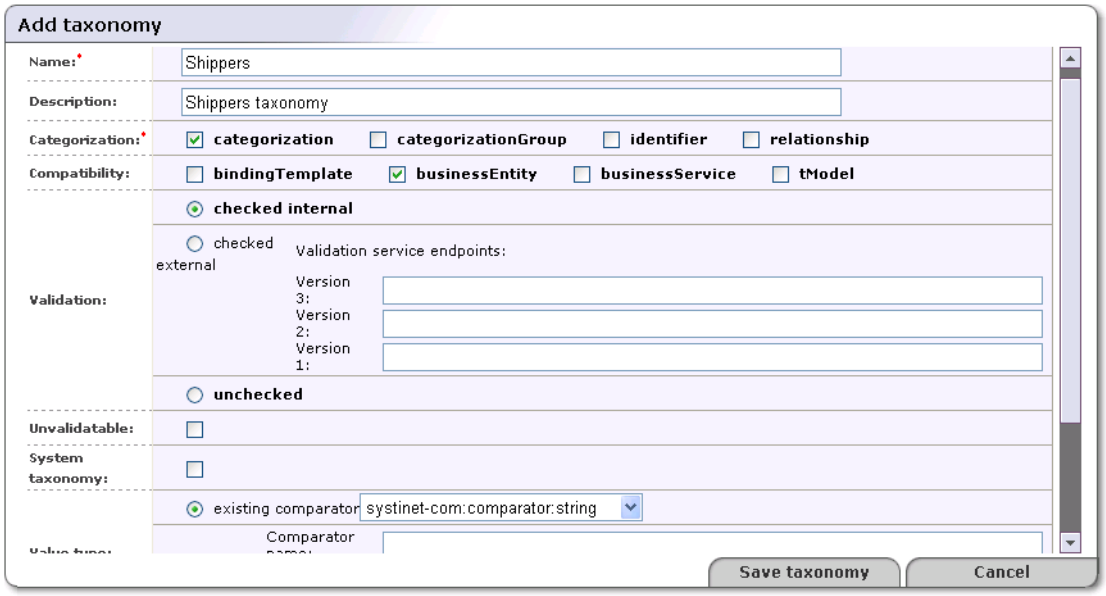
1.5.1. Adding Taxonomies

You can also add a root for a taxonomy by hand and build it through the Registry Console.

To add a taxonomy:

1. Click the **Add taxonomy** button shown in Figure 12.
2. Fill in as many of the fields on the Add taxonomy page, shown in Figure 13, as you require. Only two fields are required to create a taxonomy: **Name** and **Categorization**, however the more information you provide, the more useful your taxonomy will be.

Figure 13. Add Taxonomy



3. In the field labeled **Name**, name your taxonomy.

4. Skip the field labeled **tModel key**. This key is generated when you save the taxonomy.

5. In the field labeled **Description**, briefly describe the use of the taxonomy.

6. Check one or more of the boxes in the section labeled **Categorization**. Categorizations are discussed in Section 5.4.2, Taxonomy Types.

7. You may enforce that your taxonomy can be used only within the UDDI structures of your choice. Select one or more of the main UDDI structure types in the section labeled **Compatibility**.

8. **Validation** In this section, specify whether the values in keyedReferences within the taxonomy will be checked or not.

   - Select **checked internal** if you want BEA AquaLogic Service Registry to check keyedReferences in which the taxonomy is used against a validation service deployed within BEA AquaLogic Service Registry.

   - Select **checked external** if you want BEA AquaLogic Service Registry to check keyedReferences in which the taxonomy is used against a validation service deployed on a remote SOAP stack such as Systinet Server.

     If you are using an external validation service, provide at least one **Validation service endpoint**.

   - Select **unchecked** if you do not want BEA AquaLogic Service Registry to perform any checks on values used in keyedReferences in which the taxonomy is used.

9. Use the box labeled **Unvalidatable** to mark taxonomies as temporarily unavailable.

10. Check the box labeled **System taxonomy** if you want to mark the taxonomy for internal use. Users and administrators can filter System taxomies easily when searching in the Business Service Console.

11. Select a **Value type** for keyValues. You can choose from three existing comparators or create a custom comparator. Existing comparators are:

    - `string` - keyValues are treated as string values. If keyValues type is unknown then keyValues are treated as strings. The maximum length is 255 characters.
    - `numeric` - keyValues are treated as decimal numbers. The value can have maximum 19 digits before the decimal point and maximum 6 digits after the decimal point.
    - `date` - keyValues are treated as dates.

    If you want to categorize using a custom comparison, you must create a new comparator tModel and implement a transformation service. The **Transformation service endpoint** must start with the `class:` prefix. Please see the Section 5.4.4, Types of keyValues and Section Custom Ordinal Types for more information.

12. Use the box labeled **Add to favorites** to mark the taxonomy as either a personal favorite. This is an option available to all users.

    Check the box labeled **Add to enterprise** to mark the taxonomy specific to the particular enterprise or application. For more information, see Section Enterprise Taxonomies

13. Click **Save taxonomy**.

Note

Later, you will be able to modify all taxonomy attributes except the validation type. See Section 1.5.3, Editing Taxonomies for attribute descriptions.

1.5.2. Finding Taxonomies

To locate a taxonomy in BEA AquaLogic Service Registry:

1. Log on as administrator.
2. Click the **Manage** tab under the Main menu, and then click on the **Registry management** link under the **Manage** menu tab.
3. Click **Taxonomy management**. This returns a blank Taxonomy management page. Select a filter from the drop down list labeled **Show**. Possible filters are:

    - Favorite taxonomies
    - Enterprise taxonomies
    - All taxonomies hide system
    - All taxonomies including system

    This returns a list of taxonomies similar to that shown in Figure 11.

4. On the returned Find taxonomy page, you can further filter the results by

    a. name

    b. type - Types are discussed in Section 5.4.2, Taxonomy Types

    c. compatibility

d. validation

5. From the list of taxonomies the fit the filter criteria, select the taxonomy you wish to view by clicking on its name.

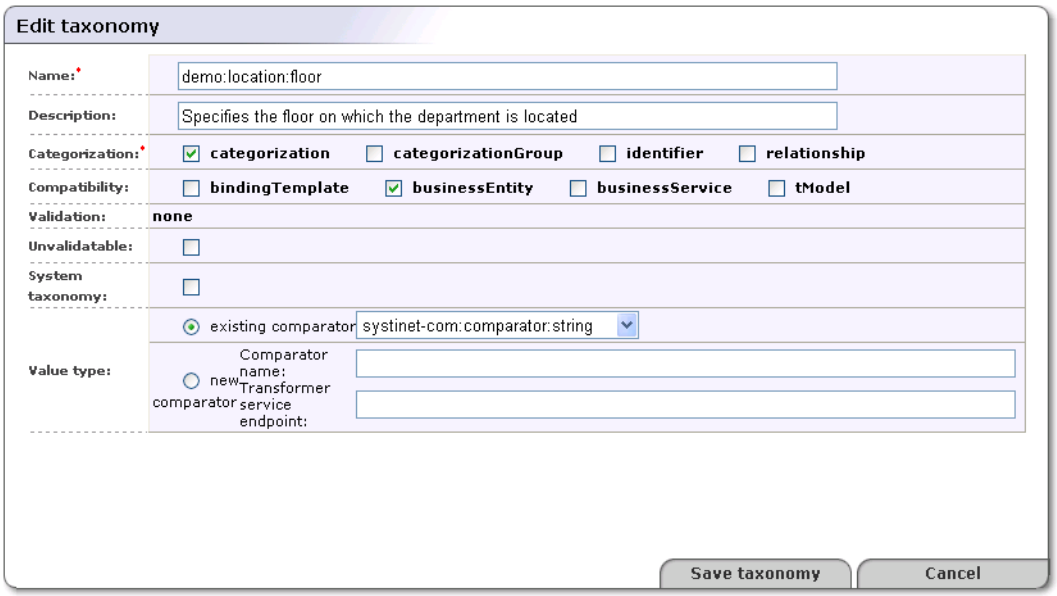
1.5.3. Editing Taxonomies

The Registry Console makes it possible to change any taxonomy attribute except the validation type attribute. To edit a taxonomy:

1. Identify the taxonomy you want to edit as described in Section 1.5.2, Finding Taxonomies.

2. Click on the **Edit Taxonomy** icon in the Find Taxonomy page shown in Figure 12. This loads the Edit taxonomy page shown in Figure 14.

Figure 14. Edit Taxonomy



3. In the field labeled **Name**, edit the taxonomy's name.

4. In the field labeled **Description**, briefly describe the use of the taxonomy.

5. Check one or more of the boxes in the section labeled **Categorization**. Categorizations are discussed in Section 5.4.2, Taxonomy Types.

6. You may enforce that your taxonomy can be used only within the UDDI structures of your choice. Select one or more of the main UDDI structure types in the section labeled **Compatibility**.

7. **Validation** In this section, specify whether the values in keyedReferences within the taxonomy will be checked or not.

   - Select **checked internal** if you want BEA AquaLogic Service Registry to check keyedReferences in which the taxonomy is used against a validation service deployed within BEA AquaLogic Service Registry.

   - Select **checked external** if you want BEA AquaLogic Service Registry to check keyedReferences in which the taxonomy is used against a validation service deployed on a remote SOAP stack such as Systinet Server.

If you are using an external validation service, provide at least one **Validation service endpoint**.

- Select **unchecked** if you do not want BEA AquaLogic Service Registry to perform any checks on values used in keyedReferences in which the taxonomy is used.

8. Check the box labeled **Unvalidatable** to mark the taxonomy as temporarily unavailable. When you save a checked taxonomy without a validation service, the taxonomy will be saved with **Unvalidatable** toggled on.

9. Select a **Value type** for keyValues. You can choose from three existing comparators or create a custom comparator. Existing comparators are:

   - `string` - keyValues are treated as string values. If keyValues type is unknown then keyValues are treated as strings. The maximum length is 255 characters.
   - `numeric` - keyValues are treated as decimal numbers. The value can have maximum 19 digits before the decimal point and maximum 6 digits after the decimal point.
   - `date` - keyValues are treated as dates.

   If you want to categorize using a custom comparison, you must create a new comparator tModel and implement a transformation service. The **Transformation service endpoint** must start with the `class:` prefix. Please see the Section 5.4.4, Types of keyValues and Section Custom Ordinal Types for more information.

1.5.4. Editing a Taxonomy Structure

While the fields in the Edit Taxonomy page are used for controlling global attributes, the management of nodes within the taxonomy itself is handled by categories. Here you can add nodes, edit node values, and enable or disable them.

Tip

Registry Console provides you with unique functionality to manage taxonomy structures using a drag-and-drop wizard.

Note

Changing taxonomy structure is allowed only for checked taxonomies which are validated by the internal validation service.

Adding Categories to a Taxonomy

Note

Before we begin assigning names to a taxonomy it is important to consider how the naming system will function.

Taxonomy values in UDDI consist of name and value pairs, like entries in a hash table. As with hash table values, the trade-off between economy of space and extensibility must be taken into consideration. Too long a `Value` string will be wasteful; too small and it will not be extendable.

To add a node to a branch or root:

1. Identify the taxonomy you want to edit as described in Section 1.5.2, Finding Taxonomies.

2. Click the **Edit taxonomy structure** icon ( ) in the **Find taxonomy** page as shown in Figure 12.

Important

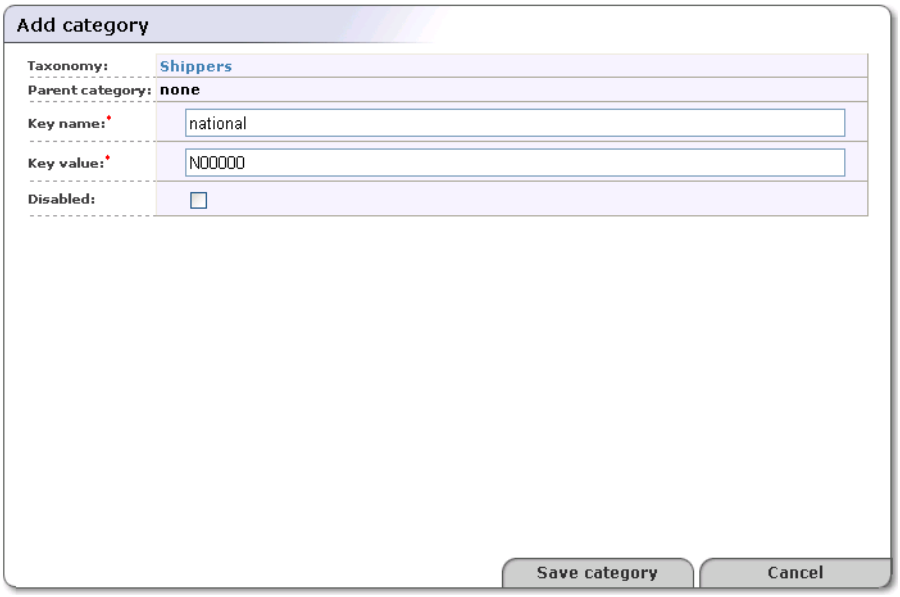
This icon is only available for checked taxonomies that are validated by the internal validation service. You cannot edit the structure of unchecked taxonomies and checked taxonimies that are validated by other services.

3. The **Edit taxonomy structure** page will appear.
4. On this page, right-click on the taxonomy's folder icon to display its context menu, and select the **Add category** action.
5. This displays the Add category page. Provide the required **Key name** and **Key value**, and click **Save category**.

   In the shipping taxonomy example shown in Figure 15, we use a value algorithm that employs an array of six alpha-numeric characters:

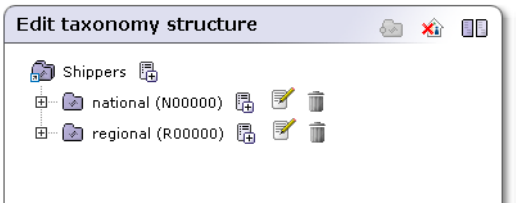
   - The first element in the array signifies the first geographic division.
   - The second and third elements signify further geographic subdivisions where necessary.
   - The fourth character indicates transport mode.
   - The fifth character is reserved for an extension to the system allowing a coded category containing a maximum of thirty-six divisions.
   - The sixth can be used for a weight coding system.

Figure 15. Add Category



6. Check the box labeled **Disabled** to mark the category as either helper or deprecated. Note that disabled categories cannot be used as valid options in keyedReferences.
7. Click the **Save category** button. This builds the taxonomy as shown in Figure 16.

Figure 16. Edit Shippers Taxonomy 1



Drag-and-Drop Editing

To demonstrate the drag-and-drop functionality, we will extend the Shippers taxonomy from previous section.

Add four non-disabled categories with the following attributes:

a. Key name: `national`; Key value: `N00000`.

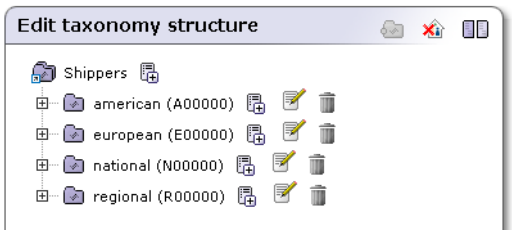
b. Key name: `regional`; Key value: `R00000`

c. Key name: `american`; Key value: `e00000`.

d. Key name: `european`; Key value: `e00000`.

The result is shown in Figure 17.

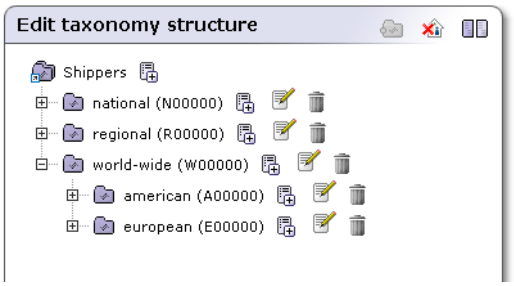
Figure 17. Edit Shippers Taxonomy 2



Add a new category {`world-wide`,`W00000`} to the same level as all previous taxonomies.

We want to put both the `european` and `american` categories under the `world-wide` category as shown in Figure 18.

Figure 18. Edit Shippers Taxonomy 3



To do so, click on the `european` category node and drag it to the `world-wide` category node. Do the same with the `american` category node. The structures will be displayed as shown in Figure 18.

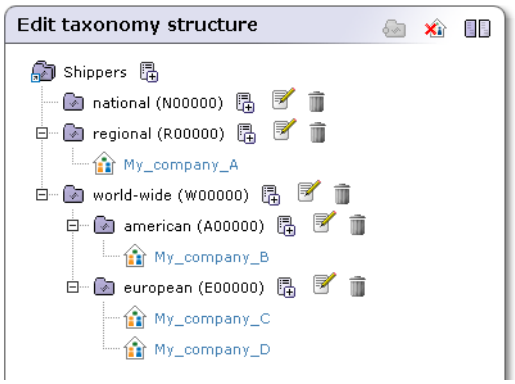
Note

When you drag a category with all child nodes, the child nodes are dragged along with the parent.

For taxonomies with many categories you can use two taxonomy trees and drag categories from the left taxonomy tree and drop them to the right taxonomy tree. To enable this two-tree display, click on the **Second taxonomy tree** icon ( ).

The Edit taxonomy structure also allows you to see UDDI entities categorized with a category from the taxonomy tree. An example of displayed business entities categorized with the Shippers taxonomy is shown in Figure 19. To switch to the view of categorized UDDI entities, click the house icon ( ).

Figure 19. Edit Shippers Taxonomy 3



Deleting and Disabling Nodes

There are two policy choices for dealing with categories of entities that cease to be active. Either:

- They can be marked as disabled.
- They can be deleted entirely from the taxonomy.

To delete a taxonomy node,

1. Navigate through the taxonomy tree via the Edit taxonomy page.
2. Right-click on the category node's icon and select the **Delete** option from its context menu.

Important

Because this process is irreversible you will be asked to confirm.

To disable a taxonomy node:

1. Navigate through the taxonomy tree via the Edit taxonomy page.
2. Right-click on the category node icon to display its context menu.
3. Select the **Edit category** option from the context menu. This returns the Edit category page.
4. On the Edit category page, check the option labeled **Disable**.
5. Click the **Save category** button.

1.5.5. Uploading Taxonomies

To upload a taxonomy:

1. Log on as administrator.
2. Click **Manage** main menu tab, then click on the link **Registry management** under the **Manage** menu tab.
3. Click the **Registry management** button. A list of taxonomies like the one shown in Figure 12 will appear.
4. Click the **Upload taxonomy** button.

Note

The format of data on this page is described in the Section Persistence Format of the Developer's Guide.

1.5.6. Downloading Taxonomies

There are two obvious cases in which you will want to download a taxonomy from the database:

1. If you are planning to edit the taxonomy, it is good to keep a safe copy for version control. You can either edit the downloaded copy directly, and even manage it through a versioning system, or keep the downloaded copy as the safety copy and edit the taxonomy directly through the Registry Console and save changes directly to the database.
2. You may wish to replicate the taxonomy for other systems in other departments of your organization. These departments or branches may even tailor the taxonomy for their own purposes.

To download the taxonomy, click the **Download** ( ) icon. This returns the system **Save file** dialog. The default name for the destination file is the taxonomy name with a `.xml` extension appended. Rename the file if you choose, then save the taxonomy file as you would any other.

1.5.7. Deleting Taxonomies

If at any point you decide that a taxonomy is no longer necessary, you can delete it by clicking the **Delete taxonomy** icon ( ) in the **Find Taxonomy** page.

Important

Because this procedure is irreversible you will be asked to confirm your deletion.

1.6. Replication Management

Selective One-way Replication is a subscription-based replication mechanism under which a slave registry retrieves update and change notifications from a master registry. The slave registry then applies these to its own data.

Replication is set up by a subscription defining the set of businessEntities or tModels being replicated. The subscription filter is a find_business or find_tModel query with no special requirements.

Each time replication is invoked, the slave registry retrieves a set of changed businessEntities and referenced tModels. The tModels are referenced in tModelKeys of either tModelInstanceInfos or keyedReferences. These changes are then saved.

Important

Referenced tModels are only replicated if the slave registry does not already contain them. If a tModel is already present in the slave registry, it will not be replicated to the slave registry, even if the tModel has been modified in the master registry.

Important

Replicated data should not be changed because such changes in the slave registry will be lost when someone changes these entities in the master registry and the replication is automatically processed. Note also that replicated data should be stored under an account having administrator's privileges (admin).

Replication may fail or produce warning messages. The failure may occur for one of the following reasons:

- The master registry is not accessible or the connection is broken during data replication;
- Saving/Deleting of a subscribed businessEntity on the slave registry fails.

A warning is produced when:

- The subscribed businessEntity is not accessible on the master registry. For example because of ACL GET denied permission;
- Referenced tModels are not accessible on the master registry;
- Referenced tModels are saved/deleted.

Replication tries to obtain all changes to subscribed data since the last successful replication.

Replication process logs can be found in the `REGISTRY_HOME/log/replicationEvents.log` file. You can edit the `REGISTRY_HOME/conf/log4j.config` and make replication logging more detailed by uncommenting the following statement:

```
log4j.category.replication_v3.com.systinet.uddi.replication.v3.ReplicatorTask=DEBUG,replicationLog
```

1.6.1. Master Registry Setup

To set up the master registry:

1. If you do not have an account on the master registry, you must create one. It can be a standard account.

   ### Note

   The default subscription limit for a new user is five. The BEA AquaLogic Service Registry Administrator may increase the subscriptions limit for the user.

2. Log into the master registry account.
3. Create a subscription for the replication with the following details:
   - The subscription filter must be a find_business or find_tModel query.
   - Set the **Notification listener type** drop down list to `None`
   - The `brief` option is recommended to reduce the amount of transferred data.

For more information, please see Section Publishing Subscriptions.

1.6.2. Slave Registry Setup

Note

Only the administrator of the slave registry should do this.

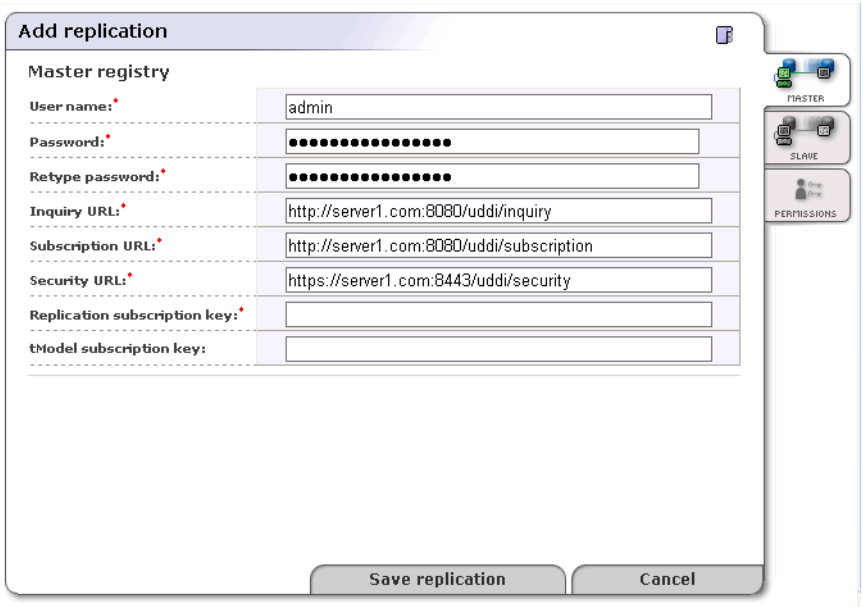
There are two parts to the slave registry configuration:

- Master registry information including the location of master registry endpoints for inquiry, subscription and security APIs, and the username/password pair on the master registry needed to obtain notifications;
- Slave registry information including the username/password pair on the slave registry for the user who will own the replicated data, and the notification interval.

To set up replication:

1. Log on as Administrator to the slave registry.
2. Click the **Manage** main menu tab, then click on the link **Registry management** under the **Manage** menu tab.
3. Click **Replication management**. This returns a list of replications.
4. Click **Add replication**.
5. Fill in the form under the **Master** tab as described in Figure 20.
6. Fill in the form under the **Slave** tab as described in Figure 21.
7. Specify permissions for replicated data under the **Permissions** tab as shown in Figure 22.
8. Click **Save replication**.

Figure 20. Add Replication Master



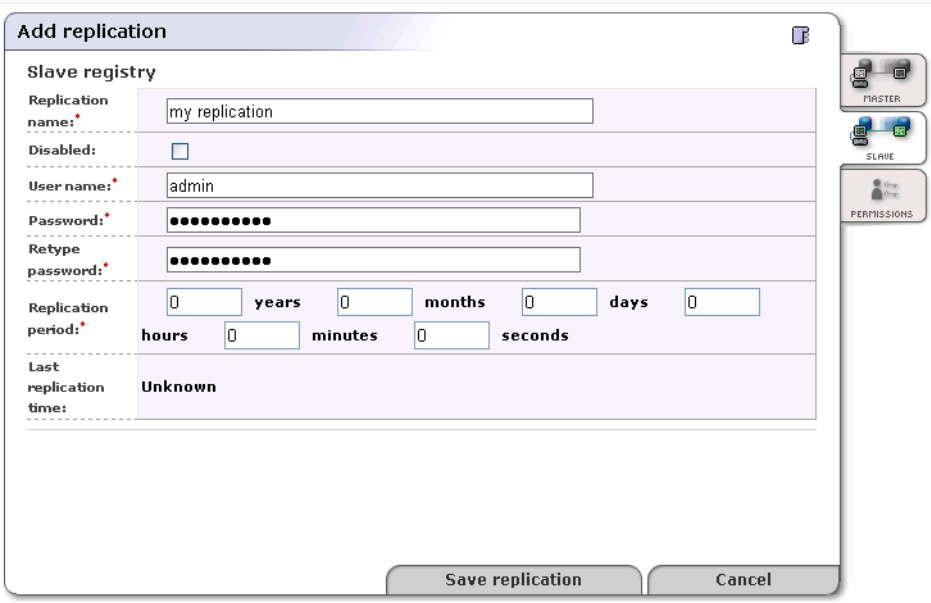
- **User name** - Name of the user who created the replication subscription on the master registry
- **Password** - Password of the user who created this subscription. This password is encrypted in the configuration file.
- **URLs of Master Registry** - All URLs (Inquiry URL, Subscription URL and Security URL) must refer to the same master registry. Moreover the URLs must not refer to the slave registry itself, otherwise you can loose some data.
  - **Inquiry URL** - Inquiry URL of master registry. For example, `http://master.mycompany.com:8080/uddi/inquiry`. The inquiry URL is used to obtain full standard UDDI v3 structures.

    Note

    UDDI v2 keys are not included in the UDDI v3 structure and replicated structures differ with regard to v2 keys. To replicate v2 keys, specify the URL of the proprietary inquiry API, which returns extended structures including v2 keys. This extended API has the context `/uddi/export`. For example, `http://master.mycompany.com:8080/uddi/export`.

  - **Subscription URL** - Master registry's subscription URL. For example, `http://master.mycompany.com:8080/uddi/subscription`.
  - **Security URL** - Master registry's security URL. For example, `https://master.mycompany.com:8443/uddi/security`.
- **Replication subscription key** - key of the `find_business` or `find_tModel` subscription from the master registry.
- **tModel subscription key** - key of the **helper** subscription for changes to tModels from the master registry.

Figure 21. Add Replication Slave



- **Replication name** - Name the replication for better orientation within the list of replications.
- **Disabled** - Check this box to disable replication.

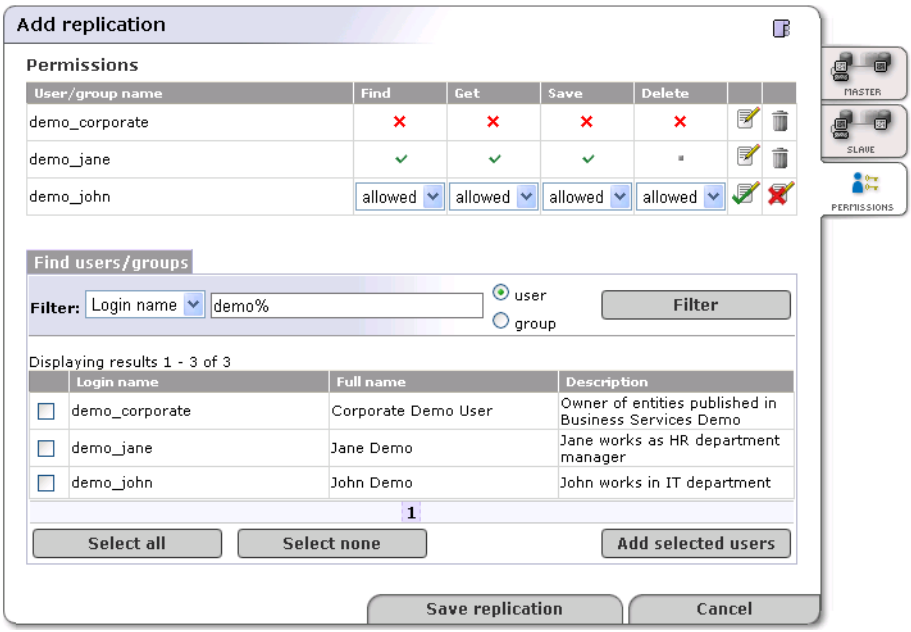
- **User name** - User account name under which replicated data will be stored.

Important

The user must have the `ApiManagerPermission` on `org.systinet.uddi.client.v3.UDDI_Publication_PortType` API for all * actions to be able to generate keys without having the appropriate keyGenerator. For more information, see User's Guide, Section 5.2.1, Generating Keys. By default, the only user who can do this is the admin.

- **Replication period** - Specify the period between replications by entering the appropriate number in the boxes for years, months, days, hours, minutes, and seconds. The default period is one hour.
- **Last replication time** - The date and time when the last replication occurred.

Figure 22. Add Replication Permissions



In the page shown in Figure 22, the administrator can set up permissions for replicated data. If you do not enter any data on this page, all users from the slave registry have `find` and `get` permissions on replicated data.

To specify permissions on replicated data:

1. Enter a filter criteria for users or groups, and click **Filter**.
2. Check the box in front of users or groups. Then, click the **Add selected users** button. Selected users or groups will be added to the permissions list.
3. Click the **Edit** icon to change permissions for `Find`, `Get`, `Save` and `Delete` operations
4. Click the **Save replication** button.

Tip

Use the button **Replicate now** on the **replication** page to test the replication settings.

1.7. Approval Process Management

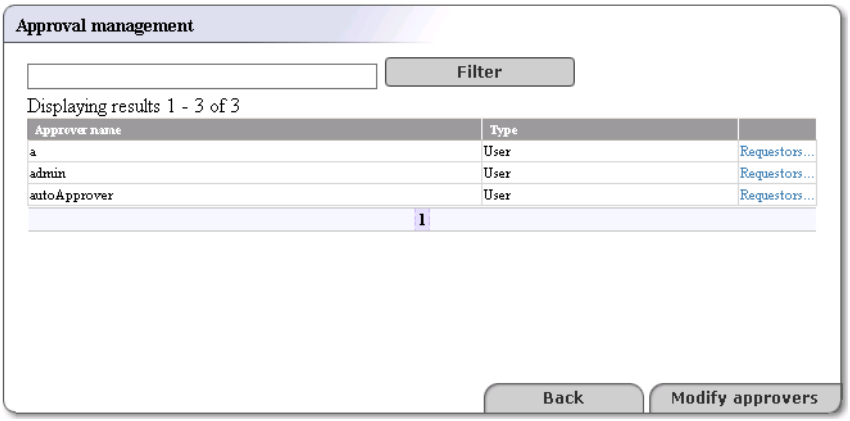
This chapter describes how administrators can manage the approval publishing process. We will show you how to set up requestors and approvers using the Registry Console. Before you start, we recommend that you read Section 6, Approval Process Principles.

1.7.1. Loading the Approval Management Page

The tasks described in this section are performed from the Approval management page. To load this page:

1. Log on as administrator.
2. Click the **Manage** main menu tab, then select the **Registry management** link under the **Manage** menu tab.
3. Click **Approval management**. This returns a list of approvers similar to that shown in Figure 23.

Figure 23. Approval Management



1.7.2. Create Approver

To create an approval contact:

1. Click the **Modify approvers** button on the Approval management page shown in Figure 23
2. This returns the Modify approvers page as shown in Figure 24

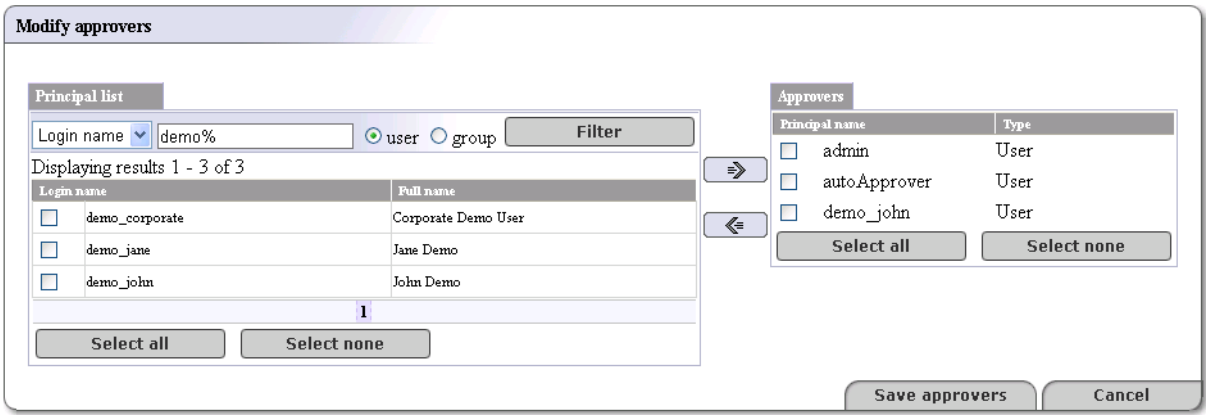
   The left side of this page, labeled **Principal list** is a list of all users and groups on the registry. The administrator may make any name on this list into an approval contact.

   The right side, labeled **Approvers** is a list of all approvers on the registry.
3. Check the box next to the login name of a user you would like to turn into an approver and click the right-facing arrow (⇨). If you would like to create an approver from a group, check the group box and use the right-facing arrow.
4. Click the **Save approvers** button.

Tip

Using the left-facing arrow buttons, you can deselect approvers in the same way.

Figure 24. Modify Approvers



1.7.3. Create Requestor

To create a requestor from a user:

1. On the Approval management page shown in Figure 23 click the link labeled **Requestors** next to an Approver type.
2. This returns the Modify requestors page shown in Figure 25

   The Requestors page consists of two lists:

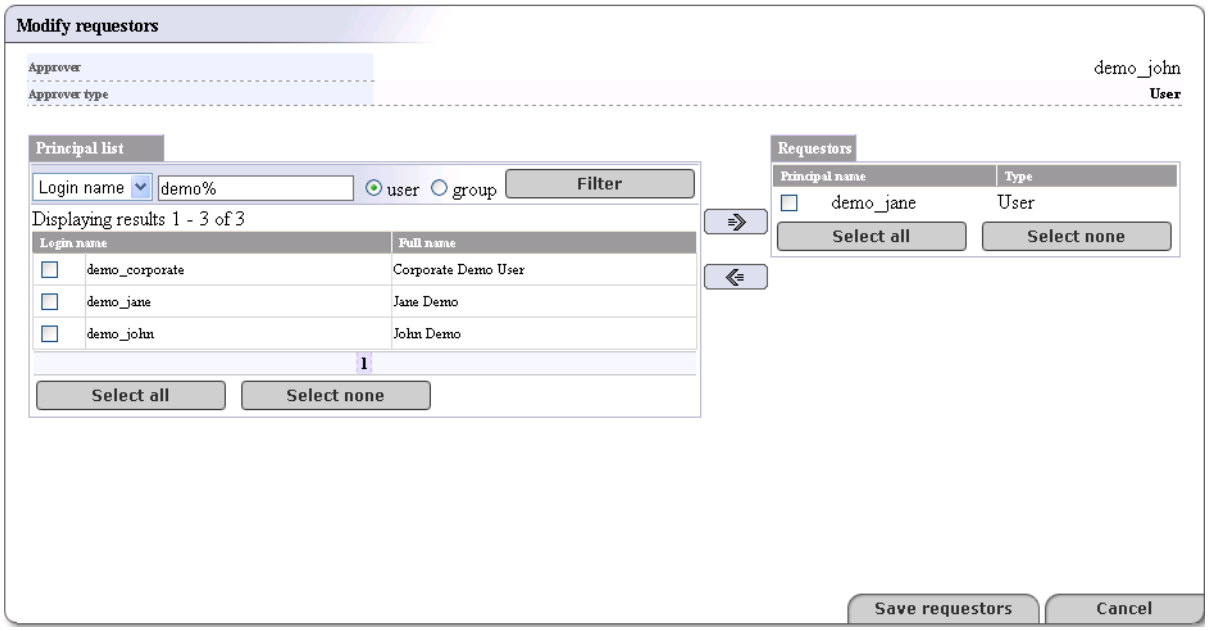
   - A list of all users and groups on the registry labeled **principals**
   - A list of users and groups, labeled **Requestors** assigned to the selected approver

3. Select a user or group from the **Principals** column (or click **Select all** if you choose), and click the right-pointing arrow (⇨) to turn the user(s) into requestors.
4. Click the **Save requestors** button.

Tip

Using the left-pointing arrow button, you can deselect requestors in the same way.

Figure 25. Modify Requestors



1.8. Replacing UDDI Keys

Replacing keys of businessEntities, businessServices, tModels, and bindingTemplates is intended to correct errors in keys before entities are commonly used by users.

To access the key replacement page:

1. Log on as administrator.
2. Click the **Registry management** link under the **Manage** tab.
3. In the row labeled **Replace UDDI keys**, click the appropriate button **tModel**, **business**, **service**, or **binding**.

Important

The replace key operation can break digital signatures on changing entity as well as on other entities which reference to the changing entity.

1.8.1. Replacing tModel keys

When you replace a tModel key, the key will be updated in the following data structures:

- tModel
- keyedReferenceGroups
- keyedReferences
- tModelInstanceInfos
- publisherAssertions
- addresses

- taxonomies

1.8.2. Replacing businessEntity keys

When you replace a businessEntity key, the key will be updated in the following data structures:

- businessEntity
- services
- keyedReferences

1.8.3. Replacing businessService keys

When you replace a businessService key, the key will be updated in the following data structures:

- businessService
- bindingTemplates
- keyedReferences

1.8.4. Replacing bindingTemplate keys

When you replace a bindingTemplate key, the key will be updated in the following data structures:

- bindingTemplate
- keyedReferences
- subscriptions
- hostingRedirector
- accessPoint with `bindingTemplate` useType

1.9. Registry Statistics

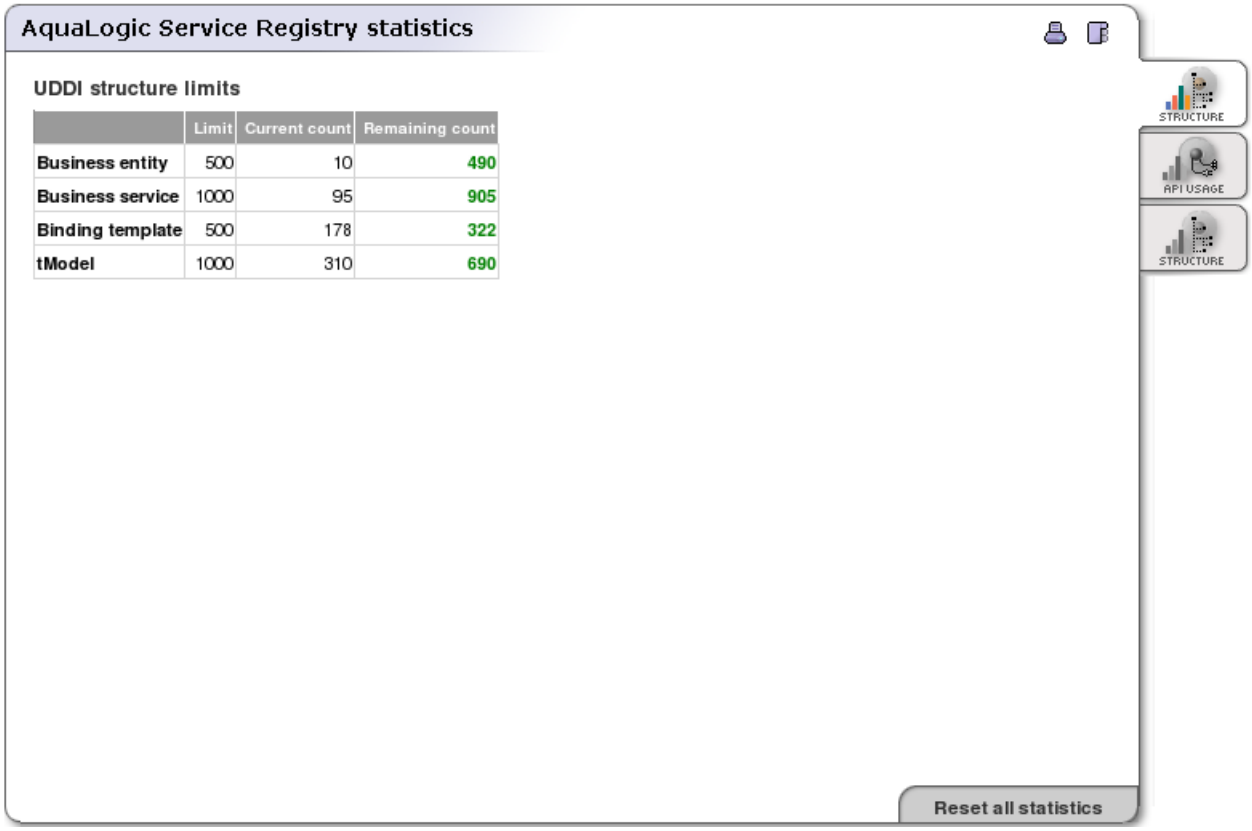
Registry statistics include statistics on::

- UDDI structure counts versus limits imposed by the product license;
- invocations of registry APIs;
- UDDI structure counts generally;

To access the registry statistics page:

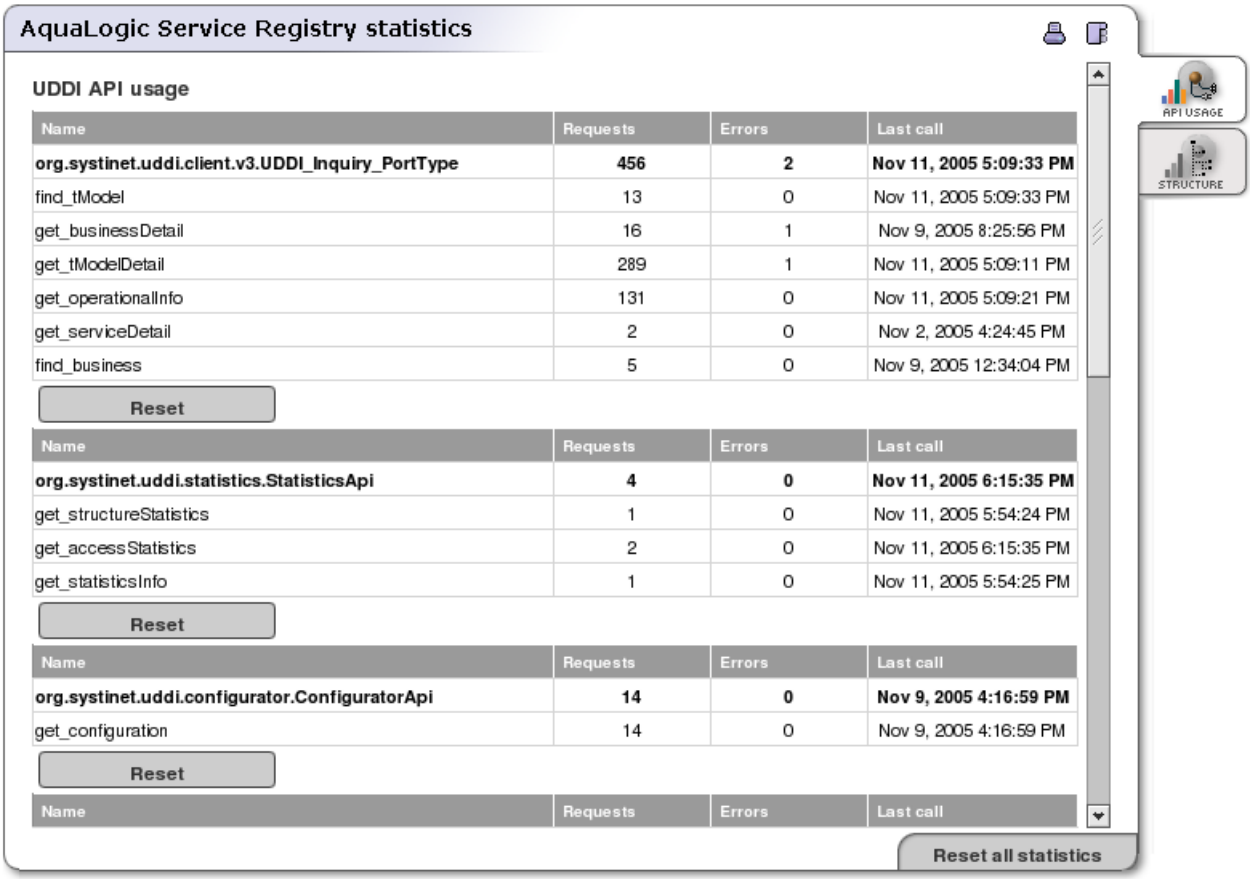
1. Log on as administrator.
2. Click the **Registry management** link under the **Manage** tab.
3. Click the **Statistics** button.
4. The page similar as shown in Figure 26 will appear, summarizing publishing limits imposed by the product license, current counts and the number remaining.

Figure 26. Statistics - Publication Limits



5. Click the **API Usage** tab and you will see a page as in Figure 27 showing the number of requests for each API, number of unsuccessful requests and datetime of last API call. You can reset count separately for each API by clicking the **Reset** button or reset counts for all API by clicking on the **Reset all statistics**.

Figure 27. Statistics - API usage



Name	Requests	Errors	Last call
org.systinet.uddi.client.v3.UDDI_Inquiry_PortType	**456**	**2**	**Nov 11, 2005 5:09:33 PM**
find_tModel	13	0	Nov 11, 2005 5:09:33 PM
get_businessDetail	16	1	Nov 9, 2005 8:25:56 PM
get_tModelDetail	289	1	Nov 11, 2005 5:09:11 PM
get_operationalInfo	131	0	Nov 11, 2005 5:09:21 PM
get_serviceDetail	2	0	Nov 2, 2005 4:24:45 PM
find_business	5	0	Nov 9, 2005 12:34:04 PM

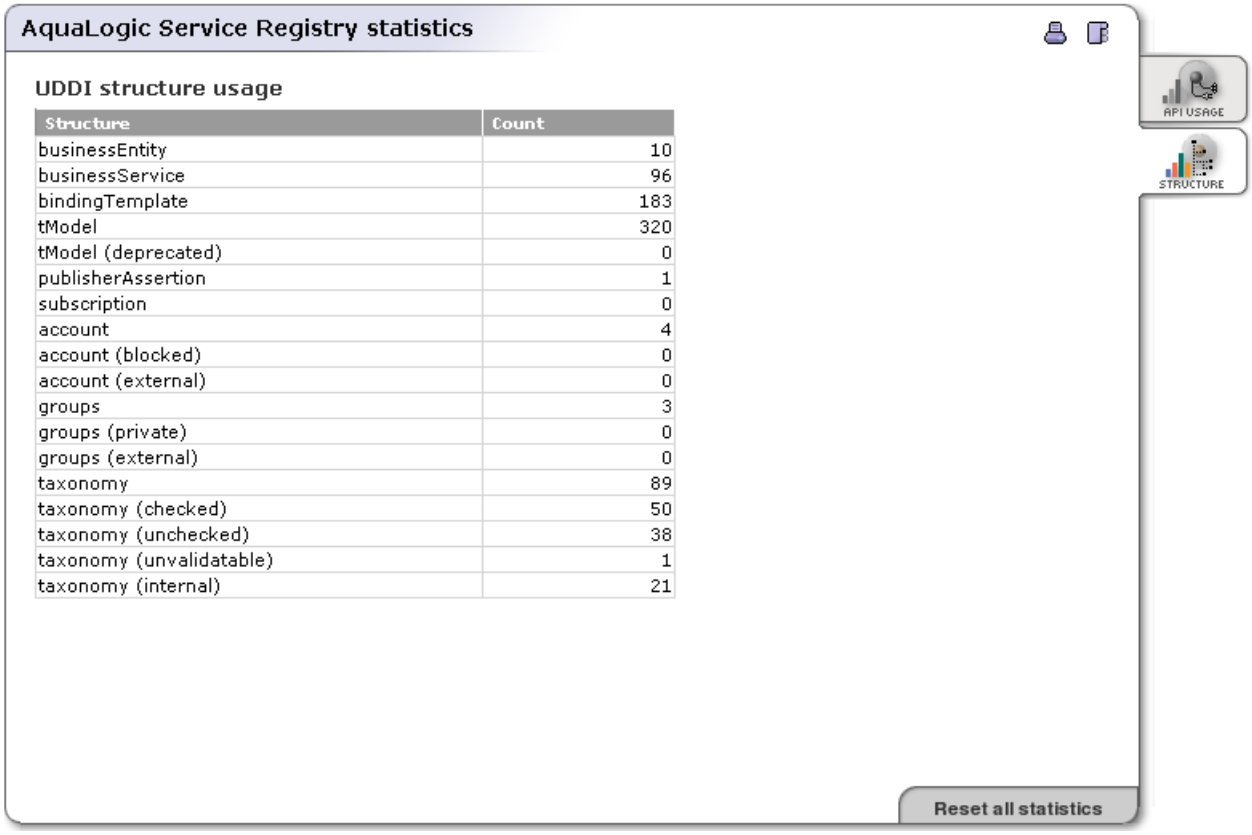
Name	Requests	Errors	Last call
org.systinet.uddi.statistics.StatisticsApi	**4**	**0**	**Nov 11, 2005 6:15:35 PM**
get_structureStatistics	1	0	Nov 11, 2005 5:54:24 PM
get_accessStatistics	2	0	Nov 11, 2005 6:15:35 PM
get_statisticsInfo	1	0	Nov 11, 2005 5:54:25 PM

Name	Requests	Errors	Last call
org.systinet.uddi.configurator.ConfiguratorApi	**14**	**0**	**Nov 9, 2005 4:16:59 PM**
get_configuration	14	0	Nov 9, 2005 4:16:59 PM

Name	Requests	Errors	Last call

6. You can click on the **Structure** tab. The page similar as shown in Figure 28 appears. On that page you can see number of UDDI entities stored in Systinet Registry.

Figure 28. Statistics - Structure



Structure	Count
businessEntity	10
businessService	96
bindingTemplate	183
tModel	320
tModel (deprecated)	0
publisherAssertion	1
subscription	0
account	4
account (blocked)	0
account (external)	0
groups	3
groups (private)	0
groups (external)	0
taxonomy	89
taxonomy (checked)	50
taxonomy (unchecked)	38
taxonomy (unvalidatable)	1
taxonomy (internal)	21

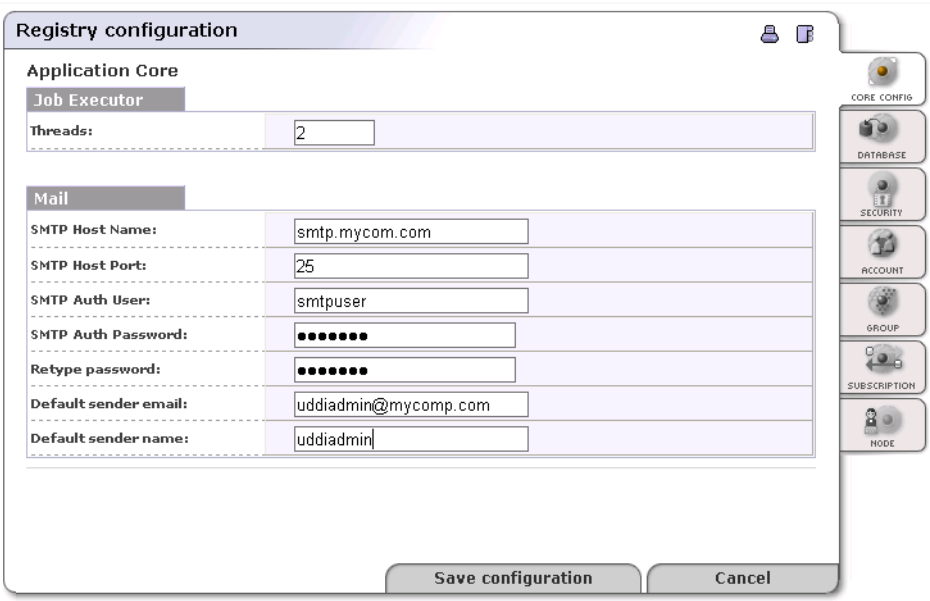
2. Registry Configuration

Registry configuration is used whenever you want to set up the database, registry parameters, or account properties.

To access Registry configuration:

1. Log on as administrator or as a user with privilege to display the **Manage** tab. For more information, see Rules to Display the Manage Tab.
2. Click the **Manage** main menu tab.
3. Select the **Registry configuration** link under **Manage** tab. This returns the Registry configuration panel shown in Figure 29.

Figure 29. Registry Configuration



The Registry configuration panel includes the following tabs:

- Core Config
- Database
- Security
- Account
- Group
- Subscription

In this part of the chapter, each of these sections settings is described in detail. Fields marked with an asterisk (*) are the most important.

2.1. Core Config

Threads

Maximum number of threads used in statement execution

The default is 2.

Mail

SMTP Host Name, SMTP Host Port, SMTP Auth User, SMTP Auth Password, Default sender email, and Default sender name are used to set up the entity that sends emails on behalf the registry administrator.

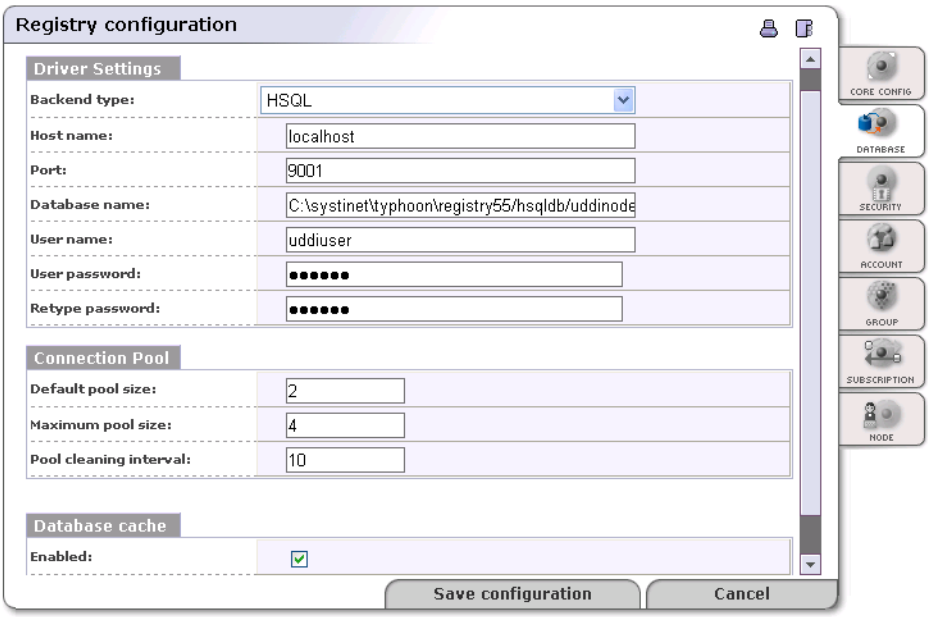
2.2. Database

This section details how to set up the database connection. The default values are set according to the database chosen at installation. For details, please see Table 1, “Default Ports for Supported Database Servers”.

Note

Database installation, that is, creating the database schema and loading basic data, is described in the Installation Guide [http://edocs.bea.com/alsr/docs21/install/].

Figure 30. Registry Configuration - Database



Backend type *

A menu of databases from which to select the vendor of your database.

Hostname *

Database host name or IP address, for example, `dbserver.mycompany.com`

Port *

Database port number. For default values see Table 1, “Default Ports for Supported Database Servers”. Note that if you are using the HSQL database, it is embedded in the same JVM and therefore the port number is ignored in this case.

Database Name *

Database name; for example, `uddinode`

User Name *

User name; `uddiuser` by default

User Password *

Database user password;`uddi` by default

Default pool size

Count of concurrent database connections initialized at start time

Max pool size

Maximum count of concurrent database connections. Each request books one connection until the request is served. If all connections are booked and new request comes in, the connection pool creates a new connection till the maximum count is reached. If this maximum is reached and new request comes in, this request must wait for a free connection to be released by a previous request.

Pool cleaning interval

How often database connections are closed over the default count. This value represents time in hours.

Database cache

This is used for performance optimization.

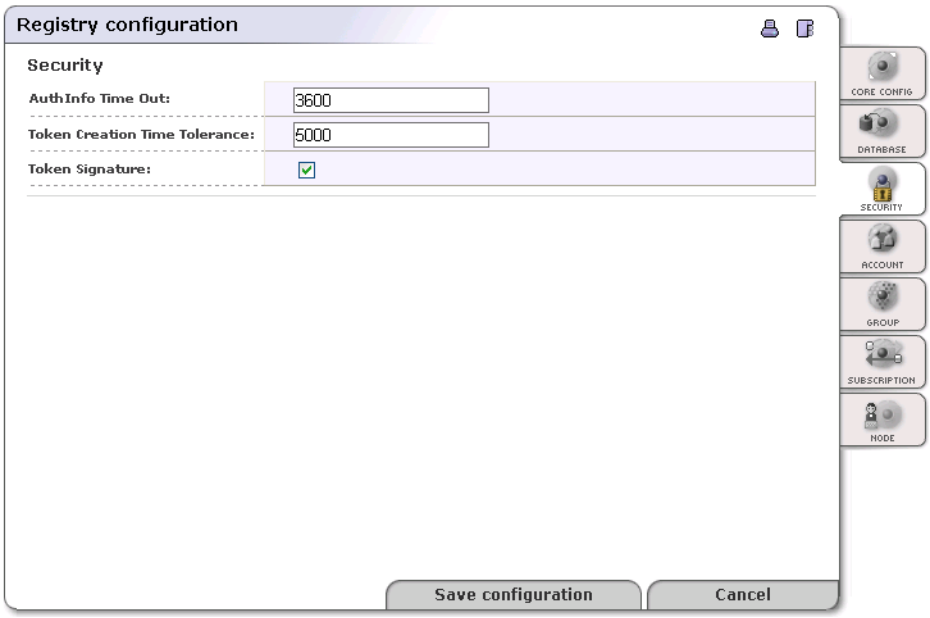
Table 1. Default Ports for Supported Database Servers

Database	Default Port
Oracle 8i	1521
MS SQL 2000	1433
DB2 8.0	6789
Sybase ASE 12.5	5000
PostgreSQL	5432
hsqldb 1.7.3	-

2.3. Security

On the **Security** tab, you can configure your digital signature token and key properties.

Figure 31. Registry Configuration - Security



AuthInfo Time Out

Authorization token is obtained by invoking the `get_authToken` method. This token is used for each operation on the publishing port. Here you can set up the authorization token time-out in seconds. The default value is one hour.

Token Creation Time Tolerance

Tolerance interval of token validity, expressed in milliseconds

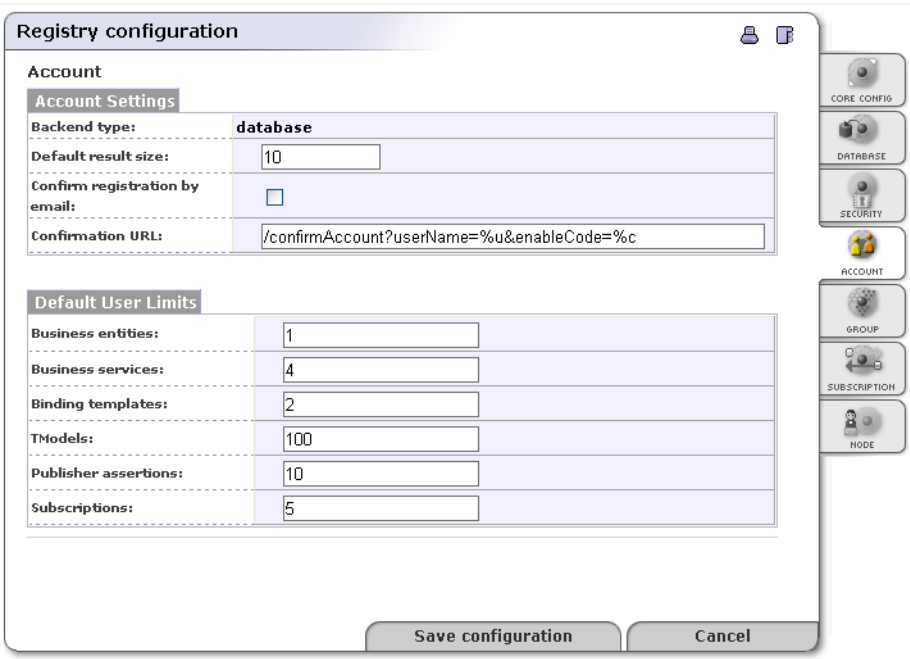
Token Signature

Whether authorization token is signed. We recommend you toggle this switch on.

2.4. Account

On this tab, you can specify accounts properties applicable for all BEA AquaLogic Service Registry user accounts.

Figure 32. Registry Configuration - Account



Backend type

This field is not editable. Its value is specified during installation.

Default result size

Number of items returned in search results when querying accounts

Confirm registration by email

Check this box if you would like new users to confirm account creation.

Confirmation URL

URL where new users can confirm registration

Default User Limits Limits are used as default values only when creating a new account. Accounts that exist at the time of change are exempt from new limit values. Limits for existing accounts can be updated with the Account Management tool.

Business entities

Business entity limit; default is 1.

Business services

Number of allowed business services per business entity; default is 4.

Binding templates

Number of allowed bindingTemplates per businessService; default is 2.

TModels

Number of allowed tModels; default is 100.

Publisher assertions

Number of allowed relationship assertions; default is 10.

Subscriptions

Number of allowed subscriptions saved by user. Default is 5.

2.5. Group

On this tab, you can specify the properties of the group API.

Backend type

Not editable, this field's value is specified during installation.

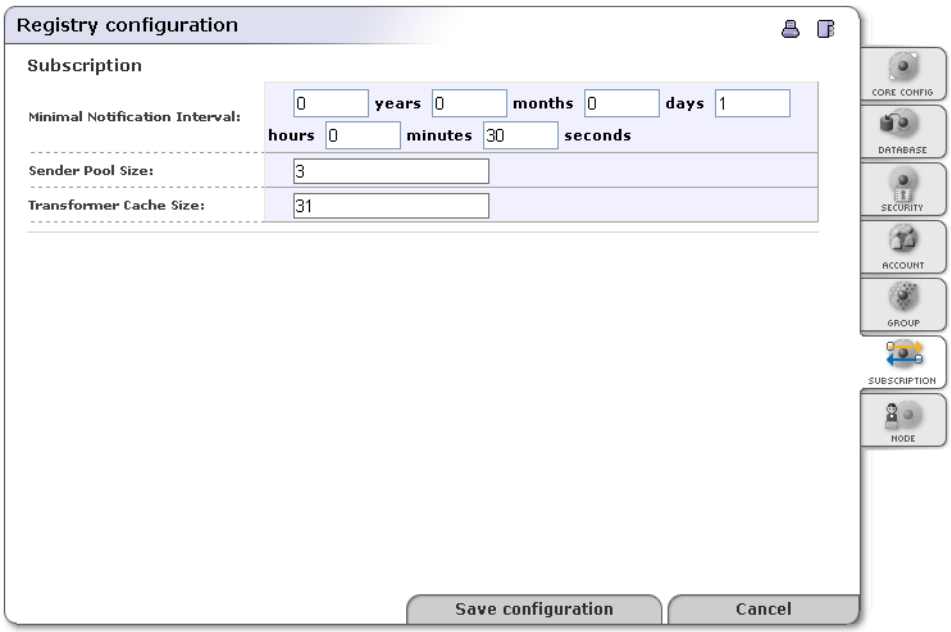
Default result size

Number of items returned in search results when querying groups; the default value for this field is 10.

2.6. Subscription

On the **Subscription** tab, you can configure server limits for subscriptions. If a user saves a subscription which does not match these limits, the registry automatically adjusts the user's values.

Figure 33. Registry Configuration - Subscriptions



There are three fields to configure on this tab:

Min. notification interval

Minimal interval between notifications provided to a subscriber

Sender Pool size

Number of stubs ready for notification

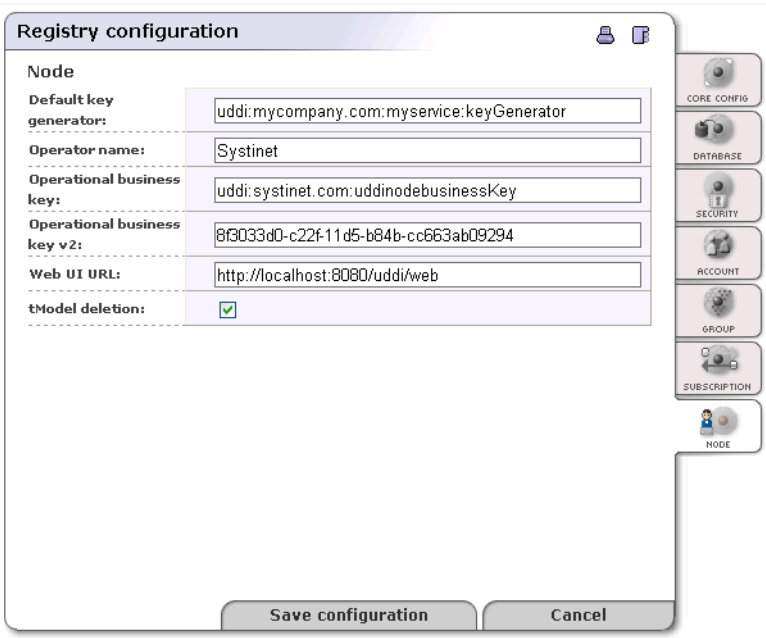
Transformer Cache Size

Number of cached XSLT transformations

2.7. Node

On the **Node** tab, you can configure UDDI node properties.

Figure 34. Registry Configuration - Node



Default key generator

The Default Key generator tModel allows the Registry to generate keys in the form `domain:string` instead of only in the form `uuid`. For example, `uddi:mycompany.com:myservice:61c08bf0-be41-11d8-aa33-b8a03c50a862` instead of only `61c08bf0-be41-11d8-aa33-b8a03c50a862`. Enter the key of the tModel that is the key generator. For example, if you enter `uddi:mycompany.com:myservice:keyGenerator`, keys will be generated with the prefix `uddi:mycompany.com:myservice:`. For more information, please see Section 5.2, Publisher-Assigned Keys in the User's Guide.

Operator name

The name of the operator of the UDDI node. The default entry for this field is configured during installation.

Operational business key

The key of the Operational business entity. This entity holds miscellaneous registry settings such as the validation service configuration.

Operational business key v2

The key of the Operational business entity in UDDI v2 format.

Web UI URL

The URL of the Registry Console.

tModel deletion

If this box is checked then deleted tModels are deleted permanently. Otherwise, tModels are marked as deprecated. (Deprecated tModels are visible by direct get tModel call, but do not appear in any search results.)

3. Business Service Console Configuration

Under the **Configuration** tab of the Business Service Console the administrator can configure the following:

- The tabs that will be displayed for users who have a specific user profile
- Types of result view for each user profile
- Browsable Taxonomies
- Result paging limits
- Configuration of the Business Service Console User Interface
- Customizable Taxonomies providing for user input when creating, editing or searching entities

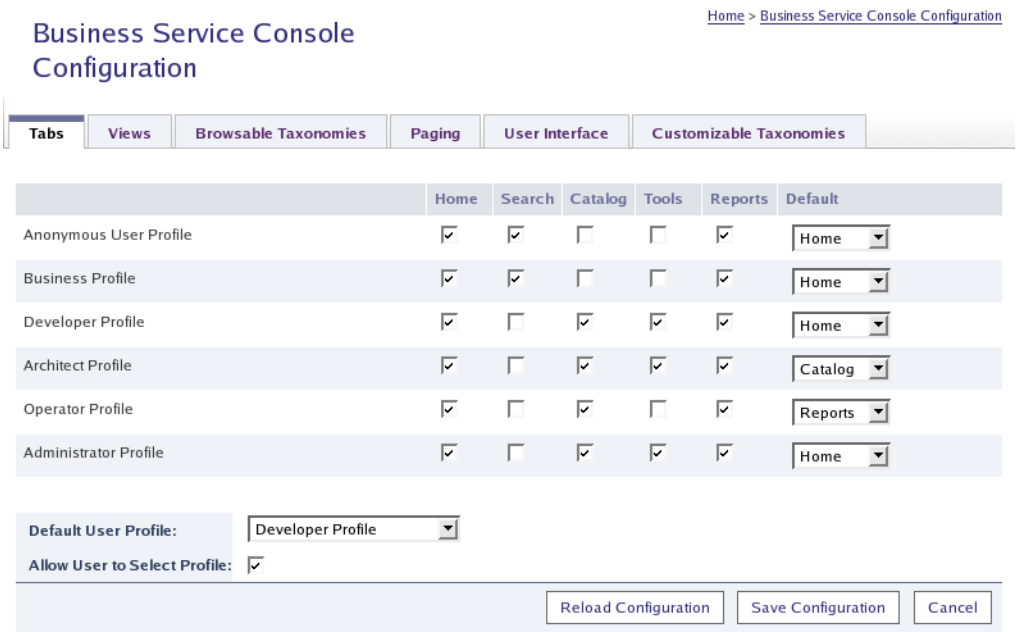
The **Configuration** tab is available if both of the following conditions are satisfied:

- The user belongs to a user profile that has the visible **Configuration** tab
- The user has `ConfiguratorManagerPermission` to all operations (*) and all configurations (*). See Administrator's Guide, Section 1.4, Permissions for more information on how to set up permissions.

Furthermore, administrators can customize individual pages wherever a **Customize** button appears.

3.1. Tabs Displayed

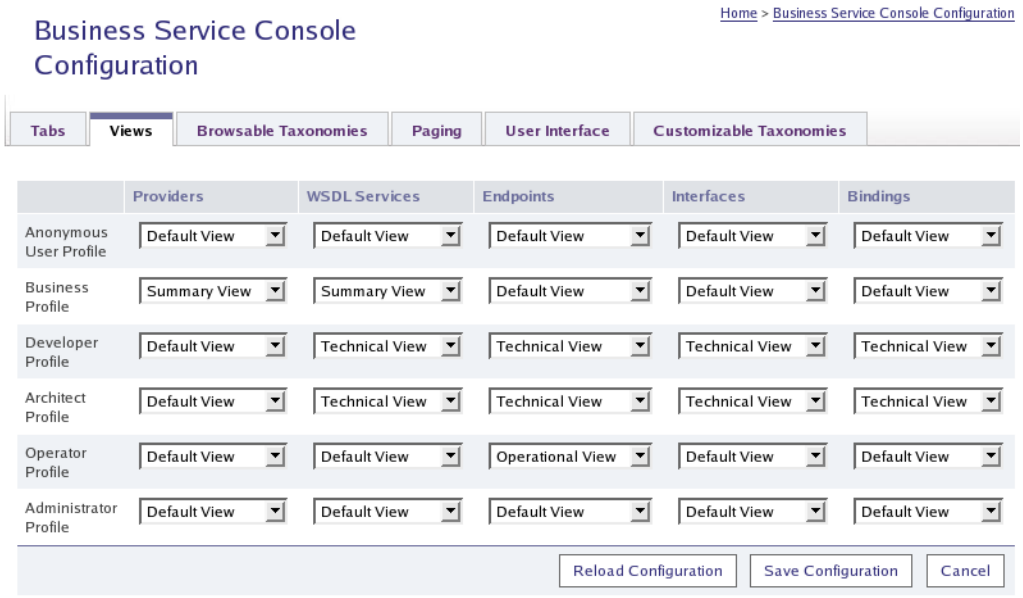
Figure 35. Business Service Console Configuration - Tabs Displayed



On the page shown in Figure 35, you can define which tabs will be available for specific user profiles. The **Default User Profile** drop down list allows you to specify the default user profile when creating a new user account. If the checkbox **Allow User to Select Profile** is checked, users are allowed to select a user profile when creating a new account, later users can switch profiles.

3.2. Search Result View

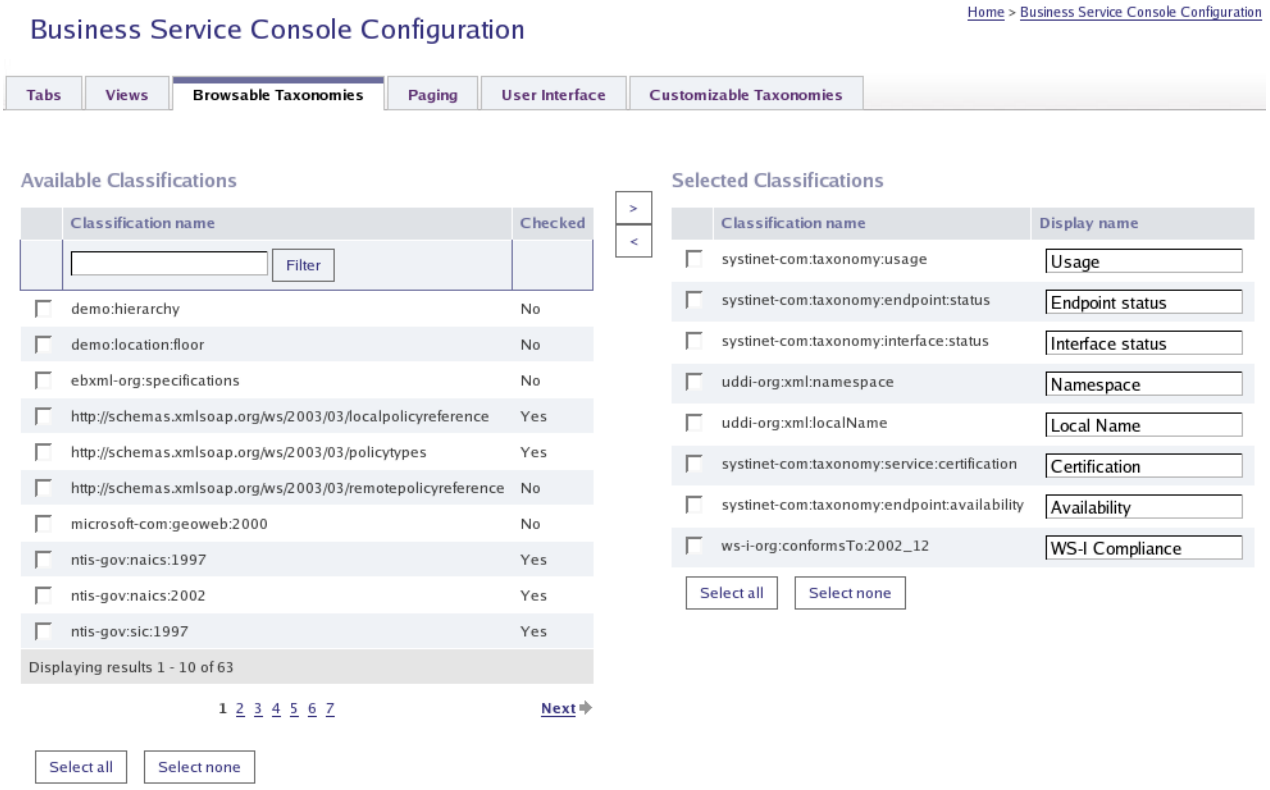
Figure 36. Business Service Console Configuration - Search Result View



On the page shown in Figure 36, you can configure default result views for user profiles.

3.3. Browsable Taxonomies

Figure 37. Business Service Console Configuration - Browsable Classifications



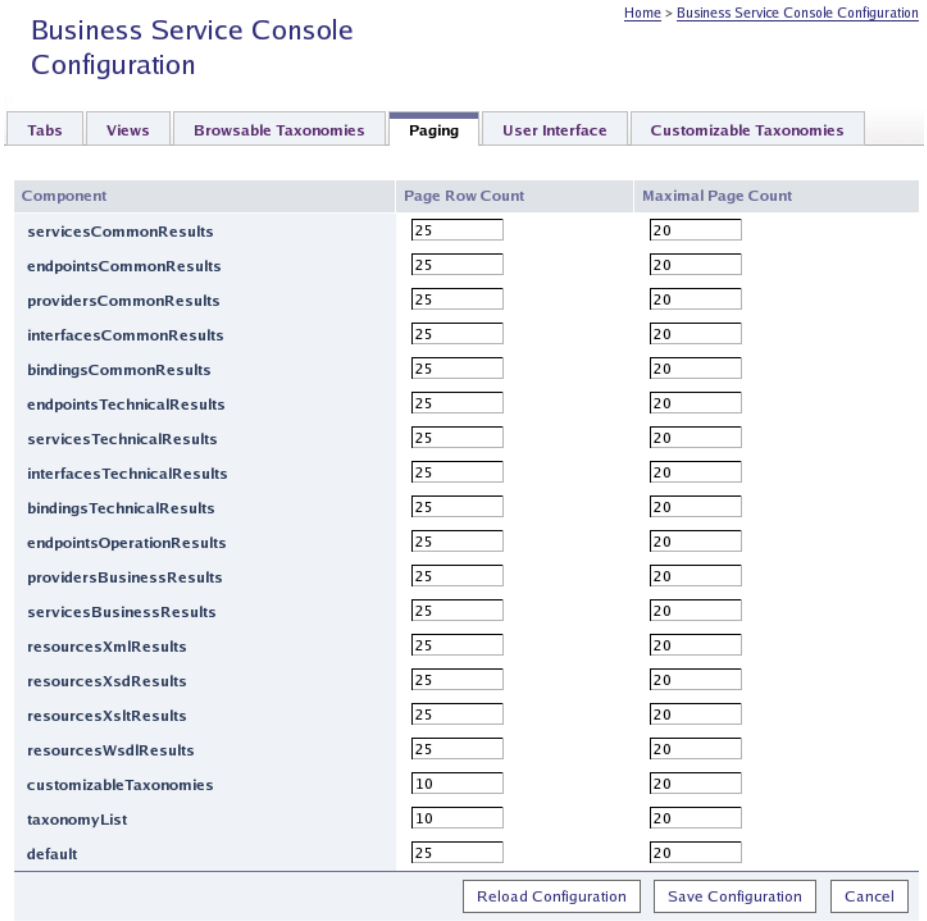
On this panel, you can choose which classifications (taxonomies) are *browsable*. Browsable taxonomies appear on the reports tree on the **Reports** tab, and also show up when viewing an entity's classification details.

Each browsable classification is displayed as a node in the Reports tree, using the **Display name** configured on the panel. If the taxonomy classification is internally checked - meaning it has a predefined set of values - a sub-node is displayed in the Reports tree for each possible value.

For example, the selected classification `systinet-com:taxonomy:service:certification` represents a node `Certification` in the **Report** tree. If you click on the **Certification** node in the report tree, the result view will contain all entities categorized by this taxonomy. Since the `systinet-com:taxonomy:service:certification` is internally checked, having the value set (Certified, Pending), the **Certification** node will contain two subnodes (Certified and Pending) representing a report of certified and pending services.

3.4. Paging Limits

Figure 38. Business Service Console Configuration - Paging Limits

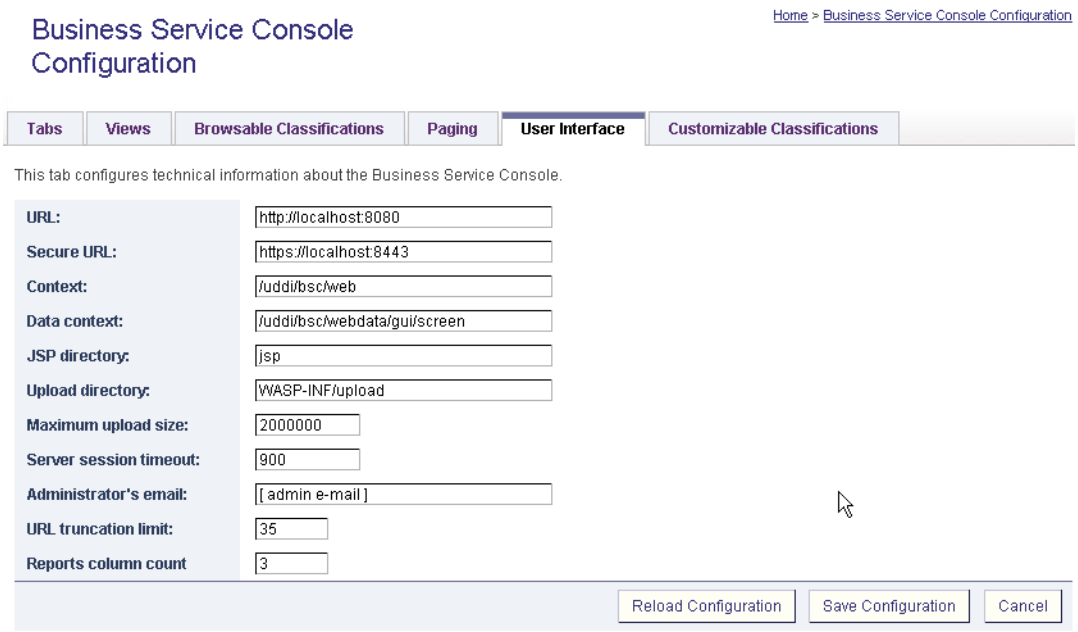


On this panel, you can specify how many records and on how many pages searched data will appear. Component names from the **Components** column consist of the component name (services, endpoints, providers, interfaces, bindings) and the type of result view (common, technical, business). For example, the row with the component name `servicesTechnicalResult` contains page limits for search results of services listing technical service data.

3.5. UI Configuration

On the Web Interface tab of the Business Service Console Configuration screen, you can configure URLs, contexts, directories, and other information related to the registry's interface.

Figure 39. Business Service Console Configuration - UI Configuration



Field description:

- **URL** - nonsecure registry URL
- **Secure URL** - secure registry URL
- **Context** - context of the Registry Console URL
- **Data context** - context where static objects such as JavaScript and images are stored
- **JSP directory** - location of JSP pages relative to `REGISTRY_HOME/work/uddi`
- **Upload directory** - upload directory used for tasks such as uploading taxonomies
- **Maximum upload size** - maximum upload size in bytes
- **Server session timeout** - session timeout (measured in seconds)
- **Administrator's email** - email address of the registry administrator.
- **URL Truncation Limit** - URLs displayed in reports and result views will be truncated to number of characters specified in this field. The truncated URL will not be exactly so long as the value specified here but the URL string can be a little bit longer. The truncated URL will be displayed in the following format:`<protocol><server name><truncated part ...><filename>`

3.6. Customizable Taxonomies

This tab controls which taxonomies are used in the **Search**, **Edit** or **Publish** pages, and how they are displayed.

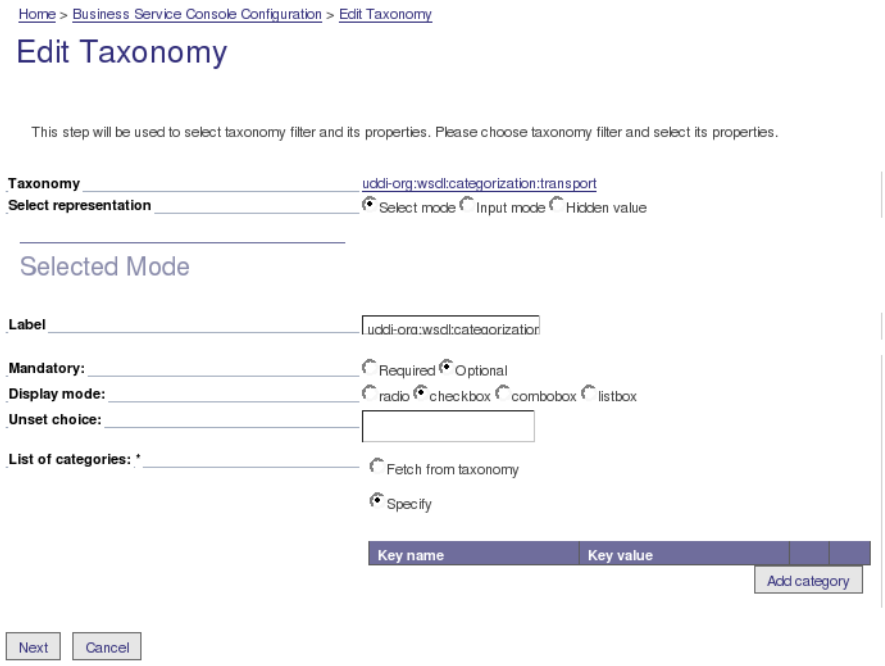
Figure 40. Customizable Taxonomies

To add a new taxonomy, click **Add New Taxonomy** at the bottom of the screen. To change how a taxonomy is currently displayed, click the **Edit** icon in the right-hand column.

The wizards for adding and editing a taxonomy (its representation) are similar. Here we describe the procedure for editing a taxonomy:

1. Click the icon in the **Edit** column for a taxonomy and you will be presented with a page as shown in Figure 41.

Figure 41. Configuring a customized taxonomy's representation



2. The details in the lower half of this page depend on the selection labeled **Select representation**:

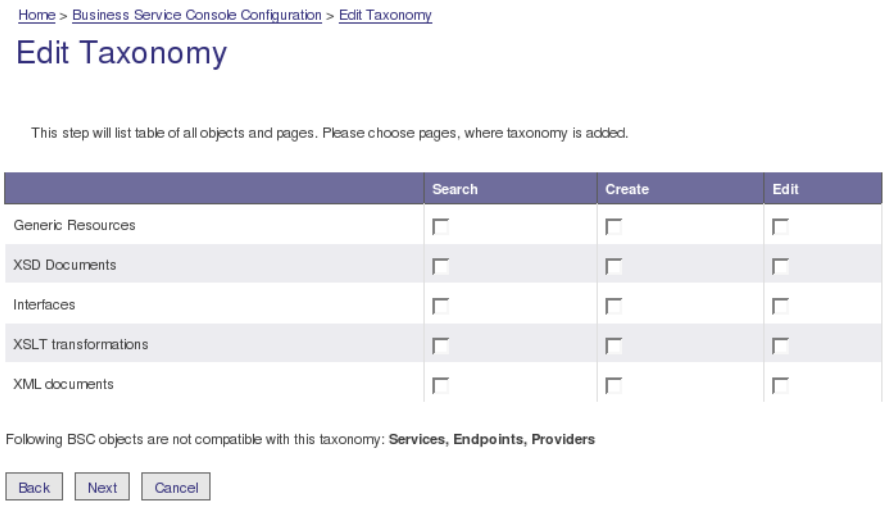
   Select mode Users select a value from a predefined set of valid values. This set can be displayed using one of the supported UI controls - checkboxes, radio buttons, listbox, etc. For checked taxonomies, the UI can fetch the valid values from the taxonomy itself - so providing values here is optional. Doing so allows you to limit users to a subset of values, and control the order in which they are displayed.

   Input mode Users *input* a value in a text box.

   Hidden value In this case it is not appropriate for the user to edit the value.

3. The next screen allows you to specify the pages to which this representation of the taxonomy will be added:

Figure 42. Selecting pages where a customized taxonomy appears



You can make it possible for the user to enter a value when an entity is created and/or edited, or to use the taxonomy in searches.

4. Click **Next** and you will be asked to specify where the representation appears on each additional page for which it is configured.

Figure 43. Specify positioning on pages

5. If there are any conflicts between the new and existing configurations, you will be asked to resolve them.. If you are adding the representation to a page where a different representation already exists then you will be asked to choose the new or existing representation.

6. Finally you will be presented with a summary of the additions.

3.7. Customizing Individual Pages

Administrators can customize individual pages of the Business Service Console wherever a **Customize** button appears. Pages sometimes have more than one composite area, in which case each can have its own **Customize** button.

Note

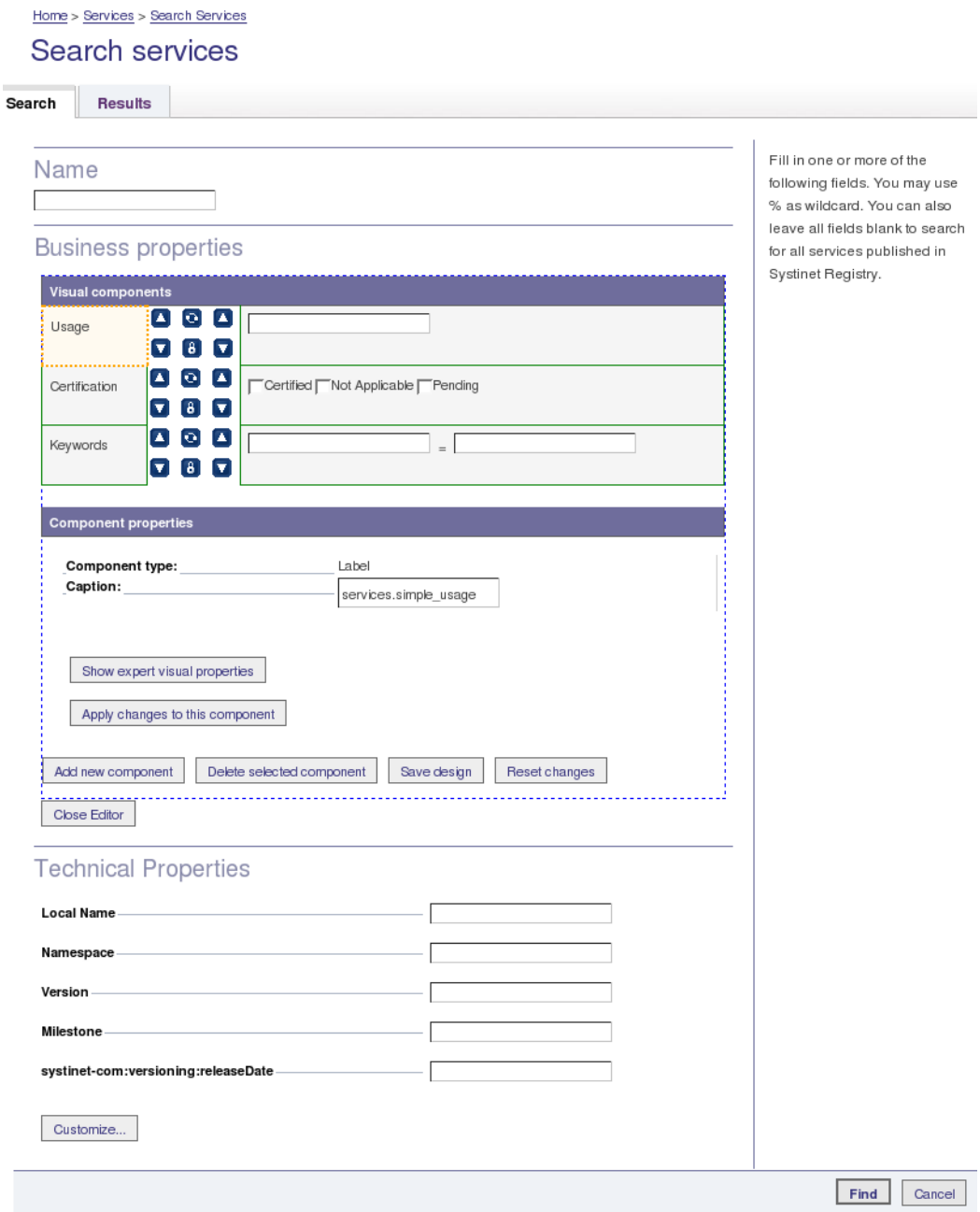
The **Customize** buttons on individual pages take precedence over the Customizable Taxonomies settings discussed above. This allows registry administrators to further customize individual pages to best meet their needs.

For example, Figure 44 shows a page with two composite areas:

- Business Properties
- Technical Properties

The user (an administrator) has clicked the **Customize** button in the Business Properties area.

Figure 44. Customizing a Page



The result is that the Business Properties are displayed in the customization editor, whereas the Technical Properties are displayed as usual in this page.

The customization editor displays:

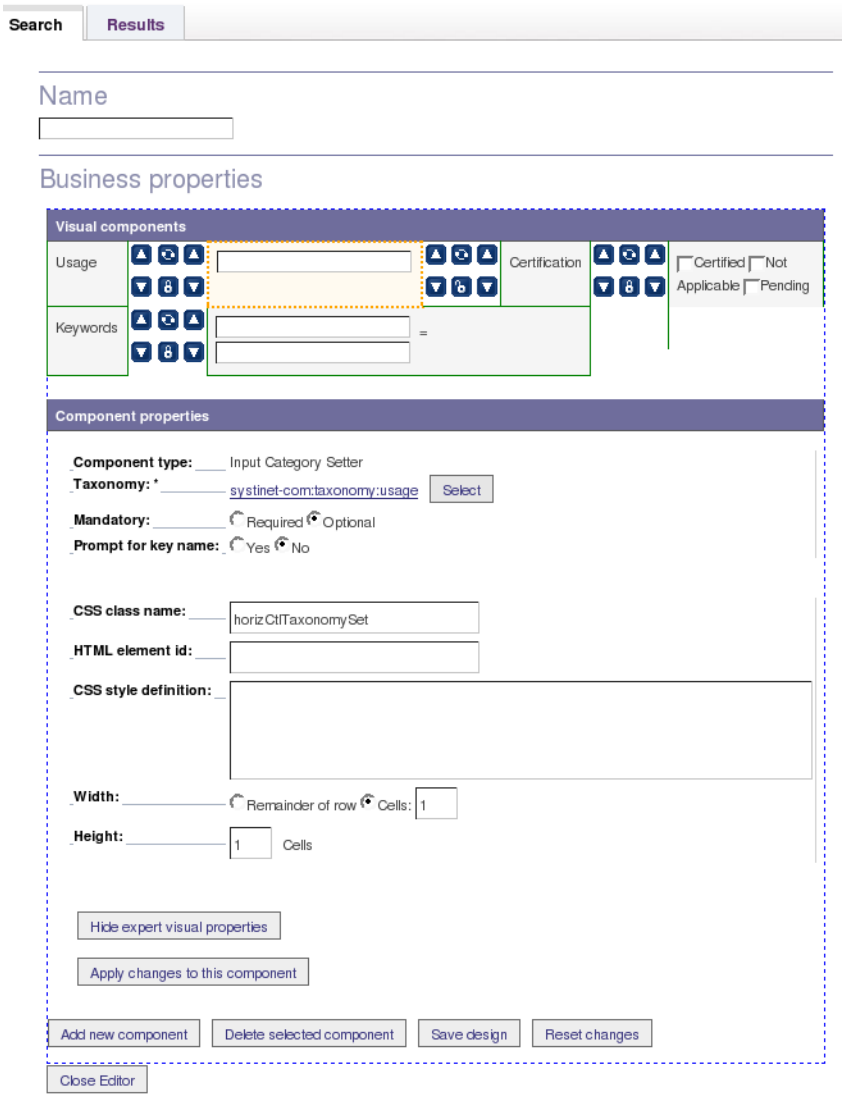
- **Visual Components** in a table, one row line. In this case there are 2 components in each line but see below. One component is selected and in this case it is the label **Usage**;
- **Component Properties** shows the properties of the selected component;

Under **Visual Components** each pair of adjacent components has a number of buttons between them. In this case there is only one set of buttons per line because there are only two components per line. The tool-tip for each button shows what it does. You can:

- Swap the positions of a pair of adjacent components horizontally;
- Move the component down or up, swapping it with the component below or above;
- Link a pair of adjacent components together so that when they are moved up or down the are moved together. Or you can unlink components that are linked;

Some of the details under **Component Properties** depend on the type of component. If you click **Show expert visual properties** it is possible to change the number of rows or columns occupied by a component - its **Height** and **Width**. The last component on the line has **Remainder of the row** checked. If you check **Cells** instead then the row is joined with the following row to make one line. For example, in Figure 45 the first two lines have been joined into one line of 4 components.

Figure 45. Expert visual properties



It is possible to perform the following actions by clicking the buttons provided:

- Add a new component;
- Delete the selected component;
- Save the design;
- Reset the changes you have made;
- Close the customization editor;

4. Registry Console Configuration

This section provides you with a catalog of web engine parameters.

Initially almost every web engine parameter is set correctly by default.

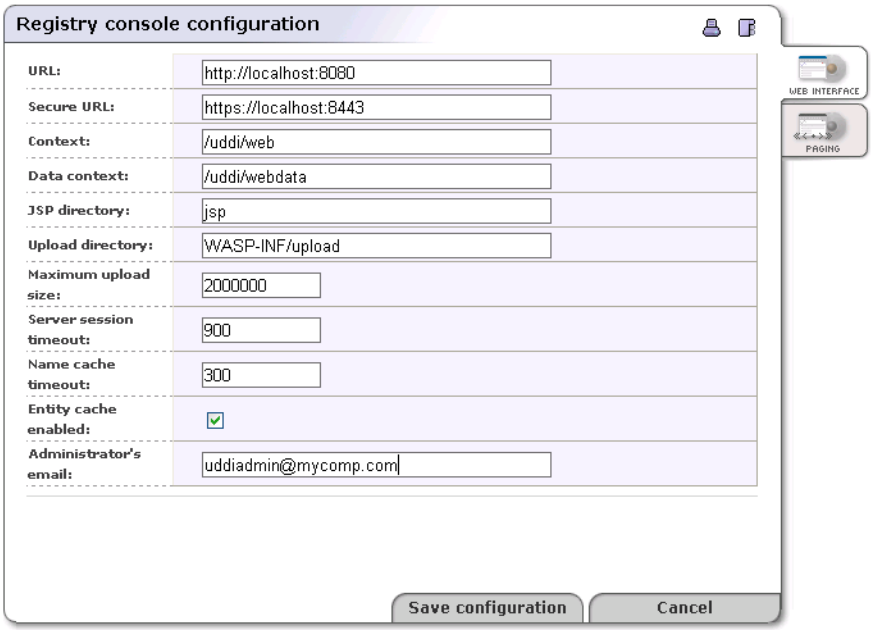
To access the Registry Console configuration:

1. Log on as administrator.
2. Click the **Manage** menu tab.
3. Click **Registry console configuration** link under the **Manage** tab. This returns the configuration screen shown in Figure 46. The Registry Console Configuration screen has two tabs:
   - On the **Web Interface** tab, you can set various parameters associated with BEA AquaLogic Service Registry's interface.
   - On the **Paging** tab, configure the number of rows per page and the maximum number of pages associated with the returns of various searches.

Note that on both tabs there is a button labeled **Reload Configuration**. When you change a registry configuration file directly, and save it, use this button to put the configuration changes into effect.

4.1. Web Interface Configuration

Figure 46. Registry Console Configuration - Web Interface Tab



Field description:

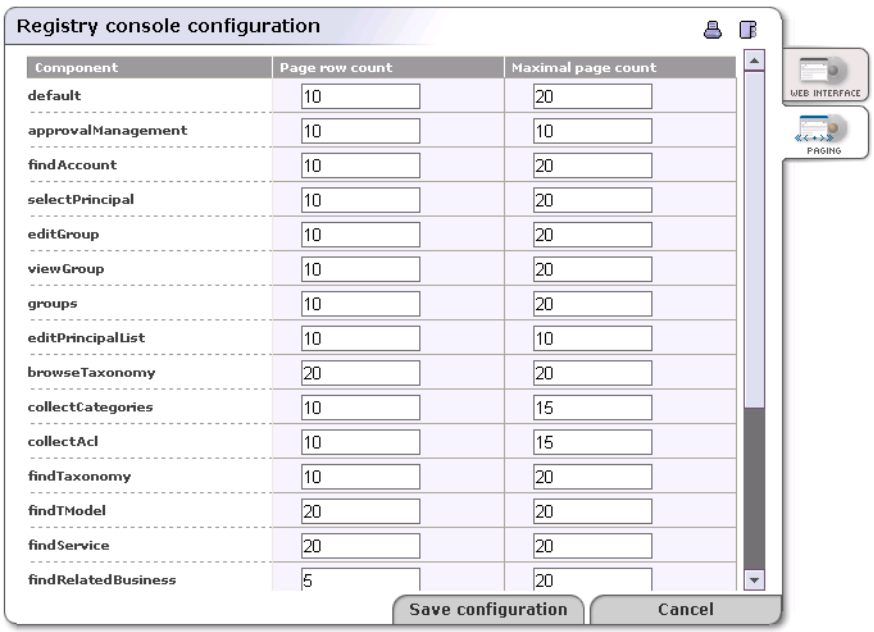
- **URL** - nonsecure registry URL
- **Secure URL** - secure registry URL
- **Context** - context of the Registry Console URL
- **Data context** - context where static objects such as JavaScript and images are stored
- **JSP directory** - location of JSP pages relative to `$REGISTRY_HOME/work/uddi`

- **Upload directory** - upload directory used for tasks such as uploading taxonomies
- **Maximum upload size** - maximum upload size in bytes
- **Server session timeout** - session timeout (measured in seconds)
- **Name cache timeout** - cache timeout for the names of UDDI structures. If someone renames a UDDI structure, the Registry Console will load the new name after this interval has passed (measured in seconds).
- **Entity cache enabled** - If you check this check box, entities will be cached.

Click **Save configuration** when finished.

4.2. Paging Configuration

Figure 47. Registry Console Configuration - Paging Tab



Component	Page row count	Maximal page count
default	10	20
approvalManagement	10	10
findAccount	10	20
selectPrincipal	10	20
editGroup	10	20
viewGroup	10	20
groups	10	20
editPrincipalList	10	10
browseTaxonomy	20	20
collectCategories	10	15
collectAcl	10	15
findTaxonomy	10	20
findTModel	20	20
findService	20	20
findRelatedBusiness	5	20

Paging limits - On this tab, you can specify how many records and on how many pages searched data will appear. Click **Save configuration** when finished.

5. Permissions: Principles

Permissions in BEA AquaLogic Service Registry were developed so that administrators might exercise control over users. Permissions:

- Provide a simple mechanism for the management of users' rights in BEA AquaLogic Service Registry.
- Allow the administrator to manage or make available different parts of the registry to different users.
- Help BEA AquaLogic Service Registry better reflect the real world where there are many roles with different responsibilities.

This chapter describes permissions in detail with some examples and a description of permission configuration.

Permission is defined as the right to perform an action on some interface. Put another way: permission is the ability to process some method on some interface. Permissions are very different from the other mechanism for rights in BEA AquaLogic Service Registry, the Access Control List.

Access Control enables the user to control access to the basic UDDI data structures (businessEntity, businessService, bindingTemplate, and tModel). Access Control on BEA AquaLogic Service Registry is provided by the Access Control List (ACL). The ACL is based on permissions given to a user or group. In the context of ACL, this means that a given user can access only that information in BEA AquaLogic Service Registry made available to the user by the registry administrator or other users. For more information about the Access Control List, see the Access Control chapter in the User's guide.

Access Control Lists limit the visibility of entities and so restrict the access to *data* in BEA AquaLogic Service Registry. Permissions on the other hand restrict access to interfaces. The ACLs restrain users by the restricting the visibility of UDDI structures. Permissions limit users through the visibility of interfaces.

5.1. Permissions Definitions

There are two basic kinds of permission:

- The first, consisting of ApiUserPermission and ApiManagerPermission, is used to restrict access for some users on some interfaces.
- The second, ConfigurationManagerPermission, is used to restrict the ability to change configurations in BEA AquaLogic Service Registry.

ApiUserPermission

ApiUserPermission consists of the interface's name and method from the given interface. This permission provides the user common access to the specified method on the given API. ApiUserPermission enables the user to call methods on an interface as a common user. Users usually must have this permission to perform any call.

ApiManagerPermission

ApiManagerPermission also consists of the names of an interface and of a method. This permission allows the user to call a determined method on the given API. It is very similar to ApiUserPermission. The only difference is in the user's significance. If a user has ApiManagerPermission, that user is considered to be a privileged user. There are many API calls where the result depends on user's importance.

ConfigurationManagerPermission

ConfigurationManagerPermission consists of configuration files and a method's name. The name of the method is either `get` or `set`. The ConfigurationManagerPermission combined with the `get` method allows user to read (get) data from the configuration file. On the other hand, the ConfigurationManagerPermission combined with the `set` method enables the user to write to the configuration.

5.2. BEA AquaLogic Service Registry Permission Rules

The following permissions' rules are always valid:

- Permission is the ability to process a method on an API.
- Permission contains the type of permission (ApiUserPermission, ApiManagerPermission, ConfigurationManagerPermission), the name (interface's or config's name) and an action (method's name).

  You are allowed to use the asterisk wildcard (`*`) to substitute all names - names of interfaces, configurations, or actions.
- There is no hierarchy in permissions. The ability to set permission for users is also a permission (for some methods on PermissionApi).

- The BEA AquaLogic Service Registry administrator has all permissions for all methods on all APIs.
- Permissions are always positive. This means that permissions say what is possible or allowed. Permissions allow user to perform an action (some method on some API). Any action that is not expressly permitted is denied.
- Permissions can be set for an individual user or for a group of members. Each user is member of the group `system#everyone`, therefore every user has the default permissions associated with this group.

For more information, see Section 5.1, Data Access Control: Principles

5.3. Setting Permissions

This section describes the configuration of permissions. The setting of permissions is written from the administrator's point of view.

There are three basic ways to set permissions for a user:

- By performing methods on PermissionApi. A user can call these methods only if that user has the appropriate permissions.
- By calling methods via SOAP or via the Registry Console.
- By changing permissions directly in the configuration file.

The `PermissionApi` contains several methods for managing permissions. These methods are described below:

`get_permission`

Used for obtaining all of a user's permissions. A user possessing the `ApiManagerPermission` can obtain permissions of other users. A user with only `ApiUserPermission`, can only discover his or her own permissions.

Note that users who have neither `ApiUserPermission` nor `ApiManagerPermission` for a method on `PermissionApi`, cannot call this method.

`set_permission`

Provides users the ability to set permissions for other users. It is necessary to possess `ApiManagerPermission` for this call.

`get_permissionDetail`

Similar to `get_permission`, this method can be called for more than one user at a time.

`get_permission` takes a principal as the input parameter. On the other hand, `get_permissionDetail` takes an array of principals as the input parameter. If you want to find out the permissions of three users, you can call `get_permission` three times or you can call `get_permissionDetail` once.

`who_hasPermission`

Enables a user to find out who owns a given permission.

Important

It is not recommended to change permissions directly in the configuration file. However, if the administrator wants to change default permissions for new users (meaning changing permissions for the group `system#everyone`), there is no other possibility. Before making any changes to these permissions, we strongly recommend making a reserve copy of the configuration. The permissions for special users or groups are stored in the file `permission_list.xml`.

5.4. Permissions and User Roles

Many systems use user roles in addition to permissions. A user role is usually a set of permissions; it can be predefined in the system or be user-defined. In BEA AquaLogic Service Registry, the user roles mechanism is implemented by groups. The administrator is allowed to set permissions not only for individual users but also for groups. Instead of restricting the relationship to users and roles, it is possible to create groups, set permissions for them and then add users into these groups. This "group" mechanism in BEA AquaLogic Service Registry is nearly the same as user role mechanism and it is used instead of user roles. For more information, see Section 1.3, Group Management.

5.5. ApiManagerPermission Reference

ApiManagerPermission allow user to use operation in a privileged mode. The following tables explain what does it mean for certain APIs and operations.

Table 2. Account API (org.systinet.uddi.account.AccountApi)

operation (action)	Description
find_userAccount	Not used.
get_userAccount	Allows to get foreign account.
save_userAccount	Allows to save/update any account. Allows to set up non default limits. Allows to skip mail confirmation (if it is required).
delete_userAccount	Allows to delete any account.
enable_userAccount	Not used.

Table 3. Admin Utils API (org.systinet.uddi.admin.AdministrationUtilsApi)

operation (action)	Description
deleteTModel	Allows to call the deleteTModel operation. (ApiUserPermission is not sufficient to call the operation.)
replaceKey	Allows to call the replaceKey operation. (ApiUserPermission is not sufficient to call the operation.)
cleanSubscriptionHistory	Allows to call the cleanSubscriptionHistory operation. (ApiUserPermission is not sufficient to call the operation.)
resetDiscoveryURLs	Allows to call the resetDiscoveryURLs operation. (ApiUserPermission is not sufficient to call the operation.)
transform_keyedReferences	Allows to call the transform_keyedReferences operation. (ApiUserPermission is not sufficient to call the operation.)
rebuild_cache	Allows to call the rebuild_cache operation. (ApiUserPermission is not sufficient to call the operation.)
replaceURL	Allows to call the replaceURL operation. (ApiUserPermission is not sufficient to call the operation.)

Table 4. Category API (org.systinet.uddi.client.category.v3.CategoryApi)

operation (action)	Description
set_category	Allows to call the set_category operation. (ApiUserPermission is not sufficient to call the operation.)
add_category	Allows to call the add_category operation. (ApiUserPermission is not sufficient to call the operation.)
move_category	Allows to call the move_category operation. (ApiUserPermission is not sufficient to call the operation.)
delete_category	Allows to call the delete_category operation. (ApiUserPermission is not sufficient to call the operation.)
find_category	Not used.
get_category	Not used.
get_rootCategory	Not used.
get_rootPath	Not used.

Table 5. Custody API (org.systinet.uddi.client.custody.v3.UDDI_CustodyTransfer_PortType)

operation (action)	Description
get_transferToken	Allows to call the get_transferToken operation on foreign entities.
discard_transferToken	Allows to call the discard_transferToken operation on foreign tokens.

Table 6. Group API (org.systinet.uddi.group.GroupApi)

operation (action)	Description
find_group	Allows to find foreign private groups.
get_group	Allows to get foreign private groups.
save_group	Allows to save/update foreign groups.
delete_group	Allows to delete foreign groups.
where_amI	Not used.
find_user	Not used.
add_user	Not used.
remove_user	Not used.

Table 7. Inquiry V1 API (org.systinet.uddi.client.v1.InquireSoap)

operation (action)	Description
find_binding	Allows to find all bindingTemplates despite ACL rights.
find_business	Allows to find all businessEntities despite ACL rights.
find_services	Allows to find all services despite ACL rights.
find_tModel	Allows to find all tModels despite ACL rights.
get_bindingDetail	Allows to get any bindingTemplate despite ACL rights.
get_businessDetail	Allows to get any businessEntity despite ACL rights.
get_businessDetailExt	Not used.
get_serviceDetail	Allows to get any businessService despite ACL rights.
get_tModelDetail	Allows to get any tModel despite ACL rights.

Table 8. Inquiry V2 API (org.systinet.uddi.client.v2.Inquire)

operation (action)	Description
find_binding	Allows to find all bindingTemplates despite ACL rights.
find_business	Allows to find all businessEntities despite ACL rights.
find_relatedBusinesses	Allows to find all related businessEntities despite ACL rights.
find_services	Allows to find all services despite ACL rights.
find_tModel	Allows to find all tModels despite ACL rights.
get_bindingDetail	Allows to get any bindingTemplate despite ACL rights.
get_businessDetail	Allows to get any businessEntity despite ACL rights.
get_businessDetailExt	Not used.
get_serviceDetail	Allows to get any businessService despite ACL rights.
get_tModelDetail	Allows to get any tModel despite ACL rights.

Table 9. Inquiry V3 API (org.systinet.uddi.client.v3.UDDI_Inquiry_PortType)

operation (action)	Description
find_binding	Allows to find all bindingTemplates despite ACL rights.
find_business	Allows to find all businessEntities despite ACL rights.
find_relatedBusinesses	Allows to find all related businessEntities despite ACL rights.
find_services	Allows to find all services despite ACL rights.
find_tModel	Allows to find all tModels despite ACL rights.
get_bindingDetail	Allows to get any bindingTemplate despite ACL rights.
get_businessDetail	Allows to get any businessEntity despite ACL rights.
get_operationalInfo	Not used.
get_serviceDetail	Allows to get any businessService despite ACL rights.
get_tModelDetail	Allows to get any tModel despite ACL rights.

Table 10. Permission API (org.systinet.uddi.permission.PermissionApi)

operation (action)	Description
get_permission	Allows to call the get_permission operation on foreign accounts and groups.
set_permission	Allows to call the set_permission operation. (ApiUserPermission is not sufficient to call the operation.)
who_hasPermission	Allows to call the who_hasPermission operation. (ApiUserPermission is not sufficient to call the operation.)
find_principal	Allows to call the find_principal operation. (ApiUserPermission is not sufficient to call the operation.)

Table 11. Publishing V1 API (org.systinet.uddi.client.v1.PublishSoap)

operation (action)	Description
delete_binding	Allows deletion of any bindingTemplate despite ACL rights.
delete_business	Allows deletion of any businessEntity despite ACL rights
delete_service	Allows deletion of any businessService despite ACL rights
delete_tModel	Allows deletion of any tModel despite ACL rights
save_binding	* Allows to update any bindingTemplate or create new bindingTemplate in any businessService despite ACL rights. * Skips bindings limit checking.
save_business	* Allows to update any businessEntity despite ACL rights. * Skips businesses limit checking.
save_service	* Allows to update any businessService or create new businessService in any businessEntity despite ACL rights. * Skips services limit checking.
save_tModel	* Allows to update any tModel despite ACL rights. * Skips tModels limit checking.
get_authToken	Not used.
discard_authToken	Not used.
get_registeredInfo	Not used.
validate_categorization	Not used.

Table 12. Publishing V2 API (org.systinet.uddi.client.v2.Publish)

operation (action)	Description
delete_binding	Allows deletion of any bindingTemplate despite ACL rights.
delete_business	Allows deletion of any businessEntity despite ACL rights
delete_service	Allows deletion of any businessService despite ACL rights
delete_tModel	Allows deletion of any tModel despite ACL rights
save_binding	* Allows to update any bindingTemplate or create new bindingTemplate in any businessService despite ACL rights. * Skips bindings limit checking.
save_business	* Allows to update any businessEntity despite ACL rights. * Skips businesses limit checking.
save_service	* Allows to update any businessService or create new businessService in any businessEntity despite ACL rights. * Skips services limit checking.
save_tModel	* Allows to update any tModel despite ACL rights. * Skips tModels limit checking.
add_publisherAssertions	Skips assertions limit checking in add_publisherAssertions operation.
set_publisherAssertions	Skips assertions limit checking in set_publisherAssertions operation.
delete_publisherAssertions	Not used.
get_publisherAssertions	Not used.
get_assertionStatusReport	Not used.
get_authToken	Not used.
discard_authToken	Not used.
get_registeredInfo	Not used.

Table 13. Publishing V3 API (org.systinet.uddi.client.v3.UDDI_Publication_PortType)

operation (action)	Description
delete_binding	Allows deletion of any bindingTemplate despite ACL rights.
delete_business	Allows deletion of any businessEntity despite ACL rights
delete_service	Allows deletion of any businessService despite ACL rights
delete_tModel	Allows deletion of any tModel despite ACL rights
save_binding	* Allows to update any bindingTemplate or create new bindingTemplate in any businessService despite ACL rights. * Skips bindings limit checking.
save_business	* Allows to update any businessEntity despite ACL rights. * Skips businesses limit checking.
save_service	* Allows to update any businessService or create new businessService in any businessEntity despite ACL rights. * Skips services limit checking.
save_tModel	* Allows to update any tModel despite ACL rights. * Skips tModels limit checking.
add_publisherAssertions	Skips assertions limit checking in add_publisherAssertions operation.
set_publisherAssertions	Skips assertions limit checking in set_publisherAssertions operation.
delete_publisherAssertions	Not used.
get_publisherAssertions	Not used.
get_assertionStatusReport	Not used.
get_registeredInfo	Not used.

Table 14. Replication V3 API (org.systinet.uddi.replication.v3.ReplicationApi)

operation (action)	Description
replicate	Allows to call the replicate operation. (ApiUserPermission is not sufficient to call the operation.)

Table 15. Statistics API (org.systinet.uddi.statistics.StatisticsApi)

operation (action)	Description
get_accessStatistics	Allows to call the get_accessStatistics operation. (ApiUserPermission is not sufficient to call the operation.)
reset_accessStatistics	Allows to call the reset_accessStatistics operation. (ApiUserPermission is not sufficient to call the operation.)
get_structureStatistics	Allows to call the get_structureStatistics operation. (ApiUserPermission is not sufficient to call the operation.)

Table 16. Subscription V3 API (org.systinet.uddi.client.subscription.v3.UDDI_Subscription_PortType)

operation (action)	Description
delete_subscription	Allows to delete any subscription despite the caller is not a subscription owner.
save_subscription	* Allows to update any subscription despite the caller is not a subscription owner. * Skips subscription limit checking.
get_subscriptionResults	Allows to get result of any subscription despite the caller is not a subscription owner.
get_subscriptions	Allows to get any subscription despite the caller is not a subscription owner.

Table 17. Taxonomy API (com.systinet.uddi.taxonomy.v3.TaxonomyApi)

operation (action)	Description
get_taxonomy	Allows to obtain all categories in the taxonomy.
find_taxonomy	Not used.
save_taxonomy	Allows to call the save_taxonomy operation. (ApiUserPermission is not sufficient to call the operation.)
delete_taxonomy	Allows to call the delete_taxonomy operation. (ApiUserPermission is not sufficient to call the operation.)
download_taxonomy	Allows to call the download_taxonomy operation. (ApiUserPermission is not sufficient to call the operation.)
upload_taxonomy	Allows to call the upload_taxonomy operation. (ApiUserPermission is not sufficient to call the operation.)

6. Approval Process Principles

In this section, we will focus on approval process from the administrator's point of view. We assume you are familiar with basic principles of approval process described in the User's Guide, Section 1.5, Approval Process in BEA AquaLogic Service Registry.

Approval process includes two types of registries: a *publication registry* and a *discovery registry* . The *publication registry* is used for testing and verification of the accuracy of data. Users publish data to the *publication registry*. The *discovery registry* houses approved data. It has no publishing API, but supports other BEA AquaLogic Service Registry APIs including inquiry, subscriptions, accounts, and so on. (In actual fact, the administrator can publish data to the discovery registry, but this is an exception.)

Note

Both *publication* and *discovery* registries must be running so that user accounts may be synchronized. When the *discovery registry* is down, it is not possible to register a new user account on the *publication registry*.

The accounts on *publication* and *discovery registry* are nearly the same. Accounts created on the *publication registry* and also all their changes are replicated to the *discovery registry*. But accounts can exist on the *discovery registry* that do not exist on *publishing registry*. The *discovery registry* contains right read-only data and can therefore be accessible for more users. It is possible to create accounts with inquiry and subscription privileges on the *discovery registry* that do not exist on the *publishing registry*. Note again that there is no Publish API on the *discovery registry* (except for administrator); the only way to publish data to the *discovery registry* is via the approval process.

Put another way: all accounts on the *publication registry* exist on the *discovery registry*, but not all accounts on *discovery registry* exist on *publication registry*.

When promotion is requested, automatic context checking is performed to ensure the consistency of data. For example, if a business service is contained in the keys for saving in the approval request and its business entity is missing on both the *discovery registry* and in the request, then the request for approval fails. The automatic context checker checks the integrity of data. If an entity is contained in keys for saving, then the parent entity must already exist on the *discovery registry* or be contained in keys for saving to the *discovery registry*. For detailed information, please see User's Guide, Section Context Checking.

6.1. Approval Process Roles

As noted above, the approval process registry has several roles associated with it:

- Section 6.1.1, Requestor
- Section 6.1.2, Approver
- Section 6.1.3, autoApprover
- Section 6.1.4, Administrator

6.1.1. Requestor

The requestor is a user on the *publication registry* who can ask for approval of data for promotion. Every user can ask for approval, but to be a requestor requires an administrator-assigned approval contact.

If a user does not have at least one assigned approval contact, an exception is thrown when this user asks for approval. There is no way for such a user to promote data to the *discovery registry*. By assigning approval contacts, the administrator determines whether to give users the opportunity to publish data to the *discovery registry*.

During the creation of users via the BEA AquaLogic Service Registry console or via API, the default approver, `administrator`, is assigned for all newly created users on the *publication registry*. The default approval contact for all users is `administrator`, though this does not apply to users defined in an external repository (LDAP). Note that demo data does not come with assigned approval contact. For example, the user `demo_john` does not have an assigned approver, thus the administrator must assign this user an approval contact in order for him to make a request.

For more information on the requestor's role, see the section Section 1.5.1, Requestor's Actions.

6.1.2. Approver

The approver is a person or group who approves changes to the *discovery registry*. If the approval contact is group, then all its members are may approve data for promotion.

For detailed information on the approval contact's role, see the User's Guide, Section 1.5.2, Approver's Actions .

6.1.3. autoApprover

A special approval contact exists in the approval process, the autoApprover. This role is defined in the registry at installation. The administrator can set autoApprover as the approval contact for trusted users.

This means that no human approval is required and such users' data is copied to the *discovery registry* upon request for approval, as long as context checking is successful.

6.1.4. Administrator

The administrator is responsible for setting up BEA AquaLogic Service Registry and is therefore also responsible for setting up the approval process. The term administrator refers to the user of BEA AquaLogic Service Registry who can manage the registry. Note that all users who have permission to configure the approval process are allowed to set relationships between requestors and approval contacts.

The manager of the approval configuration assigns approval contact(s) for requestors.

For easy management of relationships between approvers and requestors, it is possible to create an approver or requestor either from an existing user or from a group. If an approver is a group then each user in this group can approve the promotion of data. When several users (requestors) are in the same group, then an approval contact can be assigned to the whole group.

6.2. Optional Content Checking Setup

Optional content checking provides an approver the ability to programmatically check data for approval. For example, the approver can set a policy that:

- Each business service must include a binding template, or
- Each business service must be categorized by some specific categories

To enforce such a policy, a developer can write an implementation of the CheckerApi to ensure these checks. The implementation is called automatically during the approval process when an approver presses the **Approve request** button. Therefore, the approver does not have to check it manually.

To set up optional content checking:

1. Write a class that implements the `org.systinet.uddi.approval.checker.v3.CheckerApi`
2. There are two ways to make the implementation class available:
   - Copy the `.jar` file including the implementation class to the `REGISTRY_HOME/app/uddi/services/Wasp-inf/lib`, or
   - Implement a Web service that can perform the `checkRequest()` method from CheckerApi interface and deploy the service to the Systinet Server for Java
3. Register the implementation of the content checker class in the BEA AquaLogic Service Registry data:
   a. Publish the WSDL of the checker service.

      Publish the WSDL located at `http://<host_name>:<http_port>/uddi/doc/wsdl/approval_checker.wsdl` to a new or already existing business entity. You should reuse the existing WSDL portType (tModel's name: CheckerApi, tModel's key: `uddi:systinet.com:uddi:service:porttype:approvalchecker`).

   b. Specify the checker in the access point of a new binding template.
      - If you have put your implementation of the CheckerApi into the registry classpath, then the value of access point must start with the `class:` prefix and continue with the fully qualified class name. For example `class:com.systinet.uddi.approval.v3.approver.CheckerApiImpl`.
      - If you have deployed your checker as a Web service to the Systinet Server for Java, then the access point is the endpoint URL of the service. For example `http://localhost:6060/ContentChecker`.

See Developer's Guide, Section 3.6, Writing a Content Checker to see the implementation example.

7. PStore Tool

The PStoreTool provides BEA AquaLogic Service Registry Protected Store management. It provides functionality to:

- Import and export trusted certificates locally to or from a file.
- Create new security identities in the BEA AquaLogic Service Registry configuration file.
- Copy identities between protected stores.

Note

Remote protected store management via SOAP is not supported with BEA AquaLogic Service Registry.

The general usage is:
`PStoreTool` [command [options]]

You can perform operations from the command line or start up a GUI interface.

7.1. Commands Description

The PStore tool has the following commands:

- **new** - Creates a new security identity in the local protected store. The configuration file of the protected store can be specified using the *-config* parameter.
- **newServer** - Creates a new security identity on BEA AquaLogic Service Registry. The location of the server is specified with the *-url* parameter.
- **copy** - Copies the existing security identity from one protected source to another or to the BEA AquaLogic Service Registry protected store.
- **add** - Adds a trusted X.509 certificate to the local protected store. The X.509 certificate can be supplied as a local file.

  This command can also add mapping between the security identity alias and the X.509 certificate to the user store part of the protected store. (The certificate is needed only for the server-side protected store.) This can be requested by using *-user* with the *-alias* option.
- **addServer** - Adds a trusted certificate to BEA AquaLogic Service Registry. This command also adds the mapping between the security identity alias and its X.509 certificate to the user store part of the BEA AquaLogic Service Registry protected store. The certificate can be given in the local file or can be fetched from the local protected store. The configuration file can be specified using the *-config* option.
- **remove** - Removes the given alias from the local protected store. This command can also remove an alias from the user store part of the protected store using the *-user* option. When removing a mapping from the user store, the X.509 certificates mapped to the given alias are also removed from the key store.
- **removeServer** - Removes a given alias from the protected store. The alias is removed from the user store part of the protected store if it is not found in the key store. When removing mapping from the user store part, the X.509 certificates mapped to the given alias are also removed from the key store.
- **lsTrusted** - Displays a list of the trusted certificate's Subject-distinguished names from the local protected store.

- **lsTrustedServer** - Displays a list of the trusted certificate's Subject distinguished names from the server.
- **list** - Displays all aliases contained in the key store part of the local protected store.
- **listServer** - Displays all aliases contained in the key store part of the BEA AquaLogic Service Registry protected store.
- **export** - Exports the X.509 certificate chain stored in the key store or in the user store of the local protected store with the given alias.
- **exportServer** - Exports the X.509 certificate chain stored in the key store or in the user store of the protected store with the given alias.
- **gui** - Launches the graphical version of this tool.

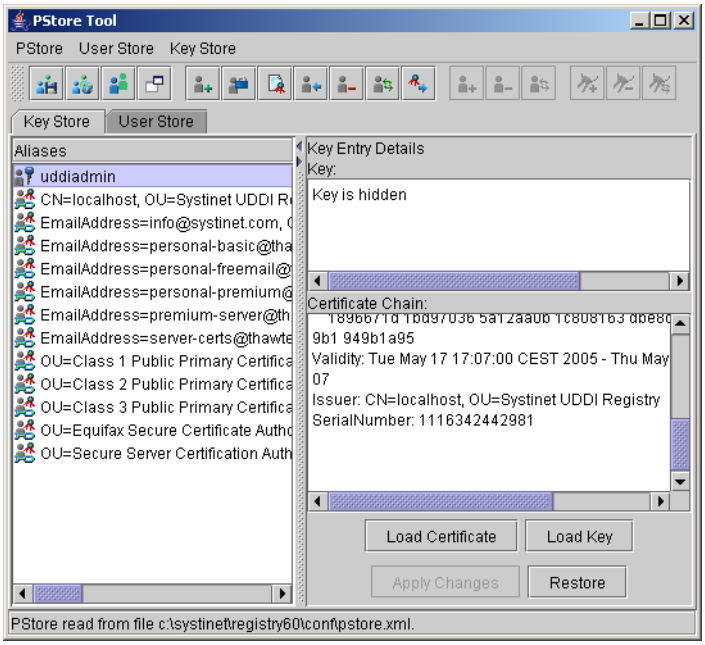
The PStore tool has the following options:

- **-alias alias** - Alias to be used for the command.
- **-keyPassword password** - Password for encrypting/decrypting the security identity private key.
- **-subject subjectDN** - Subject-distinguished name to be used in the generated X.509 certificate.
- **-config configPath** - File and path to the configuration file to be used during command execution for the source of the local protected store.
- **-username username** - Username for authentication process. Not required if the BEA AquaLogic Service Registry server is unsecured.
- **-password password** - Password for authentication process. Not required if the server is unsecured.
- **-secprovider provider** - Authentication mechanism used during the authentication process. Not required if the server is unsecured.
- **-certFile certPath** - File and path to the X.509 certificate stored in a local file.
- **-user** - Indicates that a command should be executed only with the contents of the user store of the protected store.
- **-config2 secondConfigPath** - Path to the second configuration file. Used for the copy command, when copying an identity from one local protected store to another.

7.2. PStore Tool - GUI Version

You can add, edit, or remove any user properties in the user store. You can also add, edit, and remove certificates and identities in the key store. You can do all of this with a local file containing the protected store.

Figure 48. PStore Tool



7.2.1. Running the GUI PStore Tool

To run the graphical version of this tool, use *gui* as parameter with the `PStoreTool` command.
`PStoreTool` gui

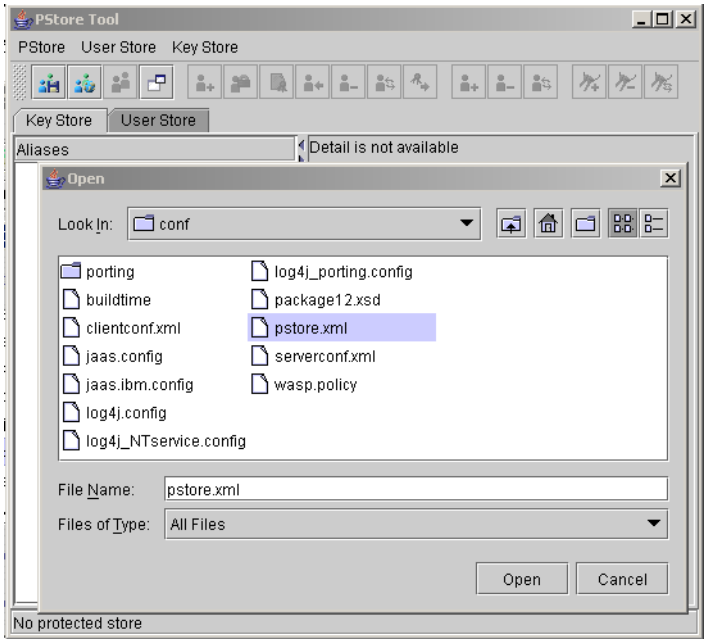
7.2.2. Opening and Closing the Protected Store

Opening Protected Store from a File

The GUI PStore Tool can manipulate every protected store in a file. To manipulate the client's protected store, open `clientconf.xml`. To open the server protected store, open `pstore.xml`.

To open protected store from file, select **Open From File...** from the PStore menu. This returns the file chooser dialog. Select the file you want to open as shown in Figure 49.

Figure 49. Open Protected Store from a File



Closing Protected Store

To close the protected store, select **Close** from the **PStore** menu.

7.2.3. Open Next Protected Store

In some cases you need to work with more than one protected store at the same time. Typically you want to copy certificates from one protected store to another. To open another protected store, select the **New Window** from the **PStore** menu. New windows appear. Now you can open the protected store from a file.

7.2.4. Copy Data Between Protected Stores

With the PStore Tool, you can manipulate more than one protected store at the same time. You can simply copy identities, certificates, users, and user properties from one protected store to another using the Copy and Paste actions located in context menus of the Aliases, Users, and Properties panels.

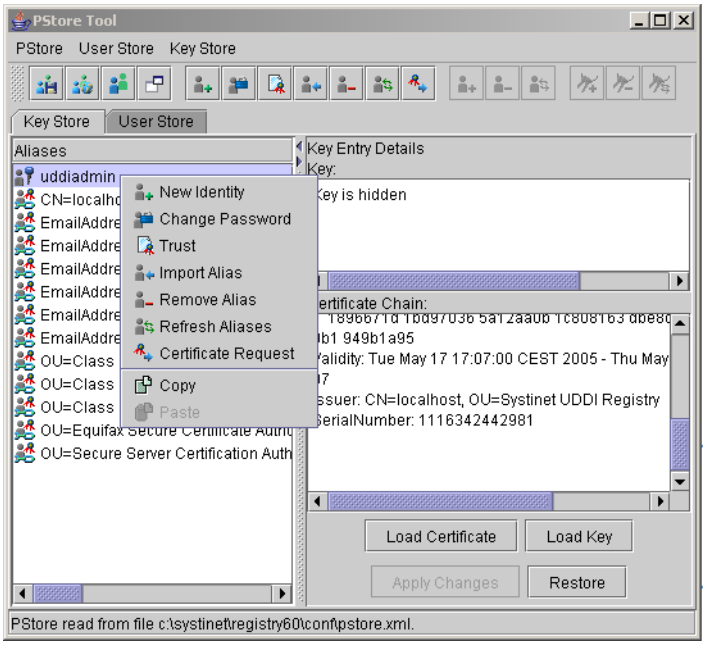
Note

When you copy data from one area to another, the Paste action is disabled for some categories of data. This means that data may be copied, but cannot be pasted to the selected area. For example, the `password` property from the user store cannot be pasted to the key store.

7.2.5. Key Store

To work with the key store, select the **Key Store** tab. This tab has two panels. The left side has a list of all entries. The right has detailed information for the selected entry.

Figure 50. Key Store Tab



Create New Identity

To create a new identity, select **New Identity...** from the **Key Store** menu. This opens a dialog for information such as Alias, Distinguished Name, and Password. (The Distinguished Name is not mandatory.) If the specified information is valid, the new identity will be added to the key store with the specified Alias. Otherwise an error dialog will be returned.

Key Store Trust

If you want to trust a key entry, select **Trust** from the **Key Store** menu. This action is available only for the key entry type.

Import Alias

To import a certificate from a file into the key store, select **Import Alias** from the **Key Store** menu. This opens a dialog in which you can specify Alias, Type, and value that depend on the entry type. In the current implementation, you can import only the certificate chain entry type.

Remove Alias

To remove an alias from the key store, select the alias you want to remove and select **Remove Alias** from the **Key Store** menu. You can remove several aliases at once.

Refresh Aliases

To synchronize information shown in this tool with the original key store source, perform a refresh by selecting **Refresh Aliases** from the **Key Store** menu.

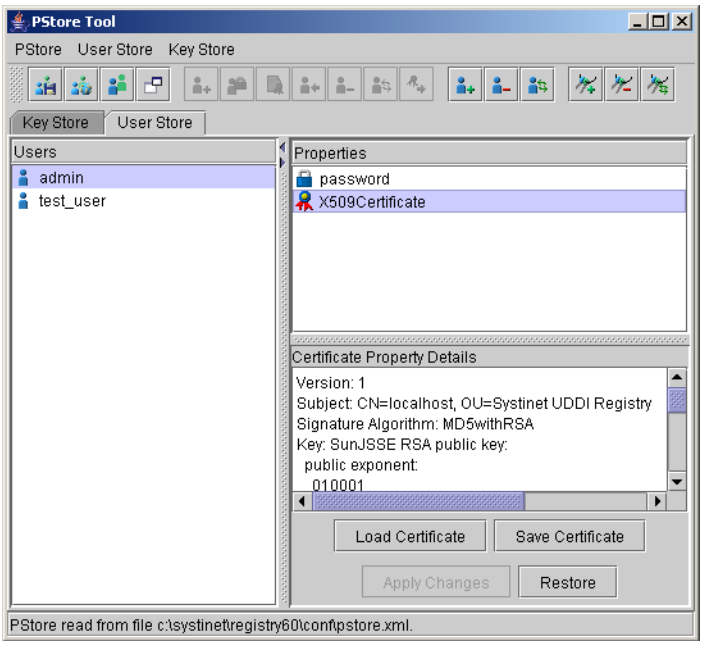
Alias Details Panel

It is not surprising that the **Details** panel has more details about the selected alias. This panel shows details that depend on the entry type. You can also change this value. If you want to store a new value, press the **Apply Changes** button. To return to the original value, press **Restore**.

7.2.6. User Store

There are three panels on the User Store tab. The left side has a list of all entries. On the right top are properties available for the selected user. On the right bottom is detailed information for the selected user property.

Figure 51. User Store Tab



Add User

To add a new user, select **Add User** from the **User Store** menu. This opens a dialog for entering the Username. Press **OK** when done.

Remove User

To remove a user from the user store, select the user you want to remove and choose **Remove User** from the **User Store** menu. You can remove several users at once.

Refresh Users

Refresh synchronizes information shown in this tool with the original user store source. To refresh, select **Refresh Users** from the **User Store** menu.

Add Property

To add a new user property, select **Properties** and **Add Property** from the **User Store** menu. This returns a dialog for the property you want to create and its value.

Remove Property

To remove one or more user properties from the user store, select them and select **Properties** and **Remove Property** from the **User Store** menu.

Refresh Properties

To synchronize information on the Properties panel with the original user store source, perform a refresh. Select **Properties** and **Refresh Properties** from the **User Store** menu.

User Properties Details Panel

The **Details** panel has more information about user properties that depend on the property type. Select the property you want to see. You can also change this value. If you want to store a new value press **Apply Changes**.

To return to the original value, press **Restore**.

Developer's Guide

The Developer's Guide is divided into the following main parts:

- Mapping of Resources covers registering various XML resources in BEA AquaLogic Service Registry including WSDL definitions, schemas, and transformations.
- Client-Side Development describes the basic principles of using BEA AquaLogic Service Registry APIs. For each client API, there is a comprehensive description of data structures and operations including links to JavaDoc, XML Schemas and WSDL documents.
- Server-Side Development discusses how to access server-side APIs, including custom modules, interceptors, external validation services, and subscription notification services. The BEA AquaLogic Service Registry web framework is also described in this section.
- UDDI From Developer Tools discusses how to access UDDI from Systinet Developer for Eclipse and Microsoft Visual Studio .NET.
- How to debug describes logging and using the SOAPSpy tool.

1. Mapping of Resources

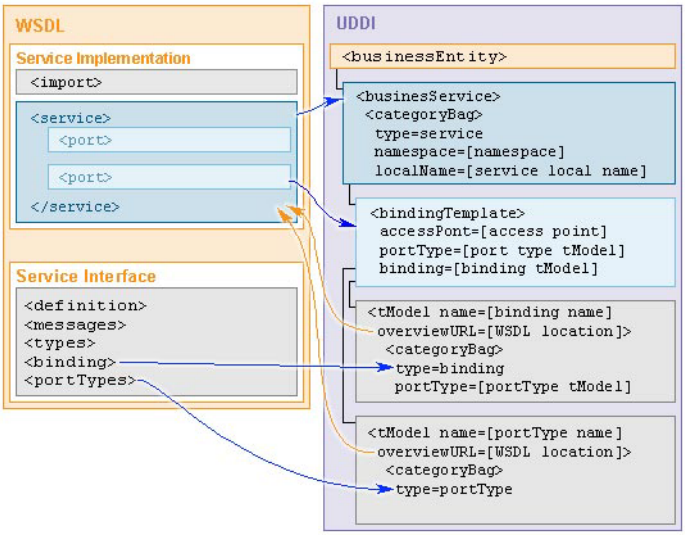
BEA AquaLogic Service Registry provides you with functionality to register the following resources:

- WSDL definition
- XML file
- XML Schema (XSD)
- XSL Transformation

1.1. WSDL

This describes how to publish a WSDL file to BEA AquaLogic Service Registry. The implementation reflects the OASIS UDDI technical note Using WSDL in a UDDI Registry, Version 2.0 [http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631.htm]. As shown in Figure 1, the technical note suggests a mapping between WSDL and UDDI.

Figure 1. WSDL TO UDDI



1.1.1. WSDL PortTypes

As shown in Table 1, “WSDL portType:UDDI Mapping”, each WSDL portType maps to a tModel having the the same name as the local name of the portType in the WSDL specification. The overviewURL of the tModel becomes the URL of the WSDL specification. The tModel contains a categoryBag with a keyedReference for the type of WSDL artifact and the namespace of the WSDL definitions element containing the portType, as follows:

- The type is categorized as portType.
- The namespace is categorized as the WSDL binding namespace.

Table 1. WSDL portType:UDDI Mapping

WSDL	UDDI
portType	tModel (categorized as portType)
Namespace of portType	keyedReference in categoryBag
Local name of portType	tModel name
WSDL location	overviewURL

1.1.2. WSDL Bindings

In similar fashion, as summarized in Table 2, “wsdl binding:UDDI mapping”, WSDL bindings are mapped to tModels created for each binding, with name of the tModel gathered from the WSDL binding local name and the overviewURL again being the URL of the WSDL specification. Again, the tModel contains a categoryBag, this time with the following keyedReferences:

- The type is categorized as binding.
- The namespace is categorized as the WSDL binding namespace.
- A portType category on the binding is used to refer to the portType tModel that was created for the WSDL portType (as described above).

- The protocol and transport categories are set to the same attributes as described in the WSDL binding, such as SOAP and HTTP, respectively.

Table 2. wsdl binding:UDDI mapping

WSDL	UDDI
Binding	tModel (categorized as binding and wsdlSpec)
Namespace of binding	keyedReference in categoryBag
Local name of binding	tModel name
WSDL location	overviewURL
portType binding	keyedReference in categoryBag
Protocol	keyedReference in categoryBag
Transport	keyedReference in categoryBag

1.1.3. WSDL Service

WSDL services are represented as UDDI businessServices. The name is a human readable name. The tModel again contains a categoryBag which this time contains the following keyedReferences:

- The type is categorized as service
- The namespace is again categorized as the WSDL binding namespace.
- The local name is categorized as the local name of the service.

The businessService also contains a bindingTemplate:

- The access type is categorized as the access point of the service.
- The portType is categorized as the tModel of the portType.
- The binding is categorized as the tModel of the binding information.
- The local name is categorized as the local name of the port.

Table 3. wsdl service:UDDI mapping

WSDL	UDDI
Service	businessService (categorized as service)
Namespace of service	keyedReference in categoryBag
Local name of service	keyedReference in categoryBag; optionally used name of service

1.1.4. Use Cases

BEA AquaLogic Service Registry supports the following use cases:

- **Publishing a WSDL file** You can also specify how artifacts of the WSDL file will be mapped to the existing UDDI structures.
- **Search for a WSDL** You can search for the WSDL file by WSDL location (URI).

- **Unpublish and republish the WSDL** You can unpublish and republish the WSDL

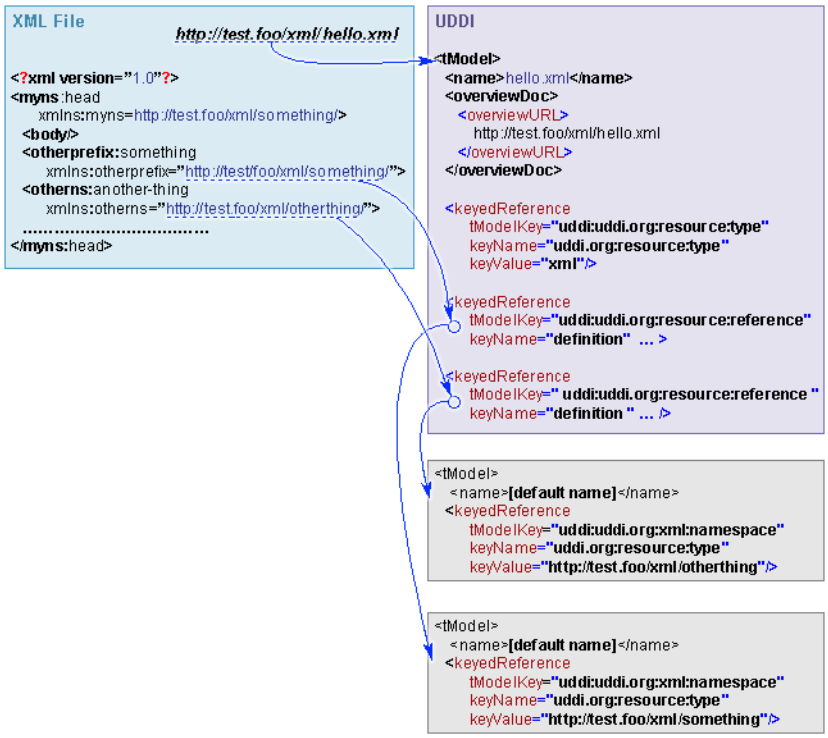
For more information, also see:

- User's Guide, Section Publishing WSDL Documents
- User's Guide, Section Find WSDL
- Developer's Guide, Section 2.2.8, WSDL Publishing

1.2. XML

As shown in Figure 2, an XML file is mapped to a tModel. The location of the XML file is added to the tModel's overviewURL element. Namespaces are mapped to keyedReferences in the tModel categoryBag. Each namespace is mapped to a tModel.

Figure 2. XML TO UDDI



1.2.1. Use Cases

BEA AquaLogic Service Registry supports the following use cases:

- **Publish an XML document** You can also specify how artifacts of the XML file will be mapped to the existing UDDI structures.
- **Search for an XML file**
  - Search for an XML file containing data of certain type (XSD) .

- Search for an XML file from a specified server or folder, using search criteria, URI prefix, and wild card characters.
- Search for an XML file that is input or output of a specified XSLT.
- Search for a generator of a specified output XML file.
- Search for a processor of a specified input XML file.

- Unpublish and republish the XML file.

For more information, also see:

- User's Guide, Section Publish XML
- User's Guide, Section Find XML
- Developer's Guide, Section 2.2.9, XML Publishing

1.3. XSD

As shown in Figure 3, an XML Schema file is mapped to a tModel. The location URI of the XSD file is put to the tModels overviewURL element and the target namespace is mapped to a keyedReference in the tModel category bag. xsd:types, xsd:elements and xsd:imports are mapped to the tModel keyedReferences. For each type, element or import, a new tModel is created.

Figure 3. XSD to UDDI

1.3.1. Use Cases

BEA AquaLogic Service Registry supports the following use cases:

- **Publish an XML Schema** You can also specify how artifacts of the XML Schema file will be mapped to existing UDDI structures
- **Search for an XML schema:**
  - Search for an XML Schema that imports artifacts declared in the specified XSD file.
  - Search for an XML Schema located in a specified server or folder.
  - Search for all XSL transformations that can process documents using a specified XSD.
  - Search for all XSL transformations producing documents that use the specified XSD.
- **Unpublish and republish the XML Schema** You can unpublish and republish the XML Schema

For more information, also see:

- User's Guide, Section Find XSD
- User's Guide, Section Publish XSD
- Developer's Guide, Section 2.2.10, XSD Publishing

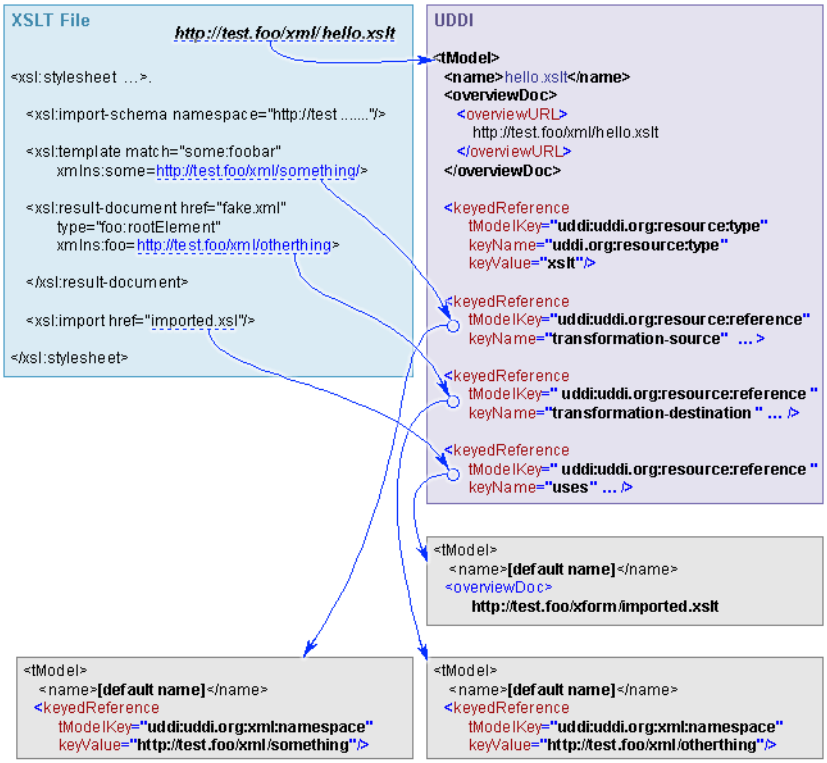
1.4. XSLT

As shown in Figure 4 an XSL Transformation is mapped to a tModel:

- The location URI of the XSLT file is added to the tModel's overviewURL element.
- Namespaces are mapped to keyedReferences in the tModel's categoryBag.
- The xsl:import elements are also mapped to keyedReferences in the tModel's categoryBag.

For each import and namespace, a new tModel is created.

Figure 4. XSLT TO UDDI



1.4.1. Use Cases

BEA AquaLogic Service Registry supports the following use cases:

- **Publish an XSL Transformation** You can also specify how artifacts of the XSLT file will be mapped to the existing UDDI structures.
- **Search for an XSL Transformation**
  - Search for inputs and outputs of the specified XSLT.
  - Search for compatible XSDs.
- **Unpublish and republish the XSL transformations** You can unpublish and republish the XSL transformations

For more information, also see:

- User's Guide, Section Find XSLT
- User's Guide, Section Publish XSLT
- Developer's Guide, Section 2.2.11, XSLT Publishing

2. Client-Side Development

Client-Side Development includes the following sections:

- UDDI APIs - Describes the principles of how to use BEA AquaLogic Service Registry APIs. The UDDI API set can be split by typical use case into two parts. The **Inquiry API** set is used to locate and obtain details on entries in the UDDI registry. For example to find out endpoint of given web service. The **publication API** set is used to publish and update information in the UDDI registry.
- Advanced APIs - Advanced APIs cover the following APIs: Validation API, Taxonomy API, Category APIs, Approval API, Administration Utilities API, Replication API, Statistics API, Inquiry UI API, Subscription Ext Api, and Publishing API for resources:
  - WSDL Publishing
  - XML Publishing
  - XSD Publishing
  - XSLT Publishing
- Security APIs - Security APIs cover the following APIs: Account API, Group API, Permission API.
- Registry Client - This section describes how to prepare your own client distribution. A client created this way allows you to access the BEA AquaLogic Service Registry API through a SOAP interface.
- Client authentication - describes how to create a client that autheticates thru HTTP Basic.

2.1. UDDI APIs

UDDI (Universal Description Discovery and Integration) is set of Web service that supports the description and discovery of Web service providers, Web services and technical fingerprints of those Web service.

The UDDI API set can be split by typical use case into two parts. The Inquiry API set is used to locate and obtain details on entries in the UDDI registry. For example to find out endpoint of given web service. The publication API set is used to publish and update information in the UDDI registry.

2.1.1. Principles To Use UDDI API

This section will show you how to use the BEA AquaLogic Service Registry API. Examples are based on UDDI version 3 Specification [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm].

To use Inquiry APIs you can follow these steps. The complete code fragment is shown in Example 1, FindBinding v3.

1. Get API implementation from stub

```
String url = "http://localhost:8080/uddi/inquiry";
UDDI_Inquiry_PortType inquiry = UDDIInquiryStub.getInstance(url);
```

2. Collect inquiry parameters

```
String serviceKey = "uddi:systinet.com:demo:hr:employeesList";
String tModelKey = "uddi:systinet.com:demo:employeeList:binding";
Find_binding find_binding = new Find_binding();
find_binding.setServiceKey(serviceKey);
```

```
find_binding.addTModelKey(tModelKey);
find_binding.setMaxRows(new Integer(10));
```

3. Call inquiry method

```
BindingDetail bindingDetail = inquiry.find_binding(find_binding);
```

4. Operate with inquiry result

```
ListDescription listDescription = bindingDetail.getListDescription();
if (listDescription != null) {
    int includeCount = listDescription.getIncludeCount();
    int actualCount = listDescription.getActualCount();
    int listHead = listDescription.getListHead();
    System.out.println("Displaying " + includeCount + " of " +
        actualCount+ ", starting at position " + listHead);
}
```

Note

If you get the `java.lang.reflect.UndeclaredThrowableException` exception, check whether BEA AquaLogic Service Registry is running.

To use the publishing API, follow these steps. The complete code fragment is shown in Example 2, SaveService v3.

1. Get API of security stub

```
String securityUrl = "http://localhost:8080/uddi/security";
UDDI_Security_PortType security = UDDISecurityStub.getInstance(securityUrl);
String publishingUrl = "http://localhost:8080/uddi/publishing";
UDDI_Publication_PortType publishing = UDDIPublishStub.getInstance(publishingUrl);
```

2. Get authentication token

```
AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
String authInfo = authToken.getAuthInfo();
```

3. Create save object

```
String businessKey = "uddi:systinet.com:demo:it";
String serviceKey = ""; // serviceKey is optional
int count = 1;
String[] serviceNames = new String[count];
String[] languageCodes = new String[count];
languageCodes[0] = null; // can set an array of language codes
serviceNames[0] = "Requests Service"; //service name
String serviceDescription = "Saved by Example"; //service description
BusinessService businessService = new BusinessService();
businessService.setBusinessKey(businessKey);
if (serviceKey != null && serviceKey.length() > 0)
    businessService.setServiceKey(serviceKey);
businessService.addName(new Name(serviceNames[0], languageCodes[0]));
```

```
businessService.addDescription(new Description(serviceDescription));
Save_service save = new Save_service();
save.addBusinessService(businessService);
save.setAuthInfo(authInfo);
```

4. Call publishing method

```
ServiceDetail serviceDetail = publishing.save_service(save);
```

5. Operate with publishing result

```
BusinessServiceArrayList
     businessServiceArrayList = serviceDetail.getBusinessServiceArrayList();
int position = 1;
for (Iterator iterator = businessServiceArrayList.iterator();
  iterator.hasNext();) {
    BusinessService service = (BusinessService) iterator.next();
    System.out.println("Service " + position + " : " + service.getServiceKey());
    System.out.println(service.toXML());
    position++;
}
```

6. Discard the authentication token

```
security.discard_authToken(new Discard_authToken(authInfo));
```

Example 1. FindBinding v3

```
// Copyright 2001-2005 Systinet Corp. All rights reserved.
// Use is subject to license terms.

package example.inquiry;

import org.systinet.uddi.client.v3.UDDIInquiryStub;
import org.systinet.uddi.client.v3.UDDI_Inquiry_PortType;
import org.systinet.uddi.client.v3.struct.*;

import java.util.Iterator;

public class PrincipleFindBinding {

    public static void main(String args[]) throws Exception {

        //1. Get API implementation from stub
        String url = "http://localhost:8080/uddi/inquiry";
        System.out.print("Using Inquiry at url " + url + " ..");
        UDDI_Inquiry_PortType inquiry = UDDIInquiryStub.getInstance(url);
        System.out.println(" done");

        //2. Collect inquiry parameters
        String serviceKey = "uddi:systinet.com:demo:hr:employeesList";
        String tModelKey = "uddi:systinet.com:demo:employeeList:binding";
        Find_binding find_binding = new Find_binding();
        find_binding.setServiceKey(serviceKey);
        find_binding.addTModelKey(tModelKey);
        find_binding.setMaxRows(new Integer(10));

        //3. Call inquiry method
        System.out.print("Search in progress ..");
        BindingDetail bindingDetail = inquiry.find_binding(find_binding);
        System.out.println(" done");

        //4. Operate with result
        ListDescription listDescription = bindingDetail.getListDescription();
        if (listDescription != null) {
            int includeCount = listDescription.getIncludeCount();
            int actualCount = listDescription.getActualCount();
            int listHead = listDescription.getListHead();
            System.out.println("Displaying " + includeCount + " of " + actualCount
               + ", starting at position " + listHead);
        }

        BindingTemplateArrayList bindingTemplateArrayList
           = bindingDetail.getBindingTemplateArrayList();
        if (bindingTemplateArrayList == null) {
            System.out.println("Nothing found");
            return;
        }

        int position = 1;
```

```
        for (Iterator iterator = bindingTemplateArrayList.iterator();
          iterator.hasNext();) {
            BindingTemplate bindingTemplate = (BindingTemplate) iterator.next();
            System.out.println("Binding " + position + " : " +
                  bindingTemplate.getBindingKey());
            System.out.println(bindingTemplate.toXML());
            position++;
        }
    }
}
```

Example 2. SaveService v3

```
// Copyright 2001-2005 Systinet Corp. All rights reserved.
// Use is subject to license terms.

package example.publishing;

import org.systinet.uddi.InvalidParameterException;
import org.systinet.uddi.client.v3.UDDIException;
import org.systinet.uddi.client.v3.UDDIPublishStub;
import org.systinet.uddi.client.v3.UDDISecurityStub;
import org.systinet.uddi.client.v3.UDDI_Publication_PortType;
import org.systinet.uddi.client.v3.UDDI_Security_PortType;
import org.systinet.uddi.client.v3.struct.AuthToken;
import org.systinet.uddi.client.v3.struct.BusinessService;
import org.systinet.uddi.client.v3.struct.BusinessServiceArrayList;
import org.systinet.uddi.client.v3.struct.Description;
import org.systinet.uddi.client.v3.struct.Discard_authToken;
import org.systinet.uddi.client.v3.struct.DispositionReport;
import org.systinet.uddi.client.v3.struct.Get_authToken;
import org.systinet.uddi.client.v3.struct.Name;
import org.systinet.uddi.client.v3.struct.Save_service;
import org.systinet.uddi.client.v3.struct.ServiceDetail;

import javax.xml.soap.SOAPException;
import java.util.Iterator;

public class PrincipleSaveService {

    public static void main(String[] args) throws UDDIException,
             InvalidParameterException, SOAPException {

        String userName = "demo_john";
        String password = "demo_john";

        //1. Get API implementation from stub
        String securityUrl = "http://localhost:8080/uddi/security";
        System.out.print("Using Security at url " + securityUrl + " ..");
        UDDI_Security_PortType security = UDDISecurityStub.getInstance(securityUrl);
        System.out.println(" done");
        String publishingUrl = "http://localhost:8080/uddi/publishing";
        System.out.print("Using Publishing at url " + publishingUrl + " ..");
        UDDI_Publication_PortType publishing = UDDIPublishStub.getInstance(publishingUrl);
        System.out.println(" done");

        //2. Get authentication token
        System.out.print("Logging in ..");
        AuthToken authToken =
            security.get_authToken(new Get_authToken(userName, password));
        System.out.println(" done");
        String authInfo = authToken.getAuthInfo();

        //3. Create save object
        String businessKey = "uddi:systinet.com:demo:it";
```

```
        String serviceKey = ""; // serviceKey is optional
        int count = 1;
        String[] serviceNames = new String[count];
        String[] languageCodes = new String[count];
        languageCodes[0] = null; // can set an array of language codes
        serviceNames[0] = "Requests Service"; //service name
        String serviceDescription = "Saved by Example"; //service description
        BusinessService businessService = new BusinessService();
        businessService.setBusinessKey(businessKey);
        if (serviceKey != null && serviceKey.length() > 0)
            businessService.setServiceKey(serviceKey);
        businessService.addName(new Name(serviceNames[0], languageCodes[0]));
        businessService.addDescription(new Description(serviceDescription));

        Save_service save = new Save_service();
        save.addBusinessService(businessService);
        save.setAuthInfo(authInfo);

        //4. Call publishing method
        System.out.print("Save in progress ...");
        ServiceDetail serviceDetail = publishing.save_service(save);
        System.out.println(" done");

        //5. Operate with publishing result
        BusinessServiceArrayList businessServiceArrayList =
             serviceDetail.getBusinessServiceArrayList();
        int position = 1;
        for (Iterator iterator = businessServiceArrayList.iterator();
          iterator.hasNext();) {
            BusinessService service = (BusinessService) iterator.next();
            System.out.println("Service " + position + " : "
               + service.getServiceKey());
            System.out.println(service.toXML());
            position++;
        }
        //6. Discard authentication token
        System.out.print("Logging out ..");
        security.discard_authToken(new Discard_authToken(authInfo));
        System.out.println(" done");
    }
}
```

2.1.2. UDDI Version 1

The UDDI version 1 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1] has provided a foundation for next versions.

Inquire

- WSDL: inquire_v1.wsdl [${doc.web.root}/wsdl/inquire_v1.wsdl]
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API:

- Demos: Inquiry demos v1

Publish

- WSDL: publish_v1.wsdl [${doc.web.root}/wsdl/publish_v1.wsdl]
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API:
- Demos: Publishing demos v1

2.1.3. UDDI Version 2

The UDDI version 2 Specification [http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm] has introduced many improvements of existing concepts and new features like service projections.

Inquiry

- Specification: Inquiry API functions [http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm#_Toc25137711]
- WSDL: inquire_v2.wsdl [${doc.web.root}/wsdl/inquire_v2.wsdl]
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API:
- Demos: Inquiry demos v2

Publish

- Specification: Publishing API Function [http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm#_Toc25137730]
- WSDL: publish_v2.wsdl [${doc.web.root}/wsdl/publish_v2.wsdl]
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API:
- Demos: Publishing demos v2

2.1.4. UDDI Version 3

The UDDI version 3 Specification [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm] is a major step in providing industry standard for building and querying XML web services registries useful in both public and private deployments.

Inquiry

- Specification: Inquiry API set [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047277]
- API endpoint: `http://<host name>:<port>/uddi/inquiry`
- Java API:
- Demos: Inquiry demos v3

Publication

- Specification: Publication API set [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047296]
- API endpoint: `http://<host name>:<port>/uddi/publishing`
- Java API:
- Demos: Publishing demos v3

Security

- Specification: Security API set [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047316]
- API endpoint: `http://<host name>:<port>/uddi/security`
- Java API:

Custody

The Custody and Ownership Transfer API is used to transfer UDDI structures between UDDI nodes and to change their ownership. One use case is when the publisher wishes to transfer responsibility for a selected UDDI structure to another user, typically after a business reorganization.

- Specification: Custody and Ownership Transfer API Set [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047319]
- API endpoint: `http://<host name>:<port>/uddi/custody`
- Java API:
- Demos: Custody Demos

Subscription

The Subscription API is a service that asynchronously sends notification to users who have registered an interest in changes to a registry. These users have a range of options in specifying matching criteria so that they receive only the information in which they are interested.

- Specification: Subscription API Set [http://uddi.org/pubs/uddi-v3.00-published-20020719.htm#_Toc42047327]
- API endpoint: `http://<host name>:<port>/uddi/custody`
- Java API:
- Demos: Subscription Demos

2.1.5. UDDI Version 3 Extension

UDDI Version 3 Extensions are Systinet extensions of the UDDI Version 3 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]. The following data structures are used by APIs for the Registry Console and APIs that will be approved as official technical notes of the UDDI specification.

Data Structures

businessEntityExt

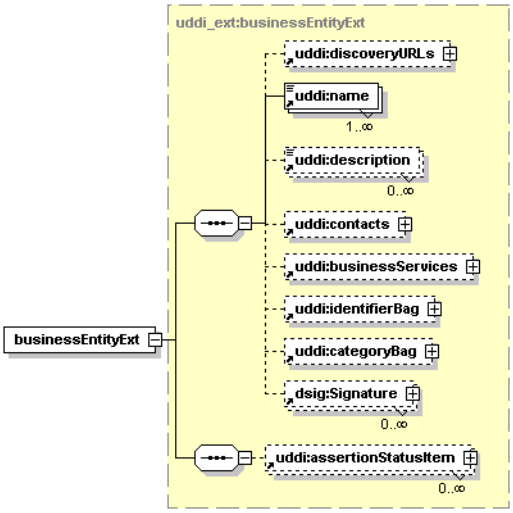


Table 4. Attributes

Name	Required
`businessKey`	Optional

This structure is used by the Registry Console for performance enhancements. The structure is an extension of businessEntity [http://uddi.org/pubs/uddi-v3.0.1-20031014.htm#_Toc53709226], the added element is uddi:assertionStatusItem [http://uddi.org/pubs/uddi-v3.0.1-20031014.htm#_Toc53709302] that points to the related businessEntity,

businessInfoExt

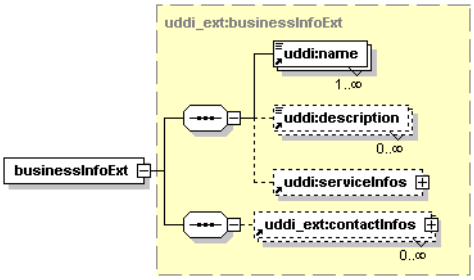


Table 5. Attributes

Name	Required
`businessKey`	Optional

This structure is an extension of the `businessInfo` structure; the added element is `uddi_ext:contactInfos`.

contactInfo

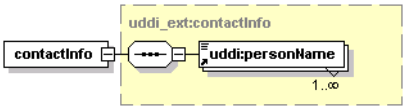


Table 6. Attributes

Name	Required
`useType`	Optional

This structure represents a person name for the businessInfoExt.

contactInfos

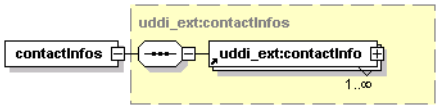


Table 7. Attributes

Name	Required
`useType`	Optional

This structure holds a list of contactInfos.

operationalInfoExt

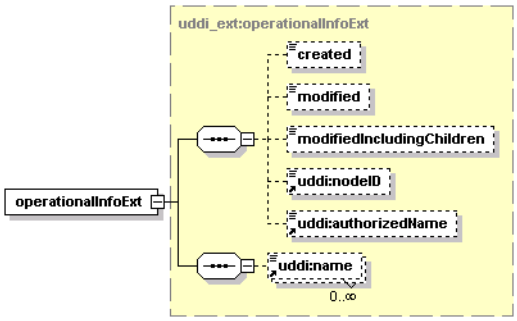


Table 8. Attributes

Name	Required
`entityKey`	Required
`entityKeyV2`	Optional

This structure is an extension of the `operationalInfo` [http://uddi.org/pubs/uddi-v3.0.1-20031014.htm#_Toc53709242] structure, the added element is `uddi:name`. The `entityKeyV2` holds UDDI v2 key values.

qualifiedKeyedReference

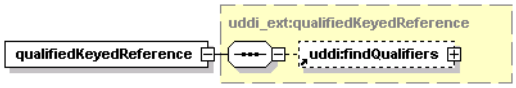


Table 9. Attributes

Name	Required
`tModelKey`	Required
`keyName`	Optional
`keyValue`	Required

This structure holds findQualifiers that are used in Range Queries.

registeredInfoExt

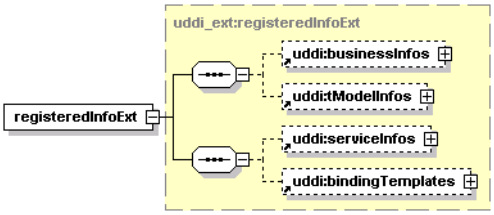


Table 10. Attributes

Name	Required
`truncated`	Optional

This structure is used by ACL functionality. The added elements are `uddi:serviceInfos` and `uddi:bindingTemplates` that point to UDDI entities the user does not own but has privileges to modify.

serviceInfoExt

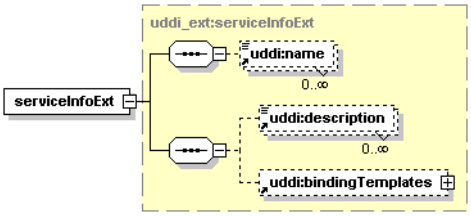


Table 11. Attributes

Name	Required
`serviceKey`	Required
`businessKey`	Required

This structure is an extension of `serviceInfo`. It is used by the web interface for performance enhancements. The added elements are `uddi:description` and `uddi:bindingTemplates`.

Find Qualifiers

UDDI V3 Specification [http://uddi.org/pubs/uddi-v3.0.1-20031014.htm#_Toc53709434] permits vendors to define new find qualifiers. Table 12, “Summary of Additional Find Qualifiers in BEA AquaLogic Service Registry ” summarizes the additional find qualifiers in BEA AquaLogic Service Registry and the find_xx operations that support them. See Section Inquiry for more information on inquiry API operations.

Each short name in Table 12, “Summary of Additional Find Qualifiers in BEA AquaLogic Service Registry ” links to a subsection that follows. Note that the tModel key is the short name prefixed with `uddi:systinet.com:findQualifier:`.

Table 12. Summary of Additional Find Qualifiers in BEA AquaLogic Service Registry

Short Name	Supporting Operations				
	find_business	find_service	find_binding	find_tModel	find_relatedBusinesses
deletedTModels				✓	
foreignEntities	✓	✓	✓	✓	
keyNameMatch	✓	✓	✓	✓	✓
myEntities	✓	✓	✓	✓	
omitKeyNameMatch	✓	✓	✓	✓	✓
omitKeyValueMatch	✓	✓	✓	✓	✓
omitTModelKeyMatch	✓	✓	✓	✓	✓
tModelKeyApproximateMatch	✓	✓	✓	✓	✓

deletedTModels

This find qualifier returns only hidden tModels, hence enabling administrators to locate and permanently delete garbage tModels.

Note that the registry settings determine whether delete_tModel:

- just hides the tModel from find_tModel operations (default behaviour required by the specification);
- really deletes the tModel, provided there are no dependencies on it;

See Administrator's Guide, Section 2.7, Node.

tModel Key	`uddi:systinet.com:findQualifier:deletedTModels`
Supporting Operations	find_tModel.

foreignEntities

This find qualifier restricts results to entities that do not belong to the caller.

Note

This qualifier does not make any sense for an anonymous caller because all entities will be returned in the query.

tModel Key	`uddi:systinet.com:findQualifier:foreignEntities`
Supporting Operations	All find_xx operations except find_relatedBusinesses.

keyNameMatch

This find qualifier changes default rules for matching keyedReferences. By default keyNames are only compared when the General Keywords tModelKey is specified. This find qualifier enforces comparison of keyNames.

The keyNameMatch and omitKeyNameMatch findQualifiers are mutually exclusive.

tModel Key	`uddi:systinet.com:findQualifier:keyNameMatch`
Supporting Operations	All find_xx operations.

myEntities

This find qualifier restricts results to entities that belong to the caller.

Note

This qualifier does not make any sense for an anonymous caller. All entities would be returned in that case.

tModel Key	`uddi:systinet.com:findQualifier:myEntities`
Supporting Operations	All find_xx operations except find_relatedBusinesses.

omitKeyNameMatch

This find qualifier changes default rules for matching keyedReferences. By default keyNames are only compared when the General Keywords tModelKey is specified. This find qualifier skips comparison of keyNames.

The keyNameMatch and omitKeyNameMatch findQualifiers are mutually exclusive.

tModel Key	`uddi:systinet.com:findQualifier:omitKeyNameMatch`
Supporting Operations	All find_xx operations.

omitKeyValueMatch

This find qualifier changes default rules for matching keyedReferences. By default keyValues are compared. This find qualifier skips comparison of keyValues.

The omitKeyValueMatch and omitTModelKeyMatch findQualifiers are mutually exclusive.

tModel Key	`uddi:systinet.com:findQualifier:omitKeyValueMatch`
Supporting Operations	All find_xx operations.

omitTModelKeyMatch

This find qualifier changes default rules for matching keyedReferences. By default tModelKeys are compared. This find qualifier skips comparison of tModelKeys.

The omitKeyValueMatch and omitTModelKeyMatch findQualifiers are mutually exclusive.

tModel Key	`uddi:systinet.com:findQualifier:omitTModelKeyMatch`
Supporting Operations	All find_xx operations.

tModelKeyApproximateMatch

This find qualifier changes the default rules for matching keyedReferences. By default tModelKeys are compared without wildcards and case insensitively. This find qualifier enables a tModelKey in a query to include wildcards:

- '%' interpreted as zero or more arbitrary characters;
- '_' interpreted as an arbitrary character.

The behavior is similar to the approximateMatch find qualifier.

tModel Key	`uddi:systinet.com:findQualifier:tModelKeyApproximateMatch`
Supporting Operations	All find_xx operations.

2.2. Advanced APIs

Advanced APIs cover the following APIs:

- Validation API - The Valueset Validation API is used to validate values in keyedReferences involved in save operations that reference checked taxonomies. Valueset validation is defined in the UDDI version 3 specification [http://uddi.org/pubs/uddi_v3.htm]. Every checked taxonomy requires a Web service that implements this API.
- Taxonomy API - The Systinet Taxonomy API provides a high-level view of taxonomies and makes them easy to manage and query. This API was designed according to the UDDI technical note Providing A Value Set For Use In UDDI Version 3 [http://oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-valuesetprovider-20030212.htm].
- Category APIs - The Systinet Category API complements the Systinet Taxonomy API. It is used to query and to manipulate Internal taxonomies in BEA AquaLogic Service Registry. More information on the subject of internal taxonomies can be found in the API documentation. The categories may be hierarchically organized. Each category may be top-level (without parent), it may have children, or it may be a child of another category. You can drill down through this pattern In the Registry Console.
- Approval API - The Approval API includes a set of APIs to manage the approval process.
- Administration Utilities API - The Systinet Administration Utilities API provides an interface to perform several low level administrative tasks in BEA AquaLogic Service Registry.
- Replication API - The Replication API is used to launch replications in BEA AquaLogic Service Registry.
- Statistics API - The Systinet Statistics API provides useful information about BEA AquaLogic Service Registry usage.
- WSDL Publishing API - BEA AquaLogic Service Registry WSDL-to-UDDI mapping is compliant with OASIS's Technical Note, Using WSDL in a UDDI registry Version 2.0 [http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm]. It enables the automatic publishing of WSDL documents to UDDI, enables precise and flexible UDDI queries based on specific WSDL artifacts and metadata, and provides a consistent mapping for UDDI v2 and UDDI v3.
- Resources Publishing APIs - XML2UDDI, XSD2UDDI and XSLT2UDDI. These API sets allow you to manipulate with resources in BEA AquaLogic Service Registry. XML documents, XML Schemas and XSL Transformations are supported.
- Inquiry UI API - The Inquiry UI API has been implemented for improving the performance of the Business Service Console. The basic idea is to retrieve data that appear in the Business Service Console using a single API call.
- Subscription Ext API - The Subscription Extension API has been implemented to allow the user to create subscriptions in the *discovery registry* of the approval process.

2.2.1. Validation

The Valueset validation API is used to validate values in keyedReferences involved in save operations that reference checked taxonomies. Valueset validation is defined in the UDDI version 3 specification [http://uddi.org/pubs/uddi_v3.htm]. Every checked taxonomy requires a Web service that implements this API. The API is defined by the `uddi:uddi.org:v3_valueSetValidation` tModel for UDDI version 3, `uddi:systinet.com:v2_validateValues` for UDDI version 2 and `uddi:systinet.com:v1_validateValues` for UDDI version 1.

BEA AquaLogic Service Registry is built according to the UDDI technical note Providing A Value Set For Use In UDDI Version 3 [http://oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-valuesetprovider-20030212.htm]. To function correctly, checked taxonomies must be categorized with `uddi-org:validatedBy` taxonomy pointing to the bindingTemplate with the valueset validation Web service accessPoint. This Web service is called whenever the checked taxonomy occurs within a keyedReference during a save operation.

If the Web service is accessible by BEA AquaLogic Service Registry's classloader, the validation Web service does not need to be invoked over SOAP, but it may run inside the registry's Java Virtual Machine.

The accessPoint value must be in a special form: It must start with the *class:* prefix and continue with fully qualified class name. For example, the internal validation service endpoint is defined as follows: `class:com.systinet.uddi.publishing.v3.validation.service.AclValidator`.

For more information, consult the UDDI version 3 specification, section 5.6 [http://uddi.org/pubs/uddi_v3.htm#_Toc53709335] .

SOAP

- Specification: uddi_vs_v3.wsdl [${doc.web.root}/wsdl/uddi_vs_v3.wsdl]

Java

- Java API:
- Demos: Validation demos

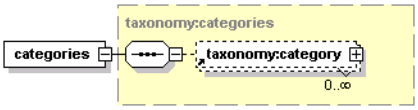
2.2.2. Taxonomy

The Systinet Taxonomy API provides high-level view of taxonomies and makes them easy to manage and query. This API was built according to the UDDI technical note Providing A Value Set For Use In UDDI Version 3 [http://oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-valuesetprovider-20030212.htm].

Data Structures

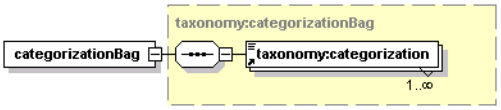
The following structures are used by the Systinet Taxonomy API:

Categories



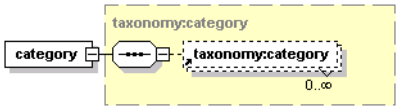
This structure is a container for zero or more `category` structures. If the taxonomy is internal, then categories are used to hold possible values of its keyedReferences.

categorizationBag



This structure is a container for one or more `categorizations`. It defines the containers (categoryBag, keyedReferenceGroup, identifierBag and Publisher Assertion) in which this taxonomy can be used. Possible values are categorization, categorizationGroup, identifier, and relationship. A save operation containing a keyedReference referencing a taxonomy in the wrong container will be denied with E_valueNotAllowed UDDI exception.

Category

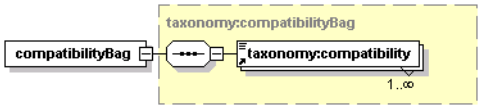


This structure corresponds to the keyedReference. It defines the keyedReference of the taxonomy in which it is used. The `keyValue` must be unique. The `disabled` attribute is used to mark the category as either helper or deprecated, so it cannot be used as a valid option in keyedReferences. The `keyName` attribute serves as a label for this category.

Table 13. Attributes

Name	Required
`keyName`	Yes
`keyValue`	Yes
`disabled`	No

compatibilityBag



This structure is a container for one or more compatibilities. It defines the compatibility of the taxonomy with the four basic UDDI data structures - tModel, businessEntity, businessService and bindingTemplate. If the taxonomy is not compatible with one of these UDDI structures, then a save operation containing a keyedReference referencing this taxonomy in this structure will be denied with E_valueNotAllowed UDDI exception.

taxonomy

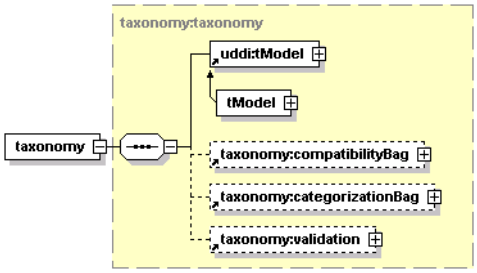


Table 14. Attributes

Name	Required
`check`	No
`unvalidatable`	No
`brief`	No

Each taxonomy is identified by its tModel.

- The optional `check` attribute is used to define whether the taxonomy is checked or not. If the tModel is checked, then a validation structure must be present.

- The `unvalidatable` attribute is used to mark the checked taxonomy as unvalidatable. Unvalidatable taxonomies cannot be used in keyedReferences.
- The `brief` attribute is related to categories structure and its meaning depends on context, in which it is used.

taxonomyDetail

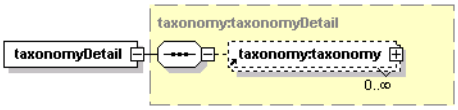


Table 15. Attributes

Name	Required
`truncated`	No

This structure is a container for zero or more `taxonomies`. The `truncated` attribute indicates whether the list of taxonomies is truncated.

taxonomyInfo

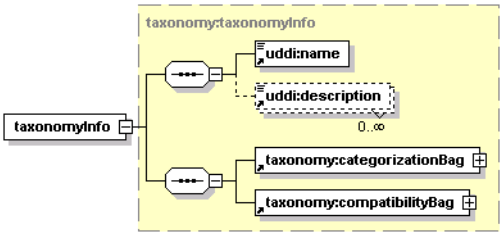


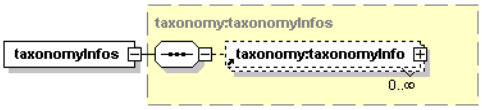
Table 16. Attributes

Name	Required
`check`	Yes
`unvalidatable`	No

The taxonomyInfo is an extension of the tModelInfo structure.

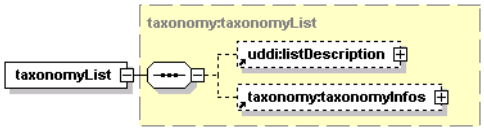
- The `check` attribute indicates whether or not the taxonomy is checked.
- The `unvalidatable` attribute is used to mark the checked taxonomy as unvalidatable. Unvalidatable taxonomies cannot be used in keyedReferences.

taxonomyInfos



This structure is a container for zero or more `taxonomyInfo` structures.

taxonomyList

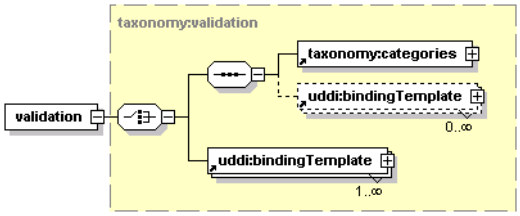


This structure serves as a container for optional `listDescription` and optional `taxonomyInfos` structures. The `truncated` attribute indicates whether the list of taxonomies is truncated.

Table 17. Attributes

Name	Required
`truncated`	No

validation

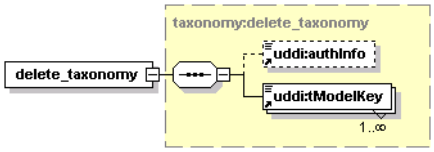


This structure is used to hold information for validating a checked taxonomy. The `categories` structure defines the list of available values for keyedReferences checked by the Internal validation service. `Binding templates` contains the valueset validation Web service endpoint.

Operations

delete_taxonomy

The delete_taxonomy API call is used to delete one or more taxonomies from BEA AquaLogic Service Registry. The taxonomy consists of a tModel and optional business services and categories.



Arguments

- `uddi:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:tModelKey` - One or more required uddiKey values that represent existing taxonomy tModels.

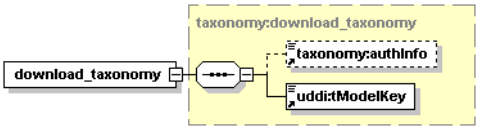
Upon successful completion, a disposition report is returned with a single success indicator.

Permissions

This API call requires API manager permission with the name `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action delete_taxonomy.

download_taxonomy

The download_taxonomy API call is used to fetch a selected taxonomy from BEA AquaLogic Service Registry. This call is stream oriented and is useful for fetching the content of very large taxonomies.



Arguments

- `taxonomy:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:tModelKey` - required uddiKey value that represents an existing taxonomy tModel.

Returns

This API call returns a ResponseMessageAttachment with the selected taxonomy upon success.

Permissions

This API call requires the API manager permission with name `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action download_taxonomy.

find_taxonomy

The find_taxonomy API call is used to find all taxonomies in a registry that match given criteria. This call is an extension of the UDDI v3 find_tModel API call.

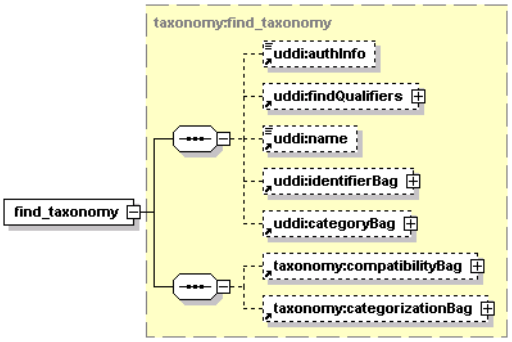


Table 18. Attributes

Name	Required
`check`	No
`unvalidatable`	No

Arguments

- `uddi:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:findQualifiers` - The collection of findQualifier used to alter default behavior.
- `uddi:name` - The string value represents the name of tModel to be found.

- `uddi:identifierBag` - The list of keyedReferences from tModel IdentifierBag.
- `uddi:categoryBag` - The list of keyedReferences from tModel categoryBag.
- `taxonomy:compatibilityBag` - An optional list of Compatibilities.
- `taxonomy:categorizationBag` - An optional list of Categorizations.
- `check` - Optional boolean value that limits returned data to checked (or unchecked) taxonomies only.
- `unvalidatable` - Optional boolean value that limits returned data to unvalidatable taxonomies only.

Important

The `unvalidatable` attribute of the tModel of a checked taxonomy will be set to true, if one of the following rules is met:

- The tModel of a checked taxonomy does not contain the validatedBy keyedReference
- The bindingTemplate from keyedReferences does not exists or is not readable because of ACLs.

Returns

This API call returns the TaxonomyList upon success.

Permissions

This API call requires API user permission `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action find_taxonomy.

get_taxonomy

The get_taxonomy API call returns the Taxonomy structure corresponding to each of the tModelKey values specified.

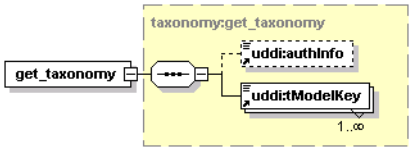


Table 19. Attributes

Name	Required
`brief`	No

Arguments

- `uddi:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:tModelKey` - Required uddiKey value representing an existing taxonomy tModel.
- `brief` - Requests not to fetch the `categories` element. Note that only the API manager can set this attribute to false.

Returns

This API call returns the TaxonomyList on success.

Important

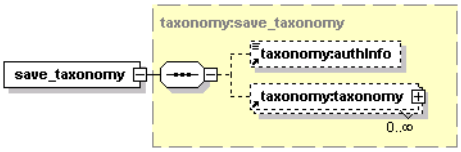
If the tModel of a checked taxonomy does not contain the validatedBy keyedReference, the taxonomy's `unvalidatable` attribute will be set to true and the validation structure will be missing.

Permissions

This API call requires the API user permission `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action get_taxonomy.

save_taxonomy

The save_taxonomy API call is used to publish taxonomies to BEA AquaLogic Service Registry.



The taxonomy properties (checked, unvalidatable, compatibilityBag, and categorizationBag) are first combined with their counterparts in the tModel's categoryBag.

Note

It is an error to specify a validation structure for an unchecked taxonomy. If the taxonomy contains a validation structure, it is automatically set to be checked. If the taxonomy is neither checked nor unchecked, it will be saved as unchecked. If a checked taxonomy does not have a validation structure, the taxonomy is saved with the `unvalidatable` attribute set to true.

If the categories structure is defined in the validation structure, then the taxonomy will be checked by the Internal validation service. The bindingTemplates are optional; if they are specified, then their AccessPoint must point to the Internal validation service's Web service endpoint.

If the categories structure is not defined in the validation structure, then there must be at least one bindingTemplate. The bindingTemplate must implement valueset validation API (either `uddi:uddi.org:v3_valueSetValidation`, `uddi:systinet.com:v2_validateValues` or `uddi:systinet.com:v1_validateValues`). There must be a valid AccessPoint.

If the serviceKey is given, then this businessService must be part of the Operational business entity (`uddi:systinet.com:uddinodebusinessKey`). During the save_taxonomy operation, the businessService will be overwritten.

Arguments

- `taxonomy:authInfo` - This optional argument is an element that contains an authentication token.
- `taxonomy:taxonomy` - A list of taxonomies to be saved.

Returns

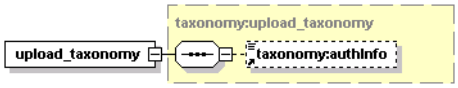
This API call returns the TaxonomyDetail on success.

Permissions

This API call requires the API manager permission `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action save_taxonomy.

upload_taxonomy

The upload_taxonomy API call is used to publish a Taxonomy into BEA AquaLogic Service Registry. This call is stream oriented and is useful for publishing very large taxonomies.



Permissions

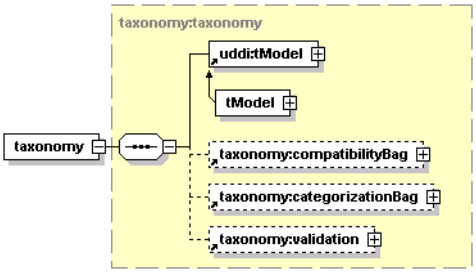
This API call requires the API manager permission named `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and the action upload_taxonomy.

Persistence Format

The taxonomy persistence format is used by taxonomy Download/Upload operations. Following is an example of the taxonomy persistence format:

```
<taxonomy xmlns="http://systinet.com/uddi/taxonomy/v3/5.0"
    xmlns:uddi="urn:uddi-org:api_v3"
         check="true">
    <tModel tModelKey="uddi:foo.com:demo:myTaxonomy">
        <uddi:name>My taxonomy</uddi:name>
        <uddi:description>Category system</uddi:description>
    </tModel>
    <compatibilityBag>
        <compatibility>businessEntity</compatibility>
    </compatibilityBag>
    <categorizationBag>
        <categorization>categorization</categorization>
    </categorizationBag>
    <validation>
        <bindingTemplate bindingKey="" serviceKey="" xmlns="urn:uddi-org:api_v3">
            <accessPoint useType="endPoint">
               http://www.foo.com/MyValidationService.wsdl
            </accessPoint>
            <tModelInstanceDetails>
                <tModelInstanceInfo
                    tModelKey="uddi:uddi.org:v3_valueSetValidation"/>
                <tModelInstanceInfo
                    tModelKey="uddi:systinet.com:demo:myTaxonomy"/>
            </tModelInstanceDetails>
        </bindingTemplate>
    </validation>
</taxonomy>
```

This format reflects the taxonomy.xsd [${doc.web.root}/wsdl/taxonomy.xsd] XML Schema Definition file. For more information, see the data structure of Section taxonomy.



WSDL

You can find the WSDL specification in the file taxonomy.wsdl [${doc.web.root}/wsdl/taxonomy.wsdl].

API Endpoint

You can find the Taxonomy API endpoint at `http://<host name>:<port>/uddi/taxonomy`.

Java

Systinet Java API is generated from Taxonomy WSDL. You are encouraged to browse `org.systinet.uddi.client.taxonomy.v3.TaxonomyApi` and to read and try Taxonomy demos.

Taxonomy 5.5 Extension

This section describes the taxonomy 5.5. extension intended for Range queries functionality implementation.

Data Structures

The following structures are used by the Systinet Taxonomy 5.5 API:

Taxonomy

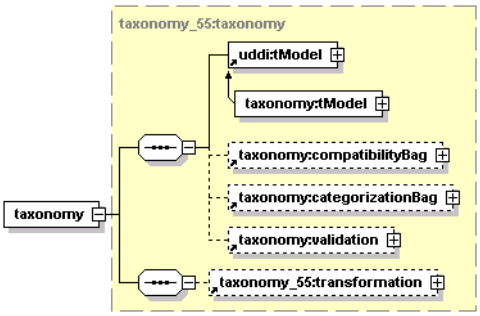


Table 20. Attributes

Name	Required
`check`	No
`unvalidatable`	No
`brief`	No

This structure is almost identical to taxonomy, except that the transformation argument has been added

taxonomyInfo

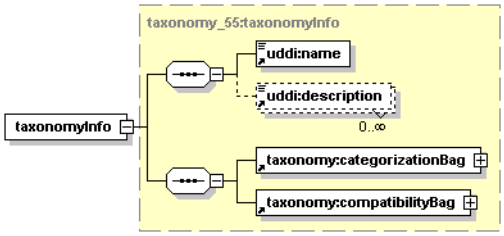
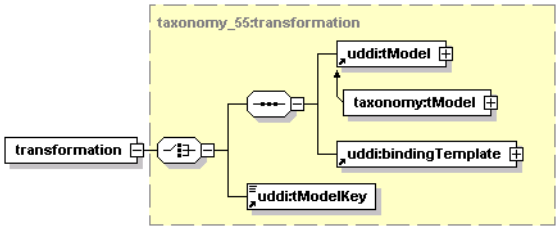


Table 21. Attributes

Name	Required
`check`	Yes
`tModelKey`	yes
`unvalidatable`	No
`isOrderedBy`	No

This structure is almost identical to taxonomyInfo, except that the optional attribute `isOrderedBy` was added to contain the name of the comparator tModel.

transformation



This structure holds a reference to a transformation service implementation. For more information about the transformation service, please see Administrator's Guide, Section Custom Ordinal Types.

- `uddi:tModel` - The tModel that represents a comparator taxonomy.
- `uddi:bindingTemplate` - This argument holds the reference of the transformation service implementation. The `accessPoint` element of the bindingTemplate includes the name of the java class implementation of the sevice with the prefix `class:`.
- `uddi:tModelKey` The key of the tModel that represents the transformation.

API Endpoint

You can find the Taxonomy 5.5 API endpoint at `http://<host name>:<port>/uddi/taxonomy55`.

2.2.3. Category

The Systinet Category API complements the Systinet Taxonomy API. It is used to query and to manipulate Internal taxonomies in BEA AquaLogic Service Registry. The categories may be hierarchically organized. Each category may be top-level (without parent), it may have children, or it may be a child of another category. You can drill down through this pattern in the Registry Console.

Data Structures

The following structures are used by the Systinet Category API:

Categories

This structure is a container for zero or more `category` elements.

category

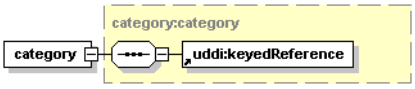


Table 22. Attributes

Attribute	Required
`disabled`	No
`leaf`	No

This element contains a single `keyedReference` element that defines value of the category.

The `disabled` attribute is used to indicate that a category cannot be used as a valid option in keyedReferences. Either it has been deprecated or it is only a parent for other categories. The tModel key value in the uddi-org:types taxonomy is one such disabled category.

The `leaf` attribute indicates whether this category is a leaf in the category tree.

categoryList

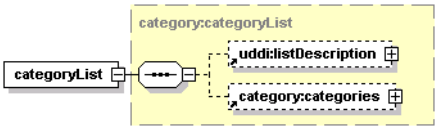


Table 23. Attributes

Attribute	Required
`truncated`	No

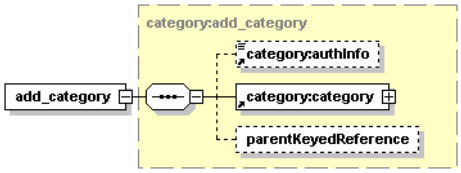
This structure serves as a container for optional `listDescription` and `categories` structures. The `truncated` attribute indicates whether a returned list of categories is truncated.

Operations

add_category

The `add_category` API call is used to add a new category to the Internal taxonomy identified by the tModelKey in the keyedReference. The `parentKeyedReference` element is used to define the parent category of new category to be saved. If the `parentKeyedReference` element is missing, then the new category will have no parent.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `category:category` - Category to be added.
- `parentKeyedReference` - Optional keyedReference; serves as parent of the new category.

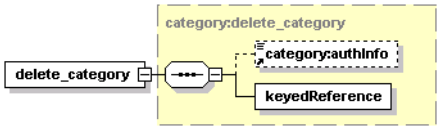
Permissions

This API call requires API manager permission for `org.systinet.uddi.client.category.v3.CategoryApi` and for the action `add_category`.

delete_category

The `delete_category` API call deletes the selected category from BEA AquaLogic Service Registry.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `keyedReference` - Category to be deleted.

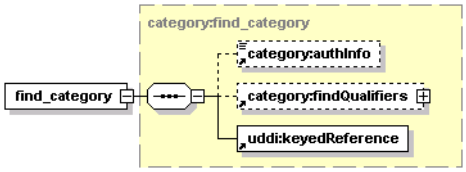
Permissions

This API call requires API manager permission for `org.systinet.uddi.client.category.v3.CategoryApi` and the action `delete_category`.

find_category

The `find_category` API call is used to query BEA AquaLogic Service Registry for categories that match given criteria.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `category:findQualifiers` - Optional list of findQualifiers, that modifies default behavior.
- `uddi:keyedReference` - The category containing search arguments.

Behavior

FindByName and findByValue findQualifiers are used to distinguish whether the call will search by keyName or keyValue from the keyedReference that is the argument of the call. The default is to search by value.

The caseSensitiveMatch and caseInsensitiveMatch findQualifiers are used to control whether the search will be case sensitive; the default is case sensitive.

The ApproximateMatch findQualifier is used to search with SQL wildcards. The default findQualifier, exactMatch, instructs the search to perform an exact comparison.

Finally there are four findQualifiers that affect the order in which categories are returned:

- sortByNameAsc
- sortByNameDesc
- sortByValueAsc (default)
- sortByValueDesc

These find qualifiers are exclusive. If you combine them, an exception is thrown.

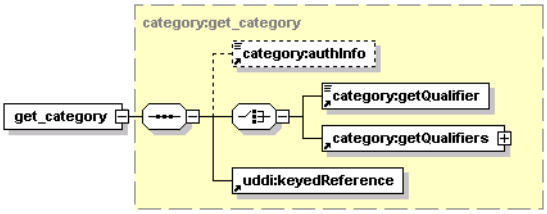
Returns

This API call returns a CategoryList upon success.

get_category

The `get_category` API call is used to get categories having a relation, identified by getQualifier, to the category identified by given keyedReference. If the getQualifier is childCategories, then the call returns categories that have the selected category as their parent. If the siblingCategories getQualifier is used, then categories having same parent as selected category are returned.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `category:getQualifier` and `category:getQualifier` - Control search behavior.

- `uddi:keyedReference` - The category whose relatives shall be received.

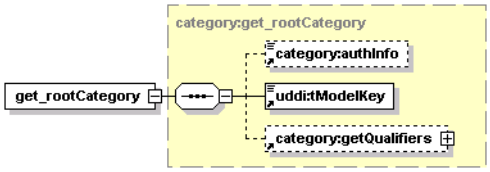
Returns

This API call returns a CategoryList upon success.

get_rootCategory

The `get_rootCategory` API call returns all categories of the Internal taxonomy identified by given tModelKey that have no parent.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:tModelKey` - Required uddiKey value that represents an existing taxonomy tModel.
- `category:getQualifiers` - Control search behavior.

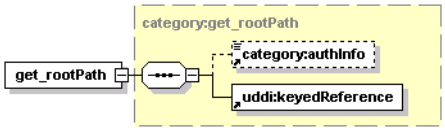
Returns

This API call returns a CategoryList upon success.

get_rootPath

The `get_rootPath` API call returns categories from root category, then its child categories until the selected category in this order: root category, parent's parent, parent and the selected category.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi:keyedReference` - Category to be searched

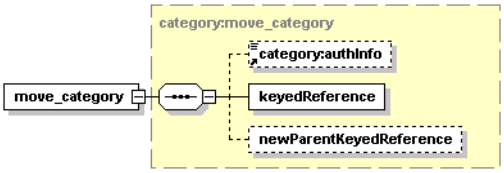
Returns

This API call returns a CategoryList upon success.

move_category

The `move_category` API call is used to move selected category from current parent (if any) to a new parent category. If the newParentKeyedReference is not defined, then the category will have no parent.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `keyedReference` - Category to be deleted.
- `newParentKeyedReference` - Optional category, that becomes new parent of the category.

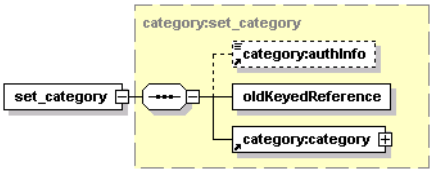
Permissions

This API call requires API manager permission for `org.systinet.uddi.client.category.v3.CategoryApi` and the action `move_category`.

set_category

The `set_category` API call is used to update the selected category in BEA AquaLogic Service Registry.

Syntax



Arguments

- `category:authInfo` - This optional argument is an element that contains an authentication token.
- `oldKeyedReference` - Current category to be updated.
- `category:category` - New category, that will replace selected category.

Permissions

This API call requires API manager permission for `org.systinet.uddi.client.category.v3.CategoryApi` and the action `set_category`.

WSDL

You can find this API's WSDL specification in the file category.wsdl [${doc.web.root}/wsdl/category.wsdl].

API Endpoint

You can find the Category API at `http://<host name>:<port>/uddi/category`.

Java

Systinet Java API is generated from Category WSDL. You are encouraged to browse `org.systinet.uddi.client.category.v3.CategoryApi` and to read and try Category demos.

2.2.4. Approval

The approval process includes the following API sets:

- Section Requestor
- Section Approver
- Section Approval Management
- Section Approval Content Checker

Requestor

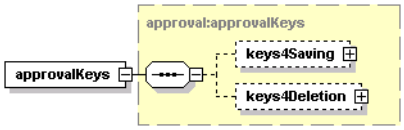
The Systinet Approval Requestor API is used to manage approval requests in BEA AquaLogic Service Registry from the requestor point of view.

Data Structures

The following structures are used by the Systinet Approval Requestor API:

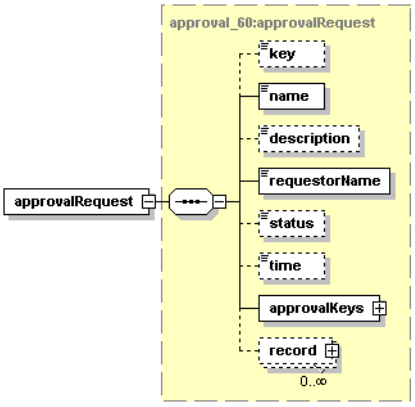
- Section approvalKeys
- Section approvalRequest
- Section approvalRequestInfo
- Section approvalRequestList
- Section approvalRequestRecord
- Section keys4Deletion
- Section keys4Saving
- Section Request
- Section requestInfo
- Section requestList
- Section requestWrapper

approvalKeys



This element is a container for the optional elements `keys4Saving` and `keys4Deletion`.

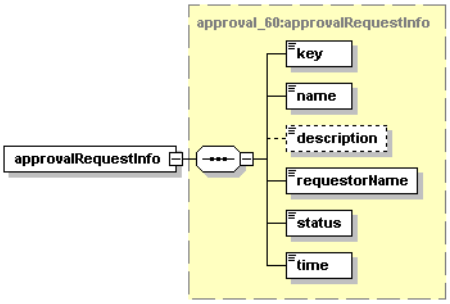
approvalRequest



This structure describes one approval request and contains the following elements:

- `key` - identifies the approval request
- `name` - user-defined name of the request
- `description` - description of the request
- `requestorName` - the loginName of the requestor
- `status` - status of the approver request (open, submitted, closeCancelled, closeRejected, closeApproved, corrupted). The `corrupted` status means some entities from the approval request have been deleted from the registry. It is not possible to search for approval requests using the `corrupted` status.
- `time` - time at which the request switched to the current status (xsd:dateTime)
- `approvalKeys` - keys of element to be saved or deleted from the *discovery registry*
- `record`

approvalRequestInfo

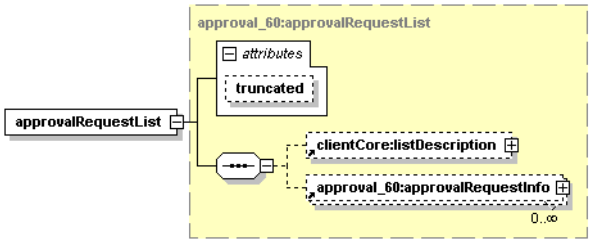


The approvalRequestInfo structure is used by approvalRequestList and contains the following elements:

- `key` - identifies the approvalRequestInfo
- `name` - name of the request
- `description` - description of the approvalRequestInfo
- `requestorName` - loginName of the requestor

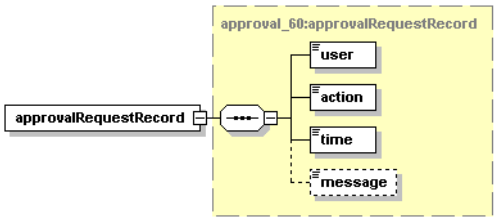
- `status` - status of the approvalRequestInfo (open, submitted, closeCancelled, closeRejected, closeApproved, corrupted). The `corrupted` status means some entities from the approval request have been deleted from the registry. It is not possible to search for approval requests using the `corrupted` status.
- `time` - time at which the request switched to the current status (xsd:dateTime)

approvalRequestList



The approvalRequestList structure contains a list of approvalRequestInfos.

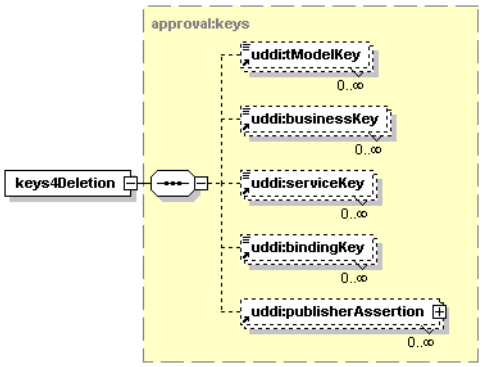
approvalRequestRecord



This structure is used in approvalRequests and contains the following elements:

- `user` - requestor's username
- `action` - action made with the request (saveRequest, askForApproval, cancelRequest, remindApprover, approveRequest, rejectRequest)
- `time` - time at which the approvalRequestRecord switched the current action (xsd:dateTime)
- `message` - may contain a requestor's message to the approval contact or approver's message to the requestor.

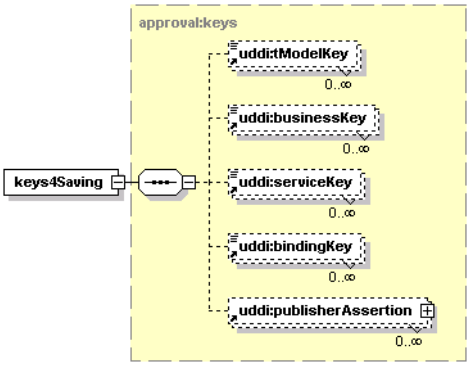
keys4Deletion



This element is a container for UDDI keys or publisher assertions to be deleted from the *discovery registry*. It can contain the optional elements:

- `tModelKey`
- `businessKey`
- `serviceKey`
- `bindingKey`
- `publisherAssertion`

keys4Saving



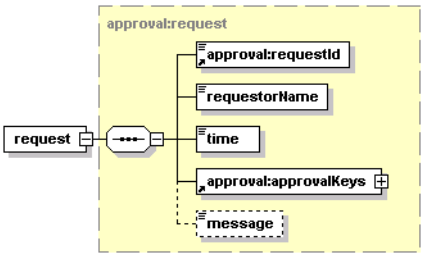
This element is a container for UDDI keys or publisher assertions to be saved to the *discovery registry*. It can contain the optional elements

- `tModelKey`
- `businessKey`
- `serviceKey`
- `bindingKey`
- `publisherAssertion`

Request

Important

This structure is deprecated. User approvalRequest instead.



This element describes one approval request. It contains:

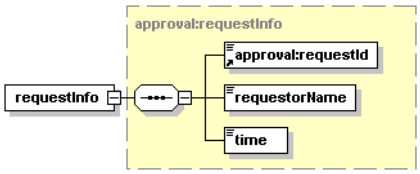
- The mandatory element `requestId` that identifies the request

- A `requestorName` holds the loginName of the requestor.
- The `time` element is set to the time the request was made.
- The `approvalKeys` element is used to store keys of element to be saved or deleted from the *discovery registry*.
- The optional `message` element may contain a requestor's message to the approval contact.

requestInfo

Important

This structure is deprecated. Used approvalRequestInfo instead.



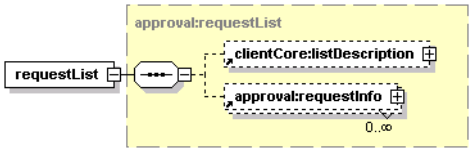
This element contains:

- The mandatory element `requestId` that identifies the request
- A `requestorName` holding loginName of requestor
- A `time` element set to the time the request was made

requestList

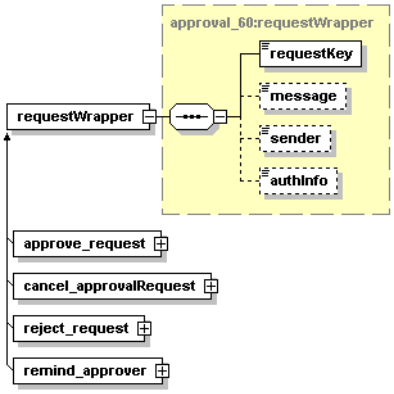
Important

This structure is deprecated. Use approvalRequestList instead.



This element is used to store an optional `listDescription` element that describes the result set and an optional set of `requestInfo` elements.

requestWrapper



This structure wraps the request structure to be inherited in `approve_request`, `cancel_approvalRequest`, `reject_request`, and `remind_approver` structures.

WSDL

You can find the WSDL specification in the file approval.wsdl [${doc.web.root}/wsdl/approval.wsdl].

Java

The Systinet Approval Requestor API is generated from approval.wsdl [${doc.web.root}/wsdl/approval.wsdl]. You are encouraged to browse its `org.systinet.uddi.approval.v3.RequestorApi`.

API Endpoint

The endpoint for the Systinet Approval Requestor API is available at `http://<host name>:<http port>/uddi/requestor`

Approver

The Systinet Approver API is used to manage approval requests in BEA AquaLogic Service Registry from the approver's point of view.

Data Structures

The Systinet Approver API shares the same definition of structures with the Systinet Requestor API. See Section Data Structures for information on these structures.

Operations

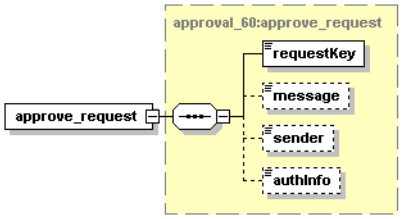
The Approver API has the following operations:

- Section approve_request
- Section Approve
- Section find_approvalRequest
- Section findRequest
- Section getBindingDetail
- Section getBusinessDetail

- Section getOperationalInfo
- Section get_approvalRequest
- Section getRequest
- Section getServiceDetail
- Section getTModelDetail
- Section reject_request
- Section Reject

approve_request

The approve_request API call is used to approve the request. The user must be an approval contact for the requestor.



Arguments

The approve_request API call has the following arguments:

- `requestKey` - a mandatory argument holding the key of the approval request
- `message` - This optional element may contain text that will be delivered to the requestor by an email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user whose authentication token is equal to the token in authInfo. If an administrator (a user with admin manager permission) calls `approve_request`, the authInfo contains the authentication token of the administrator.

  The value of the `sender` argument may contain the loginName of any existing user.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

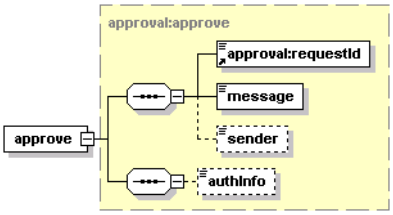
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `approve_request`.

Approve

Important

This operation is deprecated. Use approveRequest instead.

The approve API call is used to approve the request identified by requestId. The user must be an approval contact for the requestor.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - mandatory argument holding key of approval request.
- `message` - optional element that may contain text to be delivered to the requestor via email.
- `sender` - an optional helper element. If set, it must be equal to the loginName of the user.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `approve`.

find_approvalRequest

The find_approvalRequest API call is used to find all approval requests that should be handled (that is, approved or rejected) by the approver.

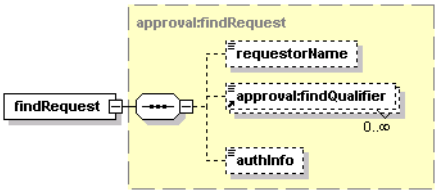
For more information, see Section find_approvalRequest

findRequest

Important

This operation is deprecated. Use find_approvalRequest instead.

The findRequest API call is used to find the requests that the current user is allowed to approve. If the requestorName element is specified, this call only returns requests made by this requestor.



Arguments

The find_request API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestorName` - This optional element identifies the requestor to be searched. The requestorName contains the value of loginName.
- `findQualifier` - The collection of findQualifiers used to alter default behavior.

Behavior

The following findQualifiers affect the behavior of the call:

- The exactMatch findQualifier requires that an exact match be returned.
- The default approximateMatch findQualifier enables SQL wildcard query.
- The sortByNameAsc (default) and sortByNameDesc findQualifiers control the order in which the data is returned.

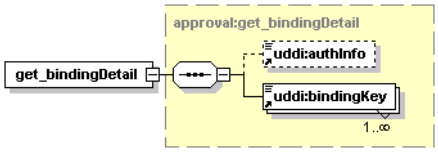
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `findRequest`.

getBindingDetail

The getBindingDetail API call is an extended version of the standard UDDI API call. It is used to get details of the selected bindingTemplate mentioned in the approval request without respect to its access control list. The structure may be configured to allow access only to selected users, but the approval contact must be able to review it.

If the given bindingKey is contained in the approvalKeys structure and the user is the approval contact for the requestor, the ACL check will be skipped and the bindingTemplate will be returned.



Arguments

The getBindingDetail API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding key of the approval request.
- `bindingKey` - This mandatory argument contains the keys of the bindingTemplates to be fetched.

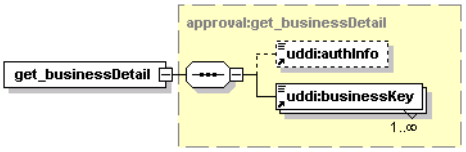
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getBindingDetail`.

getBusinessDetail

The getBusinessDetail API call is an extended version of the standard UDDI API call. It is used to get details of the selected businessEntity mentioned in the approval request without respect to its access control list. The structure may be configured to allow access only selected users, but the approval contact must be able to review it.

If the given businessKey is contained in the approvalKeys structure and the user is the approval contact for the requestor, the ACL check will be skipped and the businessEntity will be returned.



Arguments

The getBusinessDetail API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `businessKey` - This mandatory argument contains the keys of businessEntities to be fetched.

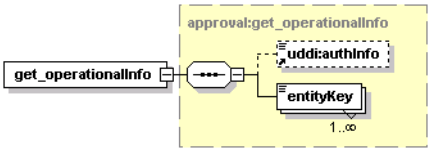
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getBusinessDetail`.

getOperationalInfo

The getOperationalInfo API call is an extended version of the standard UDDI API call. It is used to get details of the selected structure mentioned in the approval request without respect to its access control list. The structure may be configured to allow access only to selected users, but the approval contact must be able to review it.

If the given entityKey is contained in the approvalKeys structure and the user is the approval contact for the requestor, the ACL check will be skipped and the operationalInfo will be returned.



Arguments

The getOperationalInfo API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `entityKey` - This mandatory argument contains the keys of UDDI structures to be fetched.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getOperationalInfo`.

get_approvalRequest

The get_approvalRequest API call is used by an approver to get details of the approval request identified by the requestKey.

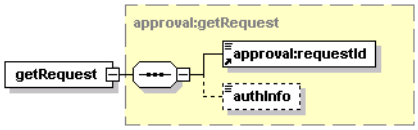
For more information, see Section get_approvalRequest

getRequest

Important

This operation is deprecated. Use get_approvalRequest instead

The getRequest API call is used to get details of the approval request identified by the requestId. The user must be an approval contact for the requestor who makes the request.



Arguments

The getRequest API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.

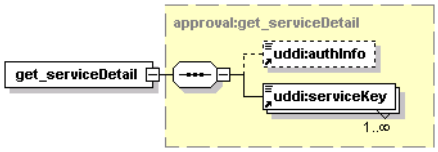
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getRequest`.

getServiceDetail

The getServiceDetail API call is an extended version of the standard UDDI API call. It is used to get details of the selected businessService mentioned in an approval request without respect to its access control list. The structure may be configured to allow access only to selected users, but the approval contact must be able to review it.

If the given serviceKey is contained in the approvalKeys structure and the user is the approval contact for the requestor, the ACL check will be skipped and the businessService will be returned.



Arguments

The getServiceDetail API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `serviceKey` - This mandatory argument contains the keys of businessServices to be fetched.

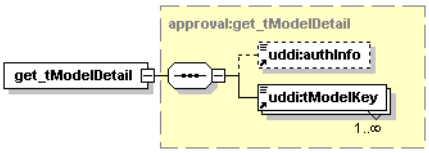
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getServiceDetail`.

getTModelDetail

The getTModelDetail API call is an extended version of the standard UDDI API call. It is used to get details of a selected tModel mentioned in an approval request without respect to its access control list. The structure may be configured to allow access only to selected users, but the approval contact must be able to review it.

If the given tModelKey is contained in the approvalKeys structure and the user is the approval contact for the requestor, the ACL check will be skipped and the tModel will be returned.



Arguments

The getTModelDetail API call has the following arguments:

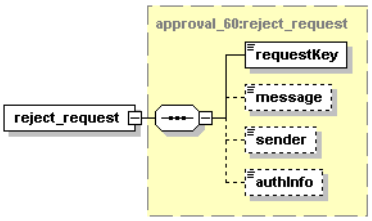
- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `tModelKey` - This mandatory argument contains keys of tModels to be fetched.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getTModelDetail`.

reject_request

The reject_request API call is used to reject a request identified by a requestKey. The user must be an approval contact for the requestor.



Arguments

The reject_request API call has the following arguments:

- `requestKey` - Mandatory argument holding the key of the approval request.
- `message` - This optional element may contain text that will be delivered to the requestor via email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user. If the administrator (a user with admin manager permission) calls `reject_request`, the authInfo contains the authentication token of administrator. The value of the `sender` argument may contain a loginName of any existing user.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

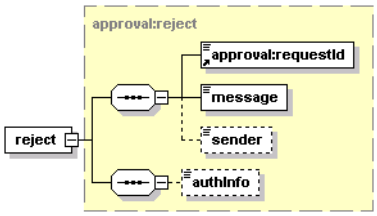
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `reject_request`.

Reject

Important

This operation is deprecated. Use reject_request instead

The Reject API call is used to reject a request identified by requestId. The user must be an approval contact for the requestor.



Arguments

The Reject API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `message` - This optional element may contain text that will be delivered to the requestor via email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `reject`.

WSDL

You can find the WSDL specification in the file approval.wsdl [${doc.web.root}/wsdl/approval.wsdl].

Java

Systinet Approval Approver API is generated from approval.wsdl [${doc.web.root}/wsdl/approval.wsdl]. You are encouraged to browse its `org.systinet.uddi.approval.v3.RequestorApi`.

API Endpoint

The endpoint for the Systinet Approval Approver API is available at `http://<host name>:<http port>/uddi/approver`

Approval Management

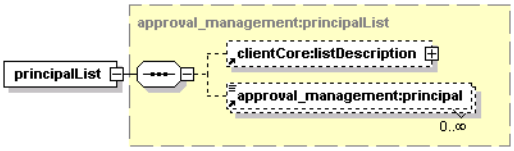
The Systinet Approval Management API is used to manage approval requestors and approval contacts in BEA AquaLogic Service Registry.

Data Structures

The following structures are used by the Systinet Approval Management API:

- Section principalList
- Section Principal
- Section Approver
- Section Requestor

principalList



This element serves as a container for zero or more `principal` elements. The optional `listDescription` element is used to describe the result set.

Principal

This element contains the optional attribute `principalType`, which may be assigned to a user or group. The element's text contains the loginName of the user, or a group name, depending on the principalType value.

Approver

This element contains the optional attribute `principalType`, which may be assigned to a user or group. The element's text contains the loginName of the user, or a group name, depending on principalType value.

Requestor

This element contains the optional attribute `principalType`, which may be assigned to a user or group. The element's text contains the loginName of the user, or a group name, depending on principalType value.

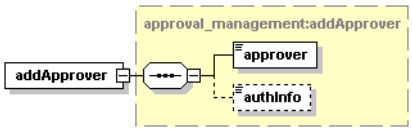
Operations

The Systinet Approval Management API has the following operations:

- Section addApprover
- Section addRequestor
- Section deleteApprover
- Section deleteRequestor
- Section findApprover
- Section findRequestor
- Section isApprover
- Section Save

addApprover

The addApprover API call is used to add a new approval contact to BEA AquaLogic Service Registry.



Arguments

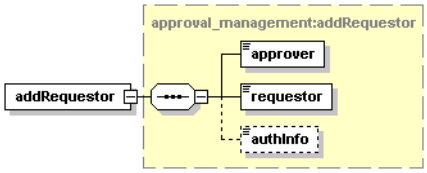
- `authInfo` - This optional argument is an element that contains an authentication token.
- `approver` - This mandatory element identifies the user or group to be added as a new approval contact.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `addApprover`.

addRequestor

The addRequestor API call is used to assign a new requestor to a given approval contact.



Arguments

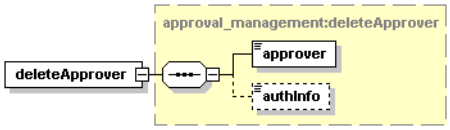
- `authInfo` - This optional argument is an element that contains an authentication token.
- `approver` - This mandatory element identifies an approval contact.
- `requestor` - This mandatory element identifies a new requestor.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `addRequestor`.

deleteApprover

The deleteApprover API call is used to remove the given approval contact from BEA AquaLogic Service Registry.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.

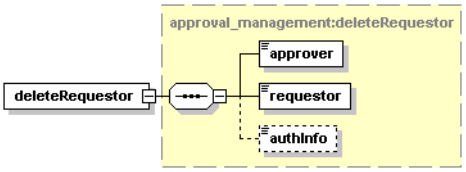
- `approver` - This mandatory element identifies an approval contact.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `deleteApprover`.

deleteRequestor

The deleteRequestor API call is used to remove relationships between the requestor and a given approval contact in BEA AquaLogic Service Registry.



Arguments

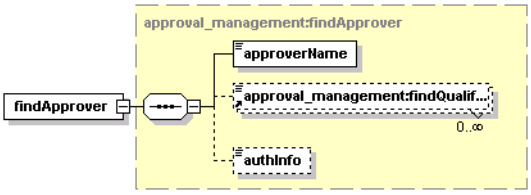
- `authInfo` - This optional argument is an element that contains an authentication token.
- `approver` - This mandatory element identifies an approval contact.
- `requestor` - This mandatory element identifies a new requestor.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `deleteRequestor`.

findApprover

The findApprover API call is used to find approval contacts in BEA AquaLogic Service Registry who match the given criteria. Default findQualifiers are approximateMatch and sortByNameAsc.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `findQualifier` - The collection of findQualifiers used to alter default behavior.
- `approverName` - This mandatory element represent an approval contact to be searched.

Returns

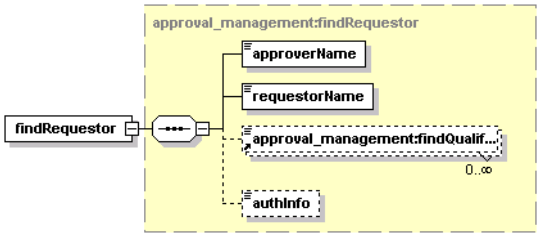
This API call returns the PrincipalList upon success.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `addApprover`.

findRequestor

The findRequestor API call is used to find all requestors of a given approval contact in the registry that match the search criteria.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `findQualifier` - The collection of findQualifiers used to alter default behavior.
- `approverName` - This mandatory element contains the approval contact's name.
- `requestorName` - This mandatory element represents the requestor to be searched. It must be the loginName of the requestor.

Returns

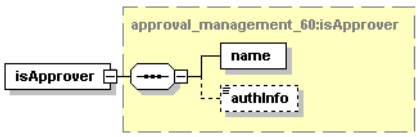
This API call returns the PrincipalList upon success.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `findRequestor`.

isApprover

The isApprover API call finds out whether the user is an approver.

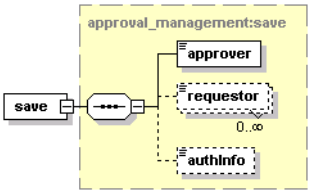


Arguments

- `name` - login name of the user
- `authInfo` - This optional argument is an element that contains an authentication token.

Save

The Save API call combines the addApprover and addRequestor API calls into a single method. If the approval contact does not exist, it is created. Then all requestors are added to this approval contact.



Arguments

The Save API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `approver` - This mandatory element identifies an approval contact.
- `requestor` - This mandatory element identifies new requestors.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.management.ApprovalManagementApi` and the action `addApprover`.

WSDL

You can find the WSDL specification in the file approval_management.wsdl [${doc.web.root}/wsdl/approval_management.wsdl].

Java

Systinet Approval Management API is generated from the Approval Management WSDL. You are encouraged to browse its `org.systinet.uddi.approval.management.ApprovalManagementApi`.

API Endpoint

The endpoint for Systinet Approval Management API is available at `http://<host name>:<http port>/uddi/approvalManagement`

Approval Content Checker

The Systinet Approval Content Checker API provides the approval contact a way to programmatically automate checks of data to be approved. For example, there might be a Web service implementing this API, which requires that each structure be signed. Another implementation may ensure that business services have binding templates. The usage is up to the will of the approval contact.

Operations

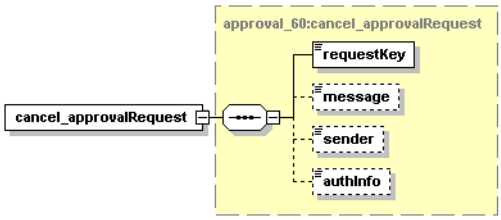
The Systinet Approval Content Checker API has the following operations:

- Section cancelRequest
- Section cancelRequest

- Section delete_approvalRequest
- Section find_approvalRequest
- Section findRequest
- Section get_approvalRequest
- Section getRequest
- Section remind_approver
- Section request_approver
- Section requestApprover
- Section save_approvalRequest
- Section synchronize

cancel_approvalRequest

This API call will cancel a request that has been submitted for approval.



Arguments

The cancel_approvalRequest API call has the following arguments:

- `requestKey` - Mandatory argument holding the key of the approval request.
- `message` - This element may contain text that will be delivered to the approval contact via email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user. If the administrator (a user with admin manager permission) calls `cancel_approvalRequest`, the authInfo contains the authentication token of administrator. The value of the `sender` argument may contain a loginName of any existing user.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

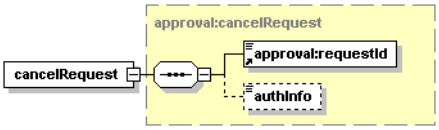
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `cancel_approvalRequest`.

cancelRequest

Important

This operation is deprecated. Use cancel_approvalRequest instead.

The cancelRequest API call is used by the requestor to cancel the request identified by requestId.



Arguments

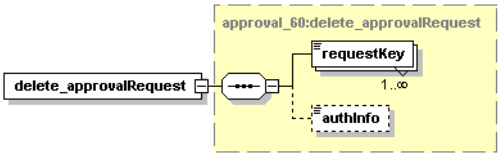
- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of the approval request.
- `message` - This element may contain text that will be delivered to the approval contact via email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `cancelRequest`.

delete_approvalRequest

This operation will delete an approval request. Requests are not deleted automatically after approval or rejection. Requests are held in the registry and a requestor/approver can look at them at any time. This method is used to clean the given requestor's requests.



Arguments

The delete_approvalRequest operation has the following arguments:

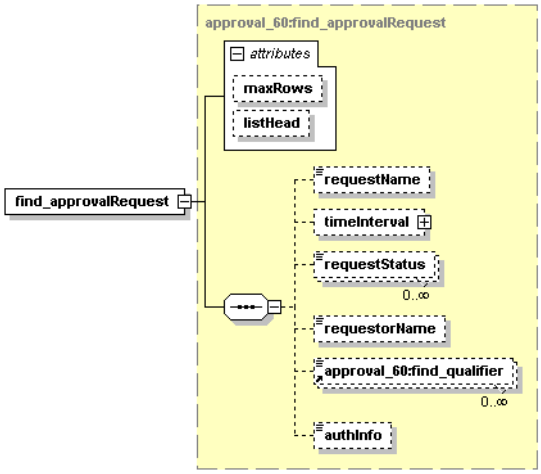
- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestKey` - Mandatory argument holding the key of the approval request.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `cancelRequest`.

find_approvalRequest

The find_approvalRequest API call is used to find all approval requests of the requestor.



Arguments

The find_approvalRequest API call has the following arguments:

- `requestName` - The name of the request
- `timeInterval` - You can specify a time interval search criteria (from, to) having `inclusive` attributes.
- `requestStatus` - A list of request statuses (open, submitted, closeCancelled, closeRejected, closeApproved)
- `requestorName` - This optional element is set to the loginName of the user.
- `approval_60:find_qualifier` - The collection of findQualifiers used to alter default behavior.
- `authInfo` - This optional argument is an element that contains an authentication token.

Behavior

The following findQualifiers affect the behavior of the call:

- The exactMatch findQualifier specifies that an exact match is required.
- The default approximateMatch findQualifier enables an SQL wildcard query.
- The sortByNameAsc (default) and sortByNameDesc findQualifiers control the order in which data is returned, as do the time, requestor and status sorts below.
- sortByTimeAsc, sortByTimeDesc
- sortByRequestorNameAsc, sortByRequestorNameDesc
- sortByStatusAsc, sortByStatusDesc

Permissions

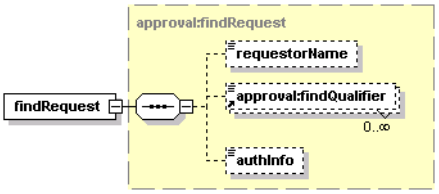
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `find_approvalRequest`.

findRequest

Important

This operation is deprecated. Use find_approvalRequest instead.

The findRequest API call is used to find all approval requests of the requestor who calls this method.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestorName` - This optional element is set to the loginName of the user.
- `findQualifier` - The collection of findQualifiers used to alter default behavior.

Behavior

The following findQualifiers affect the behavior of the call:

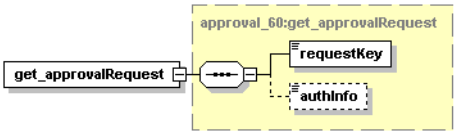
- The exactMatch findQualifier specifies that an exact match is required.
- The default approximateMatch findQualifier enables an SQL wildcard query.
- The sortByNameAsc (default) and sortByNameDesc findQualifiers control the order in which data is returned.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `findRequest`.

get_approvalRequest

The get_approvalRequest API call is used by a requestor to get details of the approval request identified by requestKey.



Arguments

- `requestKey` - Mandatory argument holding the key of an approval request.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

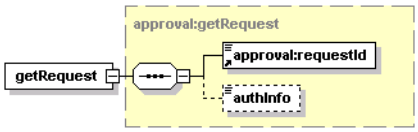
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `get_approvalRequest`.

getRequest

Important

This operation is deprecated. Use get_approvalRequest instead.

The getRequest API call is used by a requestor to get details of the approval request identified by requestId.



Arguments

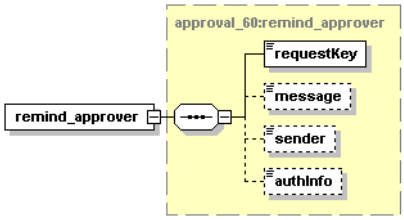
- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestId` - Mandatory argument holding the key of an approval request.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `getRequest`.

remind_approver

The remind_approver API call is used by a requestor to remind the approval contact to review a submitted request. If a requestor is not satisfied with the approver's delay, the requestor can notify the approver about the unhandled approval requests.



Arguments

The remind_approver API call has the following arguments:

- `requestKey` - identifies the request.
- `message` - This optional element may contain text that will be delivered to the approver via email.
- `sender` - Sender is an optional helper element. If set, it must be equal to the loginName of the user. If the administrator (a user with admin manager permission) calls `remind_approver`, the authInfo contains the authentication token of administrator. The value of the `sender` argument may contain the loginName of any existing user.

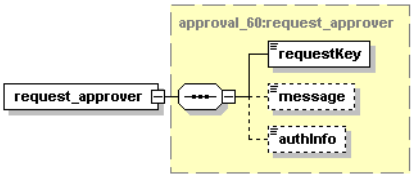
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `remind_approver`.

request_approver

The request_approver API call is used by a requestor to request data for promotion to a *discovery registry*.



Arguments

The request_approver API call has the following arguments:

- `requestKey` - identifies the request
- `message` - This optional element may contain text that will be delivered to the requestor via email.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

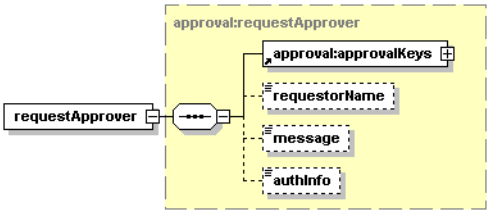
This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `request_approver`.

requestApprover

Important

This operation is deprecated. Use request_approver instead.

The requestApprover API call is used by a requestor to request that an approval contact approve changes to the *publication registry*.



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestorName` - This optional element is set to the loginName of the user.

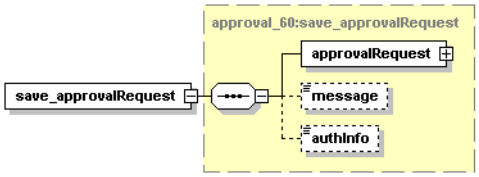
- `message` - This optional element may contain text that will be delivered to the requestor via email.
- `approvalKeys` - This mandatory element is a container for the UDDI keys of structures to be saved or deleted on the *discovery registry*.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `requestApprover`.

save_approvalRequest

This operation is used to save an approval request.



Arguments

The `save_approvalRequest` operation has the following arguments:

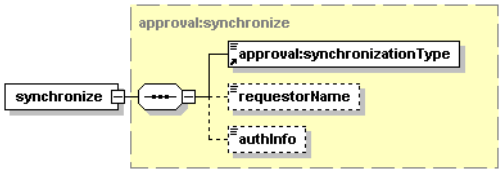
- `approvalRequest`
- `message` - This element may contain text that will be delivered to the approval contact via email.
- `authInfo` - This optional argument is an element that contains an authentication token.

Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `save_approvaRequest`.

synchronize

The Synchronize API call is used to synchronize data on *publication registry* with data on the *discovery registry*. The synchronizationType element is used to choose the way the synchronization will be performed. Possible values are publication_priority, partial_discoveryPriority, and full_discoveryPriority. The synchronization behaviors are described in Section 1.5.3, Synchronization of Data.



Arguments

The Synchronize API call has the following arguments:

- `authInfo` - This optional argument is an element that contains an authentication token.
- `requestorName` - This mandatory element identifies the loginName of the requestor.

- `synchronizationType` - This mandatory element is used to choose the synchronization method.

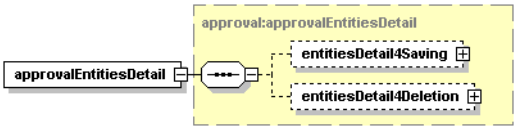
Permissions

This API call requires the API manager permission with the name `org.systinet.uddi.approval.v3.RequestorApi` and the action `synchronize`.

Data Structures

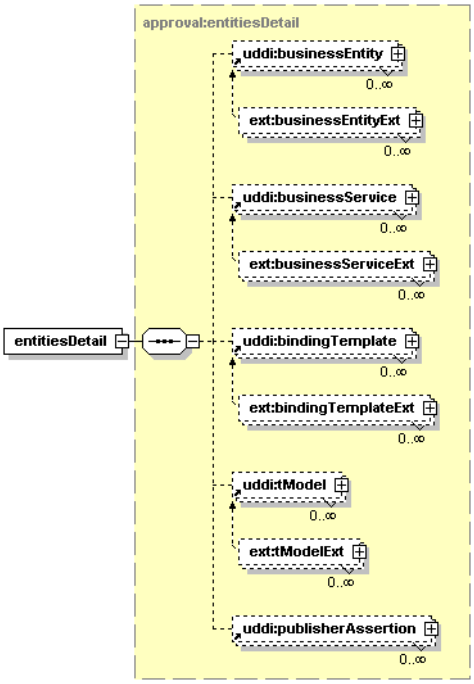
The following structures are used by the Systinet Approval Content Checker API:

approvalEntitiesDetail



This element is a container for the optional elements `entitiesDetail4Saving` and `entitiesDetail4Deletion`. The type for both structures is entitiesDetail.

entitiesDetail



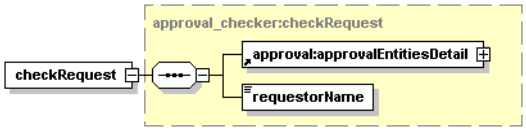
This element holds structure details to be propagated from the *publication registry* to the *discovery registry*. It contains a list of `businessEntities`, `businessServices`, `bindingTemplates`, `tModels` and `publisherAssertions`.

In fact, the extended version of this structure is returned, because it is necessary to transfer the original values of UDDI version 2 keys and standard structures are missing this data.

Operations

checkRequest

The checkRequest API call is made during an approve API call. It is used to perform user-specific checks of data. If the check fails, the implementation returns a DispositionReport with an error code other than E_SUCCESS. See the example in the Developer's Guide, Example 16, Content Checker Implementation



Arguments

The checkRequest API call has the following arguments:

- `approvalEntitiesDetail` - This element contains details of all structures to be checked.
- `requestorName` - This element identifies the requestor by loginName.

Returns

Upon successful completion, a disposition report is returned with a single success indicator.

WSDL

You can find the WSDL specification in the file approval_checker.wsdl [${doc.web.root}/wsdl/approval_checker.wsdl].

Java

Systinet Approval Content Checker API is generated from approval_checker.wsdl [${doc.web.root}/wsdl/approval_checker.wsdl]. You are encouraged to browse `org.systinet.uddi.approval.checker.v3.CheckerApi`.

See also the example, Section 3.6, Writing a Content Checker, in the Developer's Guide,

2.2.5. Administration Utilities

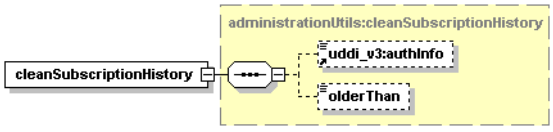
The Systinet Administration Utilities API provides an interface to perform several low level administration tasks in BEA AquaLogic Service Registry.

Operations

cleanSubscriptionHistory

This utility removes subscription histories from BEA AquaLogic Service Registry. If the `olderThan` value is not specified, the utility deletes all historical data; otherwise it deletes data older than the specified value.

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.
- `olderThan` - Optional argument specifying the date before which subscription history is deleted.

Permissions

This API call requires API manager permissions for `org.systinet.uddi.admin.AdministrationUtilsApi` and for the cleanSubscriptionHistory action.

clean_unusedAccounts

This utility is useful when LDAP is used as a user store. BEA AquaLogic Service Registry treats LDAP as read-only and all data from LDAP is mirrored to the registry's database. After you remove users from LDAP using LDAP tools, data removed from LDAP stays in the database. To remove the orphan data from the database, execute the clean_unusedAccounts operation.

Syntax

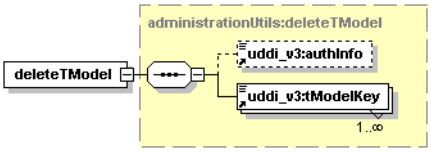
Permissions

This API call requires API manager permissions for `org.systinet.uddi.admin.AdministrationUtilsApi` and for the clean_unusedAccounts action.

deleteTModel

The delete_tModel API removes one or more tModels from BEA AquaLogic Service Registry. Note that the delete_tModel call in the UDDI version 3 specification does not physically remove the tModel from the database; it marks the tModel as deprecated. The delete_tModel call from Administration Utilities can be used to delete such deprecated tModels from the database.

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi_v3:tModelKey` - One or more required uddiKey values that represent existing tModels.

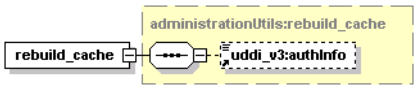
Permissions

This API call requires API manager permission for `org.systinet.uddi.admin.AdministrationUtilsApi` and the action deleteTModel.

rebuild_cache

Database cache stores v3 UDDI structures in database as objects. Using this cache increases performance of v3 inquiry get_business, get_service, get_binding, get_tModel and find_binding operations. On the other hand the cache synchronization take some time mainly in v1 and v2 publishing API operations. The cache can be enabled or disabled by Registry Console. By default, the cache is enabled. Each time caching is switched on, the cache is rebuilt. After the initial rebuild the cache is incrementally synchronized each time save_xxx or delete_xxx operation is performed on v1, v2, v3 publishing API. Explicit rebuild is enabled by rebuild_cache operation. This operation is suitable when data is changed by an administrator in a SQL console (note that such data changing is not recommended).

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.

Permissions

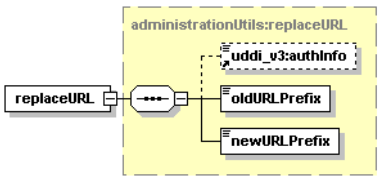
This API call requires API manager permissions for `org.systinet.uddi.admin.AdministrationUtilsApi` and for the rebuild_cache action.

replaceURL

The replaceURL API call is used to replace URL prefixes in the following entities:

- tModel - OverviewDoc URL
- tModelInstanceInfo - overviewDoc URL and DiscoveryURL
- binding template - accessPoint URL

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.
- `oldURLPrefix` - old value of URL prefix
- `newURLPrefix` - new value of URL prefix

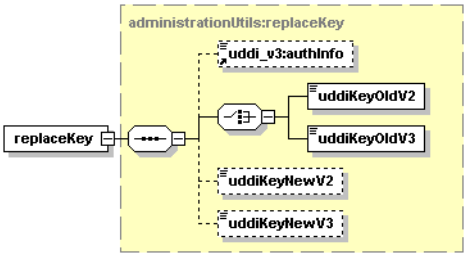
Permissions

This API call requires API manager permission for `org.systinet.uddi.admin.AdministrationUtilsApi` and the action replaceURL.

replaceKey

The replaceKey API call is used to change the uddiKey of a selected UDDI structure in BEA AquaLogic Service Registry. The key must be specified in either UDDI version 3 format or UDDI version 2 format. The optional elements `uddiKeyNewV2` and`uddiKeyNewV3` hold new values of uddiKeys for the selected UDDI structure.

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.
- `uddiKeyOldV2` - Value of the uddiKey of an existing UDDI structure in UDDI version 2 format.
- `uddiKeyOldV3` - Value of a uddiKey of an existing UDDI structure in UDDI version 3 format.
- `uddiKeyNewV2` - New value of the uddiKey in UDDI version 2 format.
- `uddiKeyNewV3` - New value of the uddiKey in UDDI version 3 format.

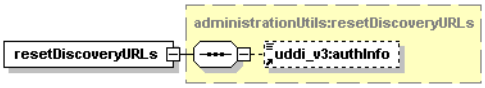
Permissions

This API call requires API manager permission for `org.systinet.uddi.admin.AdministrationUtilsApi` and the action replaceKey.

resetDiscoveryURLs

Sets the discoveryURL value of each businessEntity in BEA AquaLogic Service Registry to its default value.

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.

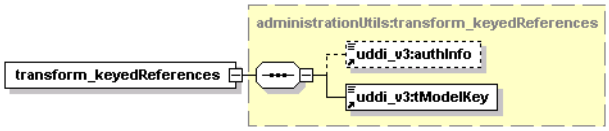
Permissions

This API call requires API manager permission for `org.systinet.uddi.admin.AdministrationUtilsApi` and the action resetDiscoveryURLs.

transform_keyedReferences

This operation is necessary when the type of taxonomy keyValues or the implementation of the taxonomy transformation service have been changed. For more information see, User's Guide, Section 5.4, Taxonomy: Principles, Creation and Validation.

Syntax



Arguments

- `uddi_v3:authInfo` - This optional argument is an element that contains an authentication token.
- `uddi_v3:tModelKey`

Permissions

This API call requires API manager permission for `org.systinet.uddi.admin.AdministrationUtilsApi` and the action transform_keyedReferences.

WSDL

You can find the WSDL specification for this API in administrationUtils.wsdl [${doc.web.root}/wsdl/administrationUtils.wsdl].

API Endpoint

You can find the Administration Utilities API endpoint at `http://<host name>:<port>/uddi/administrationUtils`.

Java

The Systinet Java API is generated from Administration Utils WSDL. You are encouraged to browse for more information.

2.2.6. Replication

The Replication API is used to launch replications in BEA AquaLogic Service Registry.

Operations

Replicate

The `replicate` API call is used to immediately start replications.

Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.

Behavior

When this API call is invoked, it stops the scheduling of replications and, if needed, waits until the completion of current replications. It then starts a new replication process in which replications are rescheduled from this time with the normal replication interval. This results in one of two scenarios:

- If no replications are in process when the `replicate` call is made, the call stops the replication schedule, runs the replication, and restarts the schedule from the time the call was made. For example, if replications had been scheduled on the hour, and the call is made at 9:15, replications will then occur at 10:15, 11:15, and so forth.

- If there is a replication in process when the `replicate` call is made, scheduling is stopped, the call waits for the current process to conclude, runs the replication, and restarts schedule from the time the call was made as in the previous scenario.

WSDL

You can find the WSDL specification in the file replication_v3.wsdl [${doc.web.root}/wsdl/replication_v3.wsdl].

API Endpoint

You can find the Replication API endpoint at `http://<host name>:<port>/uddi/replication`.

Java

The Systinet Java API is generated from the Replication WSDL. You are encouraged to browse its `org.systinet.uddi.replication.v3.ReplicationApi`.

2.2.7. Statistics

The Systinet Statistics API provides useful information about BEA AquaLogic Service Registry usage.

Data Structures

The following structures are used by the Systinet Statistics API:

accessStatisticsDetail

Table 24. Attributes

Attribute	Required
`enable`	yes

This structure is a container for zero or more `apiStatisticsDetail` elements. The `enable` attribute is used to distinguish whether the returned data is consistent or not. If set to false, the Statistics interceptor has been configured not to run and returned data will be outdated.

apiStatisticsDetail

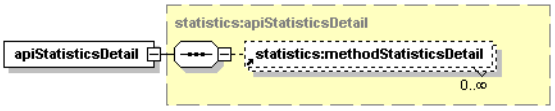


Table 25. Attributes

Attribute	Required
`apiName`	Yes
`requestCount`	Yes
`exceptionCount`	Yes
`lastCall`	Yes

This structure contains information about usage of the API specified in the attribute `apiName` and its methods. It also serves as a container for `methodStatisticsDetail` elements.

The `requestCount` attribute holds a number indicating how many times this API has been used since its last reset or since BEA AquaLogic Service Registry installation.

The `exceptionCount` attribute indicates the number of exceptions that have interrupted execution of the API's methods.

The `lastCall` attribute contains the time this API was last invoked.

methodStatisticsDetail

Table 26. Attributes

Attribute	Required
`methodName`	Yes
`requestCount`	Yes
`exceptionCount`	Yes
`lastCall`	Yes

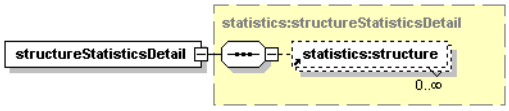
This element contains information about usage of the method specified in the attribute `methodName`.

The `requestCount` attribute holds a number indicating how many times this method has been called since its last reset or since BEA AquaLogic Service Registry installation.

The `exceptionCount` attribute indicates the number of exceptions that have interrupted execution of this method.

The `lastCall` attribute contains the time this method was last invoked.

structureStatisticsDetail



This structure serves as a container for the `structure` element.

Structure

Table 27. Attributes

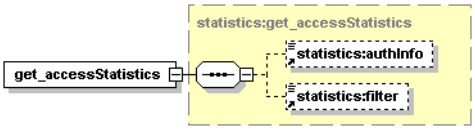
Attribute	Required
`name`	Yes
`count`	Yes

The `structure` element indicates how many UDDI structures of the type given by the `name` attribute are stored in the registry.

Operations

get_accessStatistics

The get_accessStatistics API call is used to fetch information about usage of selected UDDI APIs in BEA AquaLogic Service Registry. The `filter` element is used to specify which APIs' statistics will be returned. If it is empty, the statistics for all APIs are returned.



Arguments

- `statistics:authInfo` - This optional argument is an element that contains an authentication token.
- `statistics:filter` - Optional regular expression to match selected APIs by their name. The wildcard characters ? and * are supported.

Returns

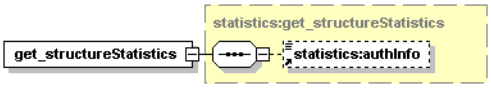
Upon successful completion, an accessStatisticsDetail structure is returned.

Permissions

This API call requires API manager permission for `org.systinet.uddi.statistics.StatisticsApi` and the action get_accessStatistics.

get_structureStatistics

The get_structureStatistics API call is used to get overview information about how many UDDI structures is stored within BEA AquaLogic Service Registry.



Arguments

- `statistics:authInfo` - This optional argument is an element that contains an authentication token.

Returns

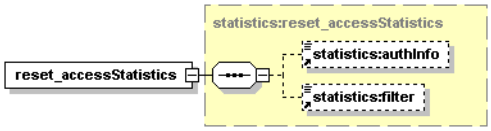
Upon successful completion, an structureStatisticsDetail structure is returned.

Permissions

This API call requires API manager permission for `org.systinet.uddi.statistics.StatisticsApi` and the action get_structureStatistics.

reset_accessStatistics

The reset_accessStatistics API call is used to reset API usage statistics in BEA AquaLogic Service Registry. The optional `filter` element is used to limit affected APIs, if it is not set, statistics for all APIs is removed.



Arguments

- `statistics:authInfo` - This optional argument is an element that contains an authentication token.
- `statistics:filter` - Optional regular expression to match selected APIs by their name. The wildcard characters ? and * are supported.

Permissions

This API call requires API manager permission for `org.systinet.uddi.statistics.StatisticsApi` and the action reset_accessStatistics.

WSDL

You can find the WSDL specification in the file statistics.wsdl [${doc.web.root}/wsdl/statistics.wsdl].

API Endpoint

You can find the Statistics API endpoint at `http://<host name>:<port>/uddi/statistics`.

Java

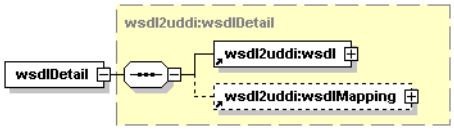
Systinet Java API is generated directly from WSDL. You are encouraged to browse `org.systinet.uddi.statistics.StatisticsApi`.

2.2.8. WSDL Publishing

BEA AquaLogic Service Registry WSDL-to-UDDI mapping is compliant with OASIS's Technical Note, Using WSDL in a UDDI registry Version 2.0 [http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm]. It enables the automatic publishing of WSDL documents to UDDI, enables precise and flexible UDDI queries based on specific WSDL artifacts and metadata, and provides a consistent mapping for UDDI v2 and UDDI v3.

Data Structures

wsdlDetail



wsdlDetail completes information about the WSDL to be mapped.

Arguments

- `wsdl2uddi:wsdl` - Contains URI or physical location of mapped WSDL.
- `wsdl2uddi:wsdlMapping` - Describes `wsdl:types` to be mapped.

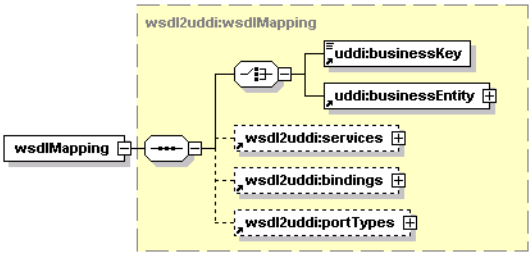
wsdl

WSDL contains information about location of a mapped WSDL.

Arguments

- `wsdlLocation` - The URI or physical location of a mapped WSDL.
- `any` - Used to make extensible documents (see XML schema [http://www.w3.org/TR/xmlschema-1/]). It is generally used as the DOM pattern of a mapped WSDL.

wsdlMapping



WsdlMapping describes the `wsdl:types` to be mapped. It is used to alter the default behavior of mapping the specified WSDL. In contained structures, it is possible to describe each mapped `wsdl:type` correctly. This is to ensure exact mapping and prevent duplication of data in the registry.

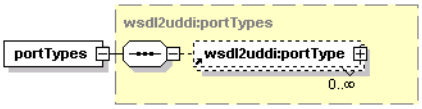
Arguments

- `uddi:businessKey` - Represents the businessKey of an existing `uddi:businessEntity` to which the assigned `wsdl:types` will be mapped.
- `uddi:businessEntity` - Represents an existing businessEntity to which the assigned `wsdl:types` will be mapped.
- `wsdl2uddi:porttypes` - Represents the container of `wsdl:portTypes` to be mapped. `wsdl2uddi:porttypes` makes it possible map a `uddi:tModel` to its corresponding `wsdl:portType`.
- `wsdl2uddi:bindings` - Represents the container of `wsdl:bindings` to be mapped. `wsdl2uddi:bindings` makes it possible to map a `uddi:tModel` to its corresponding `wsdl:binding`.
- `wsdl2uddi:services` - Represents the container of `wsdl:services` to be mapped. `wsdl2uddi:services` makes it possible to map a `uddi:businessService` to its corresponding `wsdl:service`.

Note

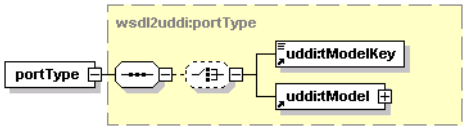
Note that `uddi:businessKey` and `uddi:businessEntity` are mutually exclusive.

portTypes



The portTypes structure is a simple container of one or more `wsdl2uddi:portTypes`.

portType



PortType represents a mapping of `wsdl:portType` in UDDI. It contains information necessary to map the `wsdl:portType` to a corresponding `uddi:tModel` accurately.

Arguments

- `uddi:tModelKey` - Represents the tModelKey of an existing `uddi:tModel` which will be reused or rewritten (depending on the publishingMethod selected by the user) with data from `wsdl:portType`.
- `uddi:tModel` - Represents an existing `uddi:tModel` which will be reused or rewritten (depending on the publishing-Method selected by the user) with data from `wsdl:portType`.

Note

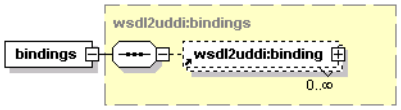
Note that uddi:tModelKey and uddi:tModel are mutually exclusive.

Table 28. Attributes

Name	Required
`name`	optional
`namespace`	optional
`publishingMethod`	optional

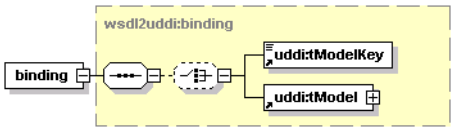
These attributes describe the `wsdl:portType` of the appropriate WSDL. `Name` and `namespace` represent the `wsdl:portType` QName. `publishingMethod` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

Bindings



The bindings structure is a simple container of one or more `wsdl2uddi:bindings`.

binding



A binding represents a mapping of `wsdl:binding` in UDDI. It contains information necessary for the precise mapping of a `wsdl:binding` to the appropriate `uddi:tModel`.

Arguments

- `uddi:tModelKey` - Represents the tModelKey of an existing `uddi:tModel` which will be reused or rewritten (depending on the publishingMethod selected by the user) with data from `wsdl:binding`.
- `uddi:tModel` - Represents an existing `uddi:tModel` which will be reused or rewritten (depending on the publishing-Method selected by the user) with data from `wsdl:binding`.

Note

Note that `uddi:tModelKey` and `uddi:tModel` are mutually exclusive.

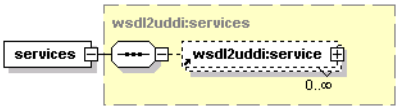
Table 29. Attributes

Name	Required
`name`	optional
`namespace`	optional
`publishingMethod`	optional

These attributes describe the `wsdl:binding` from the appropriate WSDL. `Name` and `namespace` represent the `wsdl:binding` QName.

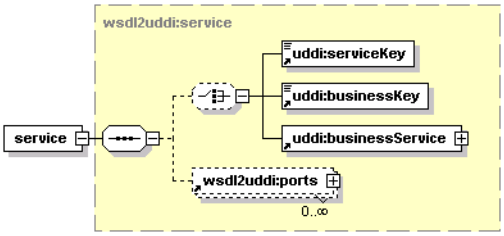
`publishingMethod` represents an enumeration of the available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

Services



The services structure is a simple container of one or more `wsdl2uddi:services`.

service



Service represents the mapping of `wsdl:service` in UDDI. It contains information necessary to map a `wsdl:service` to the appropriate `uddi:businessService` precisely.

Arguments

- `uddi:businessKey` - represents businessKey of an existing `uddi:businessEntity` to which the translated `wsdl:service` will be stored.
- `uddi:serviceKey` - represents the serviceKey of an existing `uddi:businessService` which will be reused or rewritten (depending on the publishingMethod selected by user) with data from wsdl:service.

- `uddi:businessService` - represents an existing `uddi:businessService` which will be reused or rewritten (depending on the publishingMethod selected by user) with data from `wsdl:service`.
- `wsdl:ports` - represents existing `uddi:bindingTemplates` which will be reused or rewritten (depending on the publishingMethod selected by user) with data from `wsdl:service ports`.

Note

Note that `uddi:serviceKey` and `uddi:businessService` are mutually exclusive.

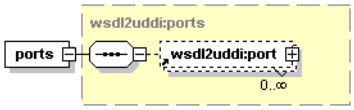
Table 30. Attributes

Name	Use
`name`	optional
`namespace`	optional
`publishingMethod`	optional

These attributes describe the `wsdl:service` from an appropriate WSDL. `Name` and `namespace` represents the `wsdl:service` QName.

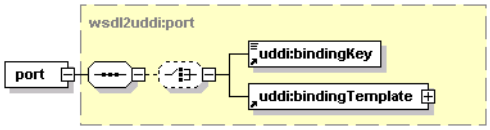
`publishingMethod` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, `reuse`, or `ignore`. The default publishingMethod is `reuse`.

ports



The ports structure is a simple container for one or more `wsdl2uddi:ports`.

port



Port represents a mapping of `wsdl:port` in UDDI. It contains information necessary to map the `wsdl:port` to the appropriate `uddi:bindingTemplate` precisely.

Arguments

- `uddi:bindingKey` - Represents the bindingKey of an existing `uddi:bindingTemplate` which will be reused or rewritten (depending on the publishingMethod selected by user) with data from `wsdl:port`.
- `uddi:bindingTemplate` - Represents an existing `uddi:bindingTemplate` which will be reused or rewritten (depending on the publishingMethod selected by user) with data from `wsdl:service`.

Note

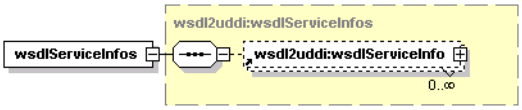
Note that `uddi:bindingKey` and `uddi:bindingTemplate` are mutually exclusive.

Table 31. Attributes

Name	Required
`name`	optional
`publishingMethod`	optional

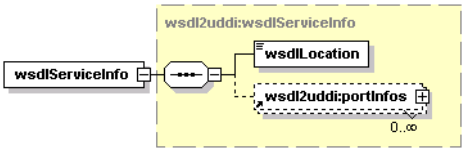
These attributes describe the `wsdl:port` from an appropriate WSDL.`Name` represents the `wsdl:port` name. `publishing-Method` represents an enumeration of available mapping use cases. It can be set to `rewrite`, `create`, or `reuse`. The default publishingMethod is `reuse`.

wsdlServiceInfos



The wsdlServiceInfo structure is a simple container of one or more `wsdl2uddi:wsdlServiceInfos`.

wsdlServiceInfo



The wsdlServiceInfo completes information about the wsdlLocation and `uddi:businessService` being searched.

Arguments

- `wsdlLocation` - The URI or physical location of a WSDL.
- `wsdl2uddi:portInfos` - Container for wsdl2uddi:ports which contain the `wsdl:port` mapped to the appropriate `uddi:bindingTemplate`.

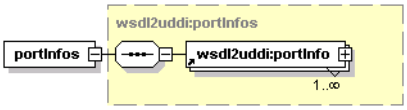
Table 32. Attributes

Name	Required
`name`	required
`namespace`	required
`serviceKey`	required

These attributes describes how the `wsdl:service` is mapped from the appropriate WSDL. `Name` and `namespace` represent the `wsdl:service` QName.

The `serviceKey` represents the `uddi:businessService` on which the `wsdl:service` is mapped.

PortInfos



The portInfos structure is a simple container of one or more `wsdl2uddi:portInfos`.

portInfo

The portInfo completes information about `uddi:bindingTemplates` used in the `uddi:businessService` being searched.

Arguments

- `uddi:accessPoint` contains information about accessing the `uddi:businessService` being searched.

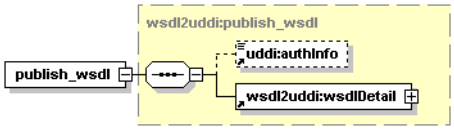
Table 33. Attributes

Name	Required
`name`	required
`bindingKey`	required

These attributes describe how the `wsdl:port` is mapped from the appropriate WSDL. `Name` represents the `wsdl:port` name. `BindingKey` represents the `uddi:bindingTemplate` on which the `wsdl:port` is mapped.

Operations

publish_wsdl



Publish_wsdl ensures the publishing of a WSDL to a UDDI registry. It uses the Publishing API to store translated `wsdl:types` to the UDDI registry. For more information about the Publishing API, please see UDDI v3 - publishing API [http://uddi.org/pubs/uddi_v3.htm#_Toc53709290]).

By default UDDI entities are rewritten by data contained in `wsdl:types` as follows: Each `wsdl:type` is first searched on the specified registry. The found UDDI entity is rewritten, or a new entity is created if one is not found. However, the user can specify how the `wsdl:types` will be published to the registry.

You can alter the default publish behavior and define which `wsdl:types` will be mapped on the appropriate UDDI entity and, naturally, whether the UDDI entity will be created, rewritten, or reused.

For more information about publish behavior and its use cases, see publishingMethod. Below are some rules by which `wsdl:types` are assigned to the appropriate UDDI entities depending on whether the `wsdl:type` is found on the user account or on a foreign account. Note that `wsdl:services` are searched only on the user's account, unlike `wsdl:portType` or `wsdl:binding`. This is because it is preferable to use tModels from a foreign account rather then tModels translated from a WSDL.

publishingMethod

PublishingMethod describes the behavior of the publish operation. In accordance with the set behavior, the corresponding `wsdl:type` will be mapped to the UDDI registry.

Note that publish_wsdl is set to `reuse` by default. However, if a user wants to rewrite an entity or a create a new entity, the default behavior can be changed from "reuse" to "rewrite" or "create" to ensure unique mapping.

`Use cases`

- `rewrite` - `wsdl:type` is searched on the registry and the found UDDI structure is redrawn by data of that `wsdl:type`. If the `wsdl:type` is not found, a new one will be created.
- `reuse` - The default behavior of the publish operation. Using this behavior, the user is able to reuse an entire existing UDDI structure. The found UDDI entity will not be redrawn by data of that `wsdl:type`. Note that when using this method, inconsistencies may occur between the published `wsdl:type` and the corresponding UDDI entity. This behavior should be helpful when we need to use existing tModels instead of tModels mapped from `wsdl:portTypes` or `wsdl:bindings` (For example, `uddi:hostingRedirectors`).
- `create` - This method is used mainly for testing purposes. By using this behavior a new UDDI entity is created from the `wsdl:type` regardless of whether the UDDI entity already exists on the registry.

  Important

  When using this behavior, undesirable duplications may occur. It is necessary to use this behavior carefully.

- `ignore` - This method is used when you do not want to publish the UDDI entity. You can restrict which parts of the WSDL document will be published.

Arguments

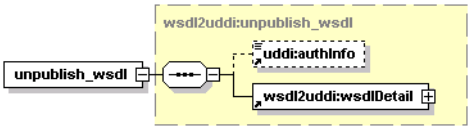
- `uddi:authInfo` - This required argument is the string representation of the `uddi:authToken`.
- `wsdl2uddi:wsdlDetail` - Completes WSDL location and user-defined WSDL mapping rules. For more information, please see wsdl2uddi:wsdlDetaill.

  Here the user can specify which `wsdl:type` from the WSDL corresponds to the entity on the target registry and how the specified `wsdl:type` will be mapped. For more information, please see wsdl2uddi:publishingMethod.

Returns

`wsdl2uddi:wsdlDetail` - Contains detailed information about how the individual `wsdl:types` are published. For more information, please see wsdl2uddi:wsdlDetaill.

unpublish_wsdl



Unpublish_wsdl ensures unpublishing of WSDL from UDDI registry. It uses the Publishing API to delete UDDI entities corresponding to `wsdl:types` from a UDDI registry. For more information about the Publishing API, please see UDDI v3 - publishing API [http://uddi.org/pubs/uddi_v3.htm#_Toc53709290].

Each `wsdl:type` is first searched on the specified registry. The found UDDI entity is deleted or if the entity is not found it is simply omitted. Found tModels are either physically deleted or only marked as `deprecated` in accordance with configuration. (When tModels are deleted by their owners, they are generally marked as deprecated. Usually only the administrator can permanently delete deprecated tModels from the registry.)

Arguments

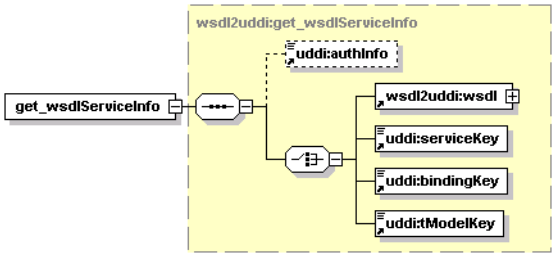
- `uddi:authInfo` - This required argument is the string representation of the `uddi:authToken`.

- `wsdl2uddi:wsdlDetail` - completes the WSDL location and user-defined WSDL unpublish rules. For more information, please see wsdl2uddi:wsdlDetaill. Here the user can specify which `wsdl:type` from a WSDL corresponds to the UDDI entity existing on the target registry. This is because that `wsdl:type` can occur more than once on a registry.

Returns

`wsdl2uddi:wsdlDetail` - Contains detailed information about how individual `wsdl:types` are unpublished from a target registry. For more information, please see wsdl2uddi:wsdlDetaill.

get_wsdlServiceInfo



Get_wsdlServiceInfo discovers `uddi:businessServices` corresponding to `wsdl:services` from a particular WSDL. It uses the Inquiry API to get UDDI entities matching `wsdl:types`. For more information about the Inquiry API, please see UDDI-inquiry API [http://uddi.org/pubs/uddi_v3.htm#_Toc53709271].

This operation discovers corresponding UDDI entities either on the user's account or on the foreign account (in accordance with the specified `uddi:authInfo`). In consideration with multiple occurrences of UDDI entities corresponding to `wsdl:types`, the search algorithm optimizes output in accordance with relations between individual `wsdl:types` from the given WSDL. Only the wsdl2uddi:wsdlServiceInfo corresponding exactly to the `wsdl:service` from the WSDL (that is, that contains all `wsdl:types` from the appropriate WSDL) will be returned.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.

- `wsdl2uddi:wsdl` - An argument used to discover wsdl2uddi:wsdlServiceInfos. This argument ensures that only the `uddi:businessService` corresponding exactly to the `wsdl:service` from that WSDL will be returned. For more information, please see wsdl2uddi:wsdl).

- `uddi:serviceKey` - `uddi:serviceKey` of `uddi:businessService` existing on the target registry. Note that only `uddi:businessServices` containing a "WSDL Type Category System" (that is, the `uddi:categoryBag` of a found `uddi:businessService` must contain a `uddi:keyedReference` with a `uddi:tModelKey` representing "WSDL Type Category System" and the keyValue "service") will be returned.

- `uddi:bindingKey` - `uddi:bindingKey` of `uddi:bindingTemplate` existing on the target registry. For UDDI v3 holds that only uddi:businessServices which contain `uddi:bindingTemplate` corresponding to a given `uddi:bindingKey` with the "WSDL Type" Category System. (that is, the `uddi:categoryBag` of a found `uddi:bindingTemplate` must contain `uddi:keyedReference` with `uddi:tModelKey` representing "WSDL Type Category System" and the keyValue "binding") will be returned. Naturally this "WSDL Type Category System" must also be contained in the appropriate `uddi:businessService`.

  Note that `uddi:bindingTemplates` in v2 do not contain `uddi:categoryBag`. Even though the found `uddi:bindingTemplate` must contain `uddi:tModels` compliant with "WSDL Type Category System" in its uddi:tModelInstanceDetails.

- `uddi:tModelKey` - the `uddi:tModelKey` of the `uddi:tModel` existing on the target registry. Note that only uddi:businessServices which use uddi:tModels compliant with "WSDL Type Category System" will be returned. That

is, the `uddi:categoryBag` of the found `uddi:tModel` must contain `uddi:keyedReference` with `uddi:tModelKey` representing "WSDL Type Category System" and the keyValue "binding" or "portType"). Naturally, this "WSDL Type Category System" must also be contained in the appropriate `uddi:businessService`.

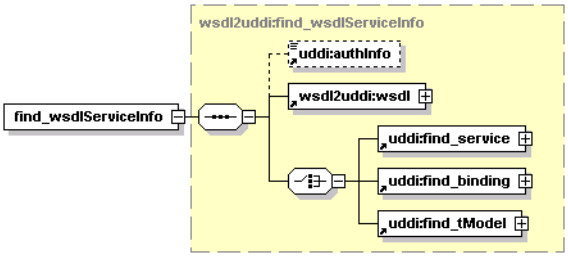
Note

Note that `wsdl2uddi:wsdl`, `uddi:serviceKey`, `uddi:bindingKey` and `uddi:tModelKey` are mutually exclusive.

Returns

`wsdl2uddi:wsdlServiceInfos` - Contains UDDI entities corresponding to `wsdl:types` from the specified WSDL. For more information, please see wsdl2uddi:wsdlServiceInfos.

find_wsdlServiceInfo



This operation is a bit more complex than wsdl2uddi:get_wsdlServiceInfo. Find_wsdlServiceInfo discovers `uddi:businessServices` corresponding to `wsdl:services` from a particular WSDL. It uses the Inquiry API to find UDDI entities matching `wsdl:types`. For more information about the Inquiry API, please see UDDI-inquiry API [http://uddi.org/pubs/uddi_v3.htm#_Toc53709271]).

This operation discovers corresponding UDDI entities either on the user's account or on a foreign account (in accordance with the specified `uddi:authInfo`). In consideration for multiple occurrence of UDDI entities corresponding to `wsdl:types`, the search algorithm optimizes output in accordance with relations between individual `wsdl:types` from the specified WSDL and the `uddi:find_xx` structure specified by the user. Only the wsdl2uddi:wsdlServiceInfo corresponding exactly to the wsdl:service from the WSDL will be returned, that is, the wsdl2uddi:wsdlServiceInfo containing all `wsdl:types` from the appropriate WSDL at once, and satisfying the user's defined `uddi:find_xx`.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `wsdl2uddi:wsdl` - required argument used to discover wsdl2uddi:wsdlServiceInfos. This argument ensures that only the `uddi:businessService` corresponding exactly to the `wsdl:service` from that WSDL will be returned. For more information, please see wsdl2uddi:wsdl.
- `uddi:find_service` - Argument used for a more detailed description of search criteria. For more information, see uddi:find_service [http://uddi.org/pubs/uddi_v3.htm#_Toc53709283]. Found `uddi:businessServices` must follow the same rules as in the case of wsdl2uddi:get_wsdlServiceInfo.
- `uddi:find_binding` - Argument used for a more detailed description of search criteria. For more information, see uddi:find_binding [http://uddi.org/pubs/uddi_v3.htm#_Toc53709280]. Found `uddi:businessServices` and `uddi:bindingTemplates` must follow the same rules as in the case of wsdl2uddi:get_wsdlServiceInfo.
- `uddi:find_tModel` - Argument used for a more detailed description of search criteria. For more information, see uddi:find_tModel [http://uddi.org/pubs/uddi_v3.htm#_Toc53709284]. Found UDDI entities must follow the same rules as in the case of wsdl2uddi:get_wsdlServiceInfo.

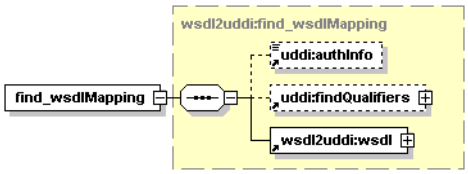
Note

Note that `uddi:find_service`, `uddi:find_binding` and `uddi:find_tModel` are mutually exclusive.

Returns

`wsdl2uddi:wsdlServiceInfos` - Contains UDDI entities corresponding to `wsdl:types` from the specified WSDL. For more information, please see wsdl2uddi:wsdlServiceInfos.

find_wsdlMapping



This operation finds mapping of the WSDL document.

Arguments

- `uddi:authInfo` - This argument is the string representation of the `uddi:authToken`.
- `uddi:findQualifiers` - See Find Qualifiers [http://uddi.org/pubs/uddi-v3.0.1-20031014.htm#_Toc53709275]
- wsdl2uddi:wsdl

Returns

This operation returns wsdl2uddi:wsdlMapping.

WSDL

wsdl2uddi_v2.wsdl.wsdl [${doc.web.root}/wsdl/wsdl2uddi_v2.wsdl]

wsdl2uddi_v3.wsdl.wsdl [${doc.web.root}/wsdl/wsdl2uddi_v3.wsdl]

API Endpoint

You can find the WSDL2UDDI API endpoint at `http://<host name>:<port>/uddi/wsdl2uddi`.

Java

Demos v2: WSDL2UDDI demos

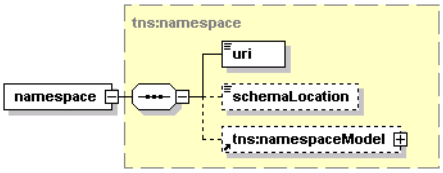
Demos v3: WSDL2UDDI demos

2.2.9. XML Publishing

Systinet XML-to-UDDI mapping enables the automatic publishing of XML documents to UDDI and precise, flexible UDDI queries based on specific XML metadata.

Data Structures

namespace



This structure is a container for a namespace.

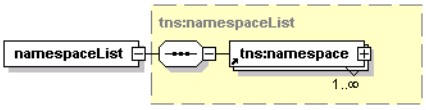
Table 34. Attributes

`publishingMethod`	optional

Arguments

- `uri` - URI of the namespace.
- `schemaLocation` - This argument holds the location of the schema specified by the XML document using `xsi:schemaLocation` declaration.
- tns:namespaceModel - This argument holds mappings that represent this namespace.

namespaceList

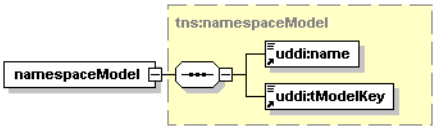


This structure represent a list of namespaces.

Arguments

- tns:namespace - represents a member of the namespaceList.

namespaceModel

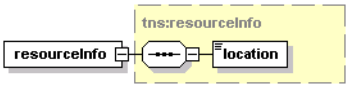


This structure describes mapping of a particular namespace (or no namespace) within the XML document.

Arguments

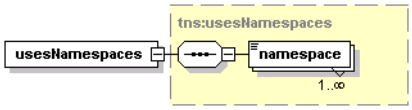
- `uddi:name` - name of the tModel corresponding to the namespace's XML Schema
- `uddi:tModelKey` - tModelKey name of the tModel corresponding to the namespace's XML Schema

resourceInfo



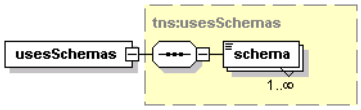
This structure holds the location of the resource.

usesNamespaces



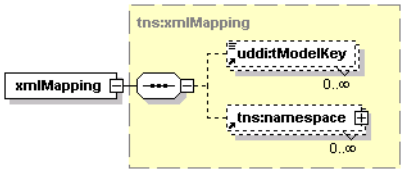
This structure represents a list of namespaces.

usesSchemas



This structure holds a list of schemas.

xmlMapping

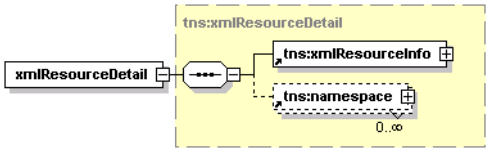


This structure represents an XML mapping.

Arguments

- `uddi:tModelKey` - tModelKeys of tModels that correspond to the XML document. When used with `publish_xml`, zero tModelKeys or a single tModelKey can be used.
- `tns:namespace` - List of namespaces used in the XML document with their mappings to UDDI tModels

xmlResourceDetail



This structure describes the published XML document. It contains the location of the document and a list of the namespaces referenced by the XML document. The document declares a prefix for the XML namespace using the `xmlns:` declaration.

Arguments

- `tns:xmlResourceInfo` - contains the location of the XML document (URI)
- `tns:namespace` - a list of namespace information, one entry for each namespace used in the XML document

xmlResourceDetails

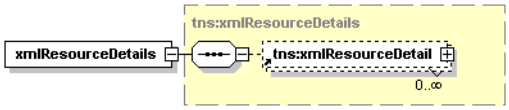
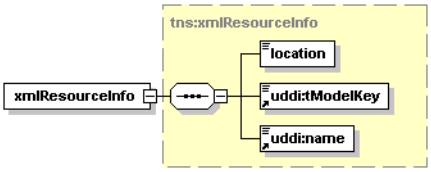


Table 35. Attributes

`truncated`	optional

This structure, used in the result list of the `find_xml` query, provides information about a published XML document.

xmlResourceInfo

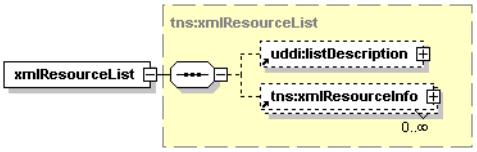


This structure, served as a result from the `find_xml` query, represents a simple information object about an XML document. It contains information needed for a simple presentation and identifies the UDDI tModel holding the rest of the information.

Arguments

- `location` - the location (URI) of the XML document
- `uddi:tModelKey` - tModelKey of the tModel that corresponds to the XML document. The key can be used with `get_xmlDetail`
- `uddi:name` - name of the tModel

xmlResourceList



This structure contains a list of XML resources, possibly a sublist or a large result set. When only a sublist is returned, the structure must contain the `listDescription` element.

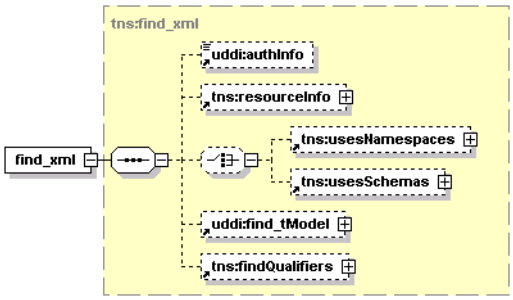
Arguments

- `uddi:listDescription` - description of the result list, in case it is a subset of a larger result set.
- `tns:xmlResourceInfo` - information about individual results (published XML documents)

Operations

find_xml

Syntax



This operation finds the XML document.

Table 36. Attributes

`listHead`	optional
`maxRows`	optional

Arguments

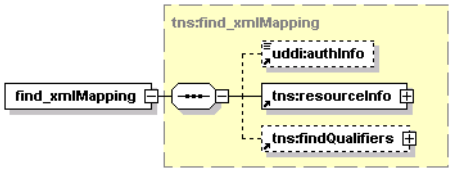
- `uddi:authInfo` - This optional argument is the string representation of the uddi:authToken.
- `tns:resourceInfo` - URI location of the published XML document.
- `tns:usesNamespaces` - search by XML namespace URIs of the published XML document.
- `tns:usesSchemas` - schemas of the published XML document.
- `uddi:find_tModel` - Argument used for a more detailed description of search criteria. For more information, see uddi:find_tModel [http://uddi.org/pubs/uddi_v3.htm#_Toc53709284]. The search criteria implied by the other members of `find_xml` structure will be merged with the contents of `uddi:find_tModel` contents.

Returns

This API call returns the xmlResourceList on success.

find_xmlMapping

Syntax



This operation finds a mapping among the UDDI entities for the XML resource.

Table 37. Attributes

`policy`	optional

Arguments

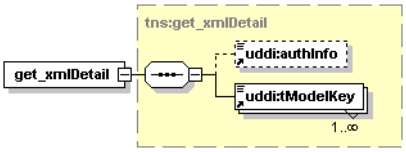
- `policy` - The policy (attribute) may be one of:
  - `automatic` (default) switches the operation to find UDDI entities for all XSD references, even those assumed from the XML namespace. For each used namespace URI, the function attempts to find all XML Schema tModels registered in UDDI which define the namespace contents and return their tModelKeys.
  - `locations` restricts the search only to namespaces containing `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`. For these namespaces, the function returns tModelKeys of the XML Schema tModels stored in the registry matching the namespace. The operation ignores usage of namespaces lacking the `schemaLocation` attribute and does not return matching UDDI tModelKeys in this case.
- `uddi:authInfo` - This required argument is the string representation of the `uddi:authToken`.
- `tns:resourceInfo` - URI location of the XML document.

Returns

This API call returns `xml2uddi:xmlMapping` upon success.

get_xmlDetail

Syntax



This operation returns the registered mapping information for the XML document identified by a key.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - Required uddiKey value representing an existing XML tModel.

Returns

This API call returns the xml2uddi:xmlResourceDetails which describes the XML and the schemas used to define the document elements.

publish_xml

Syntax

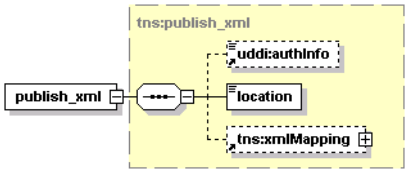


Table 38. Attributes

`policy`	optional
`publishingMethod`	optional
`namespacePublishing`	optional

This operation creates a new instance of a tModel representing the XML document.

Note

This operation does not publish the contents of an XSD file.

All existing information which overlaps with the XML-to-UDDI mapping are overwritten, or removed from the registry, according to the input data. If the arguments pass information about a namespace, the passed information will be used. Any extraneous schema tModel references will be purged from the XML tModel's category bag.

Arguments

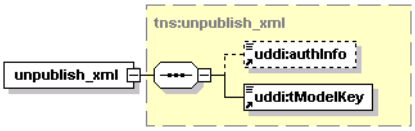
- `policy` - This optional attribute may have one of the following values:
  - `automatic` (default) - all XSD references found in the XML document, even those assumed from XML namespace prefix declarations will be published.
  - `explicit` - Only the XSD references provided in the call will be published.
  - `locations` - references to XSDs that are given with the `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` will be published.
- `publishingMethod` - This optional attribute specifies whether the operation creates a new tModel (possibly assigned its name/value from the caller-supplied structure), or renews the passed tModel contents.
- `namespacePublishingMethod` - This optional attribute controls whether new tModels will be created for namespaces, the existing tModels will be reused, or the namespaces will be ignored. When `reuse` is specified, the target tModelKey can be also given. It is an error to specify a tModelKey that does not exist in the Registry. When `create` is specified for a namespace and the tModelKey is given, it is used as the publisher-assigned key for the new tModel. Possible values are `create`, `reuse`, and `ignore`.
- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `location` - location of the XML document.
- `tns:xmlMapping` - mapping structure to be used in XML publishing.

Returns

This API call returns the xmlResourceDetail on success.

unpublish_xml

Syntax



This operation removes the metadata (tModel) for the XML document, identified by tModelKey. Since the XML structure is not published, data about the XML document are effectively discarded. If the XML document's metadata is referenced from outside, the unpublish call fails. The dispositionReport will contain keys of the UDDI entities that refer to the XML document.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - tModelKey of the XML document.

Returns

This API call returns the xmlResourceDetail on success.

WSDL

xml2uddi_v3.wsdl [${doc.web.root}/wsdl/xml2uddi_v3.wsdl]

API Endpoint

You can find the XML2UDDI API endpoint at `http://<host name>:<port>/uddi/xml2uddi`.

Java

2.2.10. XSD Publishing

Systinet XSD-to-UDDI mapping enables the automatic publishing of XML Schema Documents into UDDI and enables precise, flexible UDDI queries based on specific XML schema metadata.

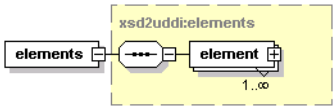
The mapping of XML Schema Document information to UDDI covers:

- XML types - Types declared at the global level in the XML Schema Document. These types are mapped to tModels in UDDI.
- XML elements - XML elements declared at the global level in the XML Schema Document. These elements are mapped to tModels in UDDI.
- References to other XML namespaces - Information about imported schemas are stored in the registry.

The API allows the user to search for an schema's tModels based on the namespace they define, or the elements and types they declare within that namespace. The API can also extract the published information back from the registry, so it can be accessed as a list of elements, types, and schemas rather than tModels and other UDDI entities.

Data Structures

Elements

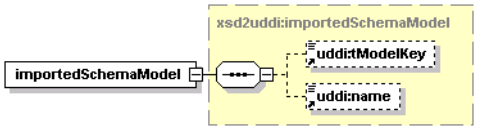


This structure represents elements declared by the published XML Schema Document.

Arguments

- `element` - This argument represents an element declared by the published XML Schema Document.

importedSchemaModel

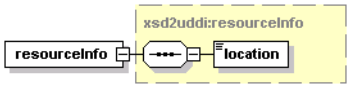


This structure contains the basics of the imported XML Schema tModel.

Arguments

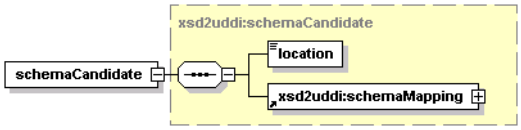
- `uddi:tModelKey` - The key of the tModel of the schema of the imported XML namespace.
- `uddi:name` - The name of that schema's tModel.

resourceInfo



This structure describes the location of the XML Schema Document.

schemaCandidate

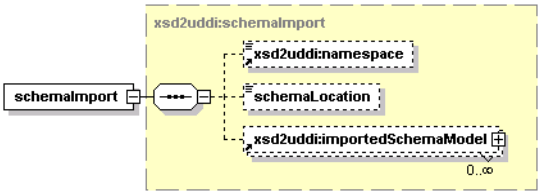


This structure holds possible mappings of how the XML Schema Document can be published.

Arguments

- `location` - The location of the candidate XML Schema Document.
- `xsd2uddi:schemaMapping` - The mapping of the candidate XML Schema Document contents

schemaImport

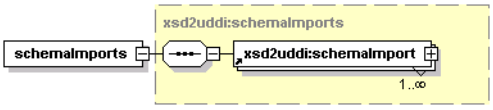


This structure holds the imported namespace, that is, the list of possible mappings for this `xsd:import`, for an `xsd:import` clause in the XML Schema Document. If a specific location is specified in the XML Schema Document text for the imported XML Schema Document, it is also present.

Arguments

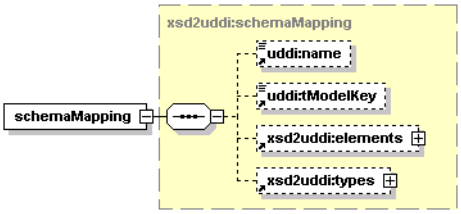
- `xsd2uddi:namespace` - The imported namespace. If missing, a no-namespaced XML schema is imported
- `schemaLocation` - The location for the XML Schema Document, if given explicitly. If the imported XML Schema Document does not specify an exact schema location, this value is null.
- `xsd2uddi:importedSchemaModel` - The tModel information of the candidates for this import.

schemaImports



This structure describes a list of `xs:imports` in the schema.

schemaMapping

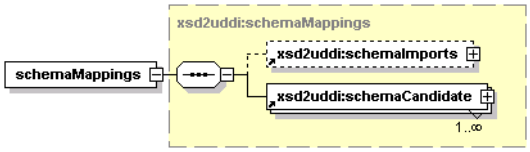


This structure describes a mapping of the XSD contents to an individual XSD tModel and its contents.

Arguments

- `uddi:name` - Name of the XML Schema tModel.
- `uddi:tModelKey` - tModelKey for the XML Schema tModel
- `xsd2uddi:elements` - Mapping for contained XML elements
- `xsd2uddi:types` - Mapping for contained XML types.

schemaMappings

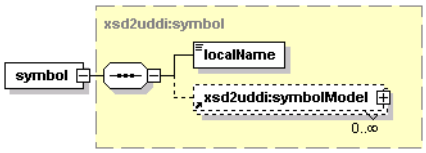


This structure describes a mapping from the contents of a XML Schema Document to UDDI entities. There are two parts. The first part describes possible matches for `xs:imports` specified by the XML Schema Document; the second, individual candidates that may match the XML Schema Document contents. The candidate structure then contains a mapping of the XML Schema Document onto the particular candidate tModel and the related UDDI entities.

Arguments

- `xsd2uddi:schemaImports` - mapping for referenced (imported) XML Schema Documents.
- `xsd2uddi:schemaCandidate` - an individual mapping candidate.

symbol

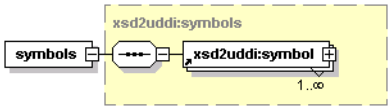


This structure holds mapping of an individual symbol (XSD element and type) to the registry.

Arguments

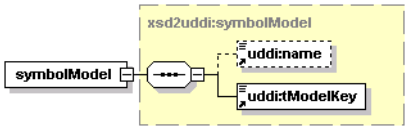
- `localName` - Local name of the mapped symbol.
- `xsd2uddi:symbolModel` - The basics of the tModel that represents the symbol.

symbols



A common structure for mapping types and elements.

symbolModel

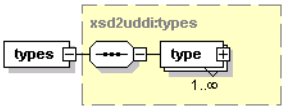


Basic information about a tModel that represents an element or a type declared by the XML Schema Document

Arguments

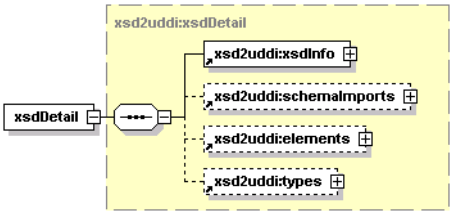
- `uddi:name` - Name of the symbol's tModel. This argument is optional when publishing a XML Schema Document; it is always filled in API responses.
- `uddi:tModelKey` - tModelKey of the symbol's model

types



Mapping of types declared by the XML Schema Document being mapped

xsdDetail

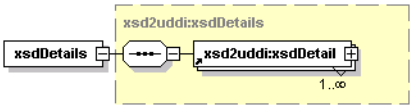


The structure provides detailed information about a specific XML Schema Document, its contents and its references.

Arguments

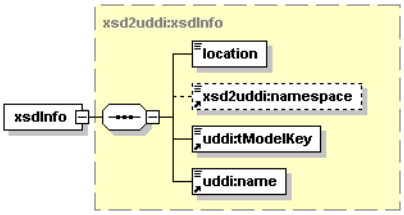
- `xsd2uddi:xsdInfo` - General information about the XML Schema Document itself
- `xsd2uddi:schemaImports` - Information about XML namespaces imported into the XML Schema Document
- `xsd2uddi:elements` - List of elements in the schema
- `xsd2uddi:types` - List of types in the schema

xsdDetails



Details of the XSD

xsdInfo



This structure holds general information about the XML Schema Document.

Arguments

- `location` - The location of the XML Schema Document. This location can be used to retrieve the contents
- `xsd2uddi:namespace` - The URI of the XML namespace defined by the XML Schema Document
- `uddi:tModelKey` - tModel key for the schema's tModel

- `uddi:name` - tModel name for the schema's tModel

xsdResourceList

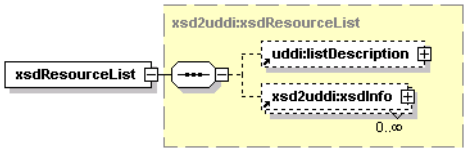


Table 39. Attributes

Name	Required
`truncated`	optional

This structure holds a list of XSDs, returned from a find_xsd call.

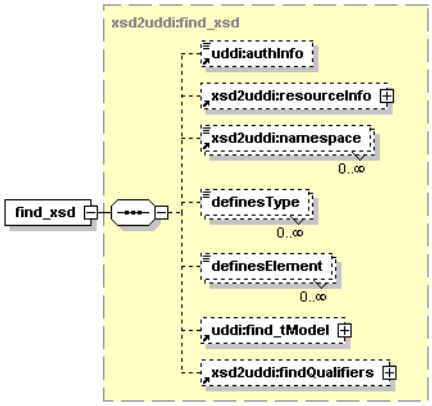
Arguments

- `uddi:listDescription` - holds a list of descriptions as specified in UDDI's API documentation.
- `xsd2uddi:xsdInfo` - holds information about individual registered XSD models.

Operations

find_xsd

Syntax



This operation finds the XML Schema Document. The caller can limit the number of search results to be returned and can iterate through the search results using the `listHead` and `maxRows` arguments.

The name and URI lists passed as the input search criteria may use wildcard characters provided that the `approximateMatch` findQualifier is present. If the `ownEntities` findQualifier is used, the operation returns only entities owned by the authenticated user. Other entities are not returned even though they match the other search criteria.

Table 40. Attributes

Name	Required
`listHead`	optional
`maxRows`	optional

Arguments

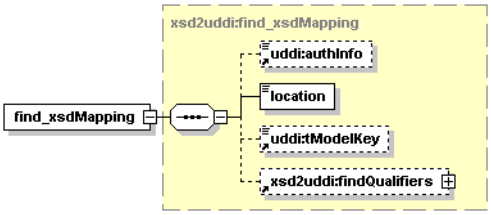
- `uddi:authInfo` - This optional argument is the string representation of the uddi:authToken.
- `xsd2uddi:resourceInfo` - URI location of the published XML Schema Document. The registry does not read from the location, it is used as a search criteria for the current UDDI contents only.
- `xsd2uddi:namespace` - Allows to search by the namespace defined by a XML Schema Document. Contains a list of XML namespace URIs. An XML Schema Document satisfies this condition if its `targetNamespace` attribute is among the URIs.
- `definesType` - Allows the user to search by defined type. Contains a list of type names. An XML Schema Document satisfies this condition if it defines a global type with a name passed in the list.
- `definesElement` - The returned schemas must define the named element.
- `uddi:find_tModel` - An argument used for a more detailed description of search criteria. For more information, see uddi:find_tModel [http://uddi.org/pubs/uddi_v3.htm#_Toc53709284]. These criteria are combined with the other criteria specified by the `find_xsd` structure. In the case of a conflict, the criteria in `find_xsd` take precedence.

Returns

This API call returns thexsdResourceList on success. If the caller specifies the `maxRows` attribute, the returned `xsdResourceList` will contain, at most, that many results. Note that the search may yield a tModel, which does not entirely comply with the XSD-to-UDDI mapping specification, such as when the tModel information is altered manually. In these cases, an attempt to use get_xsdDetail on such a tModel will produce an exception.

find_xsdMapping

Syntax



This operation finds a suitable mapping for contents of the given XML Schema Document. The operation downloads and parses the XML Schema Document at the given location, and matches the contents against the information already published in the registry. It will produce zero or more possible mappings for the given XML Schema Document.

The caller may request that the mapping is attempted only against a specific tModel that represents an XML Schema Document. In that case, only one mapping will be returned.

If the document at the specified location, or one of its dependencies (for example, schemas for XML namespaces which the document imports) are not accessible to the registry, an exception will be raised. If the document is not an XML schema or contains errors, the operation will throw an exception.

Arguments

- `uddi:authInfo` - (Optional) - authentication
- `xsd2uddi:resourceInfo` - The XSD identification (location)

- `uddi:tModelKey` - (Optional), the proposed schema tModel whose contents should be matched. If set, only published contents of that XML Schema Document will be considered for mapping.

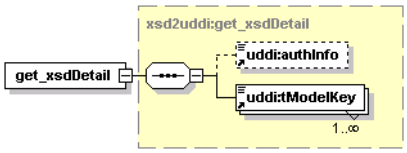
Returns

This API call returns `xsd2uddi:schemaMapping` upon success. The structure contains possible matches for the XML Schema Document at the specified location, which are already stored in the UDDI. There are also possible matches for the XML Schema Documents for XML namespaces imported into the main XML Schema Document.

The call will fail if it cannot access the XML Schema Document or one of its dependencies.

get_xsdDetail

Syntax



Gets the detail about a published XML Schema Document tModels.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - Required uddiKey value representing an existing XML Schema Document tModel.

Returns

This API call returns the xsd2uddi:xsdDetails.

If the passed tModelKey does not exist, or identifies a tModel that does not represent an XML Schema Document, an exception is raised.

publish_xsd

Syntax

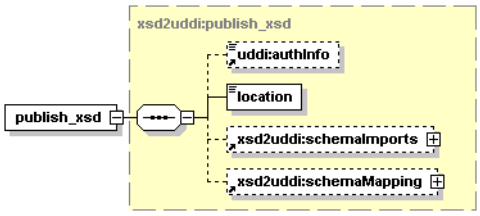


Table 41. Attributes

Name	Required
`importPolicy`	optional
`contentPolicy`	optional
`publishingMethod`	optional
`contentPublishingMethod`	optional
`importPublishingMethod`	optional

Request to publish XML schema information to the registry. The user may pass only minimal information and rely on the matching algorithm used internally to find the appropriate mapping for the published XML Schema Document.

Using the `importPolicy` and `contentPolicy`, the caller may limit the scope of the published data. By the`publishing-Method`, `contentPublishingMethod` and `importPublishingMethod` attributes, the caller may specify the default behavior for publishing - whether an existing UDDI entity is reused and possibly updated, or a new UDDI entity is created, or the particular kind of information is ignored at all.

The registry will need to read the XML Schema Document during the call as well as any resources referenced (imported) by it. If a XML Schema Document or a referenced resource is not available, the operation will fail.

If the caller does not specify a mapping for some element, type, or XML namespace import and there will be more possible matching UDDI entities, the call will fail because the mapping of that XML schema entity is considered ambiguous. It is the responsibility of the caller to provide specific directions for the publishing in such cases.

If the `schemaMapping` entry for a type, an element or an import specifies a publishingMethod `reuse`, the API will try to find a suitable UDDI entity. If such an entity is not found, the API will create one. If the caller provides a specific tModelKey with the `reuse` publishingMethod, the tModelKey must exist and that tModel will be updated with the element, type or import data.

If the `schemaMapping` entry for a type, an element or an import specifies a publishing method `create`, the API will always create a new UDDI entity for that XML Schema Document piece. If the caller specifies the tModelKey in the schemaMapping entry, the new UDDI entity will be assigned that tModelKey. The caller may specify a name for the new tModel, too.

If the caller specifies `ignore` publishing method for an element, a type or an import, that particular XML Schema Document piece will not be published at all. If the publishing operation updates an existing entity in the registry that contains a reference to the element, type or an import, the reference will be purged. When an element or type is ignored, the matching UDDI entity will be deleted from the registry as well by the publish operation.

Arguments

- `uddi:authInfo` - (Optional) - authentication
- `location` - XSD identification (location).
- `xsd2uddi:schemaImports` - Mapping for referenced (imported) XML Schema Documents
- `xsd2uddi:schemaMapping` - (Optional) customized mapping for the schema contents and references
- `importPolicy` - attribute specifying which imports will be published
- `contentPolicy` - attribute specifying which content will be published

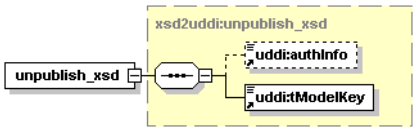
- `publishingMethod` - attribute specifying the default publishing method for the contents (elements, types) declared by the schema; default = update
- `contentPublishingMethod` - The default publishing method for elements and types (ignore, create, reuse); default = reuse. This publishing method will be used for all elements or types unless the `schemaMapping` contains an entry for the element or type that provides a different value.
- `contentPublishingMethod` - The default publishing method for imports (ignore, create, reuse); default = reuse. This publishing method will be used for all imported XML namespaces unless the `schemaMapping` contains an entry for the XML namespace that provides a different value.

Returns

This API call returns the `xsdDetail` with the published XML Schema Document information on success.

unpublish_xsd

Syntax



Unpublish the XML Schema Document. The operation checks whether the XML Schema Document is referenced from other data published in the UDDI. If so, the operation fails as the semantics of the referencing data might break if the XML Schema Document information is removed from the UDDI registry.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - tModelKey of the tModel that represents the XML Schema Document.

Returns

This API call returns the `xsdDetail` on success.

WSDL

xsd2uddi_v3.wsdl [${doc.web.root}/wsdl/xsd2uddi_v3.wsdl]

API Endpoint

You can find the XSD2UDDI API endpoint at `http://<host name>:<port>/uddi/xsd2uddi`.

Java

2.2.11. XSLT Publishing

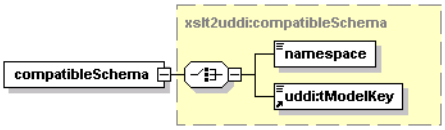
Systinet XSLT-to-UDDI mapping enables the automatic publishing of XSLT into UDDI and enables precise, flexible UDDI queries based on specific XSL Transformation Documents.

The UDDI stores information about the input and output formats accepted and produced by the XSL transformation and about other XSL stylesheets imported into the transformation. The input format is defined by an XML Schema. The output format may be defined by an XML Schema (its representing tModel), or it may be typed by a general tModel that represents the user's definition of the output. The UDDI also stores the output method used by the stylesheet: html, xml, text.

The XSLT Publishing API allows to search for the stylesheets by types of their input and output in order to locate a XSL suitable for processing a particular document, or a XSL that may produce some desired format.

Data Structures

compatibleSchema

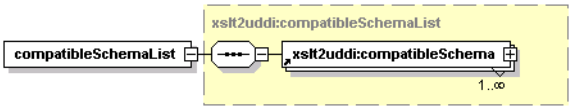


A query for the input format of the style sheet. Selects those style sheets, which accept the specified schema. The schema can be given either using a namespace URI or directly using the tModelKey of the XML Schema tModel representation in the UDDI.

Arguments

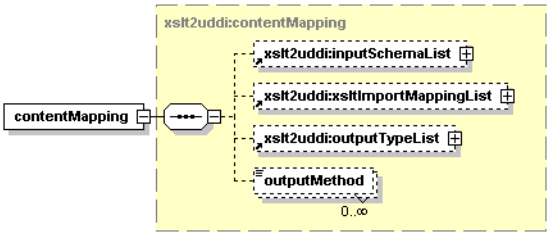
- `namespace` - the URI of the XML namespace defined by the schema
- uddi:tModelKey - tModelKey of the tModel that represents the XML Schema

compatibleSchemaList



This structure holds a list of compatibleSchemas.

contentMapping

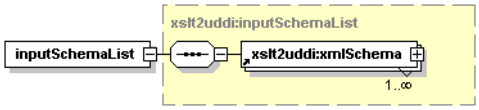


Describes how the contents of the XSLT transformation are mapped to the entities published in the registry.

Arguments

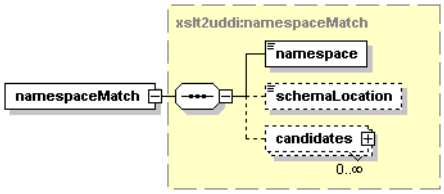
- xslt2uddi:inputSchemaList
- xslt2uddi:xsltImportMappingList
- xslt2uddi:outputTypeList
- outputMethod - One or more output methods, as defined by the XSLT specification. The default value substituted by the API when no output method is given is "xml".

inputSchemaList



List of the XSL transformation's information structures and references to input schemas.

namespaceMatch

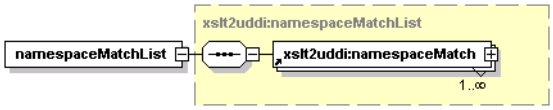


This structure represents matches found in the UDDI registry for a specific XML namespace UI referenced by the XSL Transformation Document.

Arguments

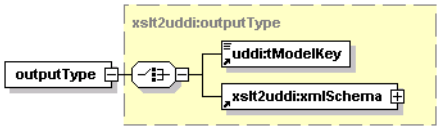
- `namespace` - XML namespace URI referenced in the XSL transformation
- schemaLocation - explicit location of the XML schema for the namespace. Optional.
- `candidates` - possible mappings to tModels. For more information, please see xslt2uddi:tModelRef.

namespaceMatchList



This structure holds a list of namespaceMatches.

outputType

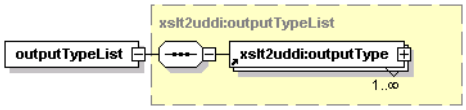


The types of resources the XSL transformation may produce. Currently only xml is supported, typed by a XML Schema tModel

Arguments

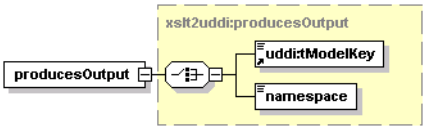
- uddi:tModelKey - tModel that represents the formal description of the output format
- xslt2uddi:xmlSchema

outputTypeList



List of descriptions of output formats the style sheet can produce.

producesOutput

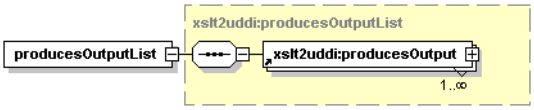


Query parameter that selects results based on the output produced by the XSL Transformation Document

Arguments

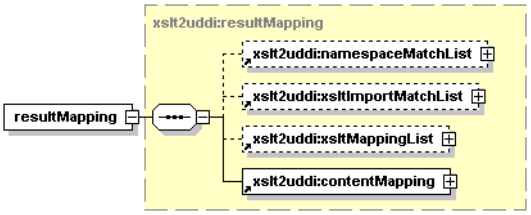
- uddi:tModelKey - key of a tModel that represents the formal description of the output format. Currently only tModels that represent XML schemas are supported
- namespace - the namespace URI of the XML namespace that defines output elements produced by the XSL Transformation Document

producesOutputList



List of output format query parameters

resultMapping



This structure holds the result of find_xsltMapping. It describes possible mappings for XML namespaces (their schemas) and mappings for stylesheet imported to the XSLT passed to the request. Finally, the tModels that match the XSL Transformation Document itself are reported in the mapping. For each of tModel that matches the mapped %xslt;, there's a suggested mapping of the XSLT Document contents onto the particular tModel and its related data.

The nested `contentMapping` structure is a suggestion how to map the XSL Transformation Document on a new tModel, rather than on some existing one. Mappings to already existing tModels are described in `xsltMappingList` nested structure.

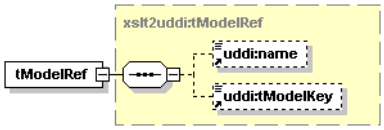
It may happen, that a XML namespace URI or an importer XSL Transformation Document has several mappings into the UDDI. In such cases, the entries in the `xsltMappingList` or the `contentMapping` may contain no tModelKeys as an indication that the mapping algorithm could not decide the mapping. It is up to the caller to resolve such ambiguities.

Arguments

- xslt2uddi:namespaceMatchList
- xslt2uddi:xsltImportMatchList
- xslt2uddi:xsltMappingList

- xslt2uddi:contentMapping

tModelRef

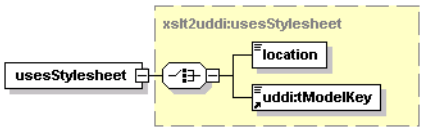


This structure holds a reference to a tModel representing an XML Schema document, or XML style sheet document.

Arguments

- `uddi:name` - name of the tModel. This name is always present in API response messages.
- `uddi:tModelKey` - tModelKey that represents an XML schema or XSLT document.

usesStylesheet

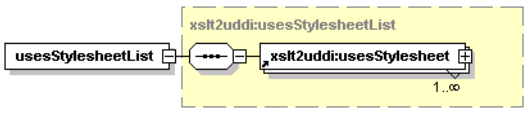


This structure is used in find_xslt queries.

Arguments

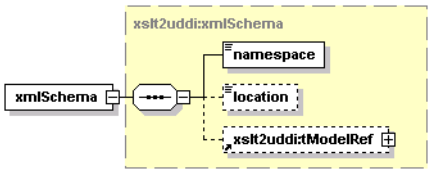
- `location` - location of the XSLT document.
- `uddi:tModelKey` - tModelKey of the tModel that represents the XSLT document.

usesStylesheetList



This structure holds a list of usesStylesheets.

xmlSchema

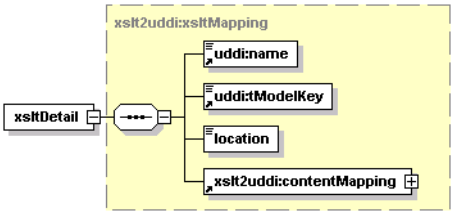


Description of a referenced XML Schema

Arguments

- namespace - The namespace referenced from the XSL Transformation Document
- location - The explicit location of the XML Schema for the namespace, if given in the XSLT. Optional.
- xslt2uddi:tModelRef

xsltDetail

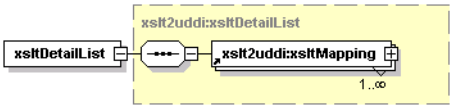


This structure holds the representation of an XSLT document in the UDDI registry.

Arguments

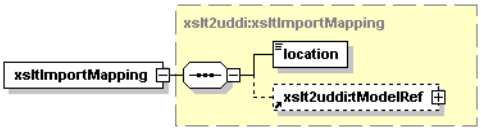
- `uddi:name` - name of the XSL Transformation Document tModel .
- uddi:tModelKey - the tModelKey of the tModel that represents the XSL Transformation Document
- location - the URI of the XSL Transformation Document document
- xslt2uddi:contentMapping

xsltDetailList



This structure represents a list of xsltDetails.

xsltImportMapping



This structure holds a mapping XSL Transformation Document imported to UDDI entities.

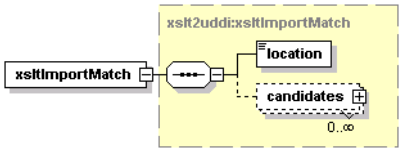
Arguments

- `location` - location of the imported XSL Transformation Document.
- xslt2uddi:tModelRef - references to tModels that match the imported XSL Transformation Document.

xsltImportMappingList

This structure represents a list of xsltImportMappings.

xsltimportMatch



This structure represents a matching between imported XSL Transformation Documents and UDDI entities.

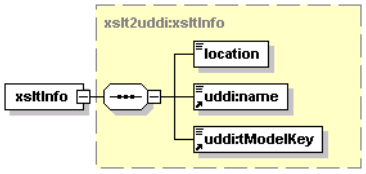
Arguments

- `location` - location of the imported XSL Transformation Document.
- `candidates` - possible mappings to UDDI tModels. See xslt2uddi:tModelRef

xsltImportMatchList

This structure holds a list of xsltImportMatches.

xsltInfo

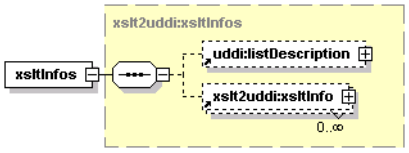


This structure represents an item from the list returned by find_xslt operations.

Arguments

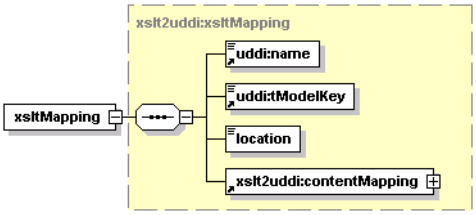
- `location` - location of the XSL Transformation Document
- `uddi:name` - name of the XSL Transformation Document
- `uddi:tModelKey` - the key of tModel that represents the XSL Transformation Document.

xsltInfos



This structure holds a list of xsltInfos.

xsltMapping

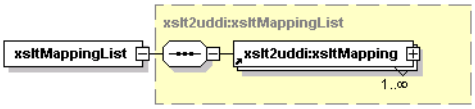


This structure describes the mapping of an XSL Transformation Document.

Arguments

- `uddi:name` - name for the XSLT tModel
- `uddi:tModelKey` - tModelKey of the target tModel
- `location` - location of the XSL Transformation Document.
- xslt2uddi:contentMapping

xsltMappingList



This structure represents a list of xslMappings

Operations

find_xslt

Syntax

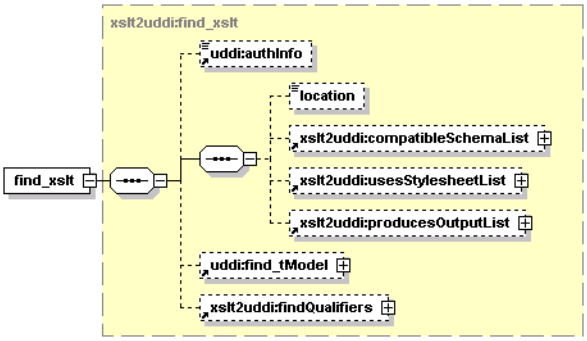


Table 42. Attributes

Name	Required
`listHead`	optional
`maxRows`	optional

This operation finds the XSLT tModel that satisfies the search criteria. The caller may limit the number of results or page through the list of results using `listHead` and `maxRows` attributes. They have the same semantics as in find_tModel in the UDDI Inquiry API.

The name and URI lists passed as the input search criteria may use wildcard characters provided that the `approximateMatch` findQualifier is present. If the `ownEntities` findQualifier is used, the operation returns only entities owned by the authenticated user. Other entities are not returned even though they match the other search criteria.

Arguments

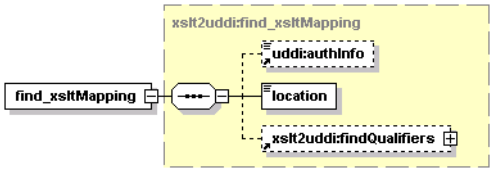
- `uddi:authInfo` - This optional argument is the string representation of the uddi:authToken.
- `location` - location of the XSL Transformation Document.
- xslt2uddi:compatibleSchemaList
- xslt2uddi:usesStylesheetList
- xslt2uddi:producesOutputList
- uddi:find_tModel - a generic query parameter to further restrict the search using user-defined criteria
- xslt2uddi:findQualifiers - see find qualifiers

Returns

This API call returns the a list of xsltInfos on success.

find_xsltMapping

Syntax



This operation finds a suitable mapping for contents of the given XSL Transformation Document.

The mapping algorithm tries not to report ambiguous mapping unless necessary. If some reference to a XML namespace or an imported XSL Transformation Document is ambiguous, the mapping algorithm will consider the already published data and suggest the tModelKey used by the existing tModel that represents the XSL Transformation Document. So in other words, if there is an XSL Transformation Document tModel already published, that references a specific tModelKey for a XML namespace, that tModelKey will be reported in the XsltMappingList even though there are more possible matching entities for the XML namespace.

Arguments

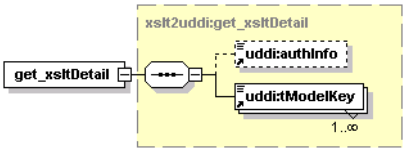
- `uddi:authInfo` - authentication
- `location` - location of the XSL Transformation Document
- xslt2uddi:findQualifiers - see find qualifiers

Returns

This API call returns xslt2uddi:resultMapping upon success.

get_xsltDetail

Syntax



This operation gets the detail about published XSLT tModels.

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - required key value representing an existing XSLT tModel.

Returns

This API call returns the xslt2uddi:xsltDetailList..

publish_xslt

Syntax

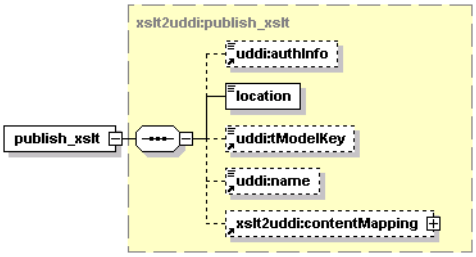


Table 43. Attributes

Name	Required
`publishingMethod`	optional
`schemaMethod`	optional
`importMethod`	optional

A request to publish XSLT information to the UDDI registry.

The `publishingMethod` defines how the XSL Transformation Document will be published in the UDDI registry. The `schemaMethod` and `importMethod` attributes define the defaults for publishing XML schema references, or references to imported XSL Transformation Documents, respectively. It is possible to override those defaults in entries of the passed `contentMapping`.

The registry will need to read the XSL Transformation Document document being published. If the XSLT is not available to the UDDI registry, the operation will fail.

If the caller does not specify a mapping for some referenced XML namespace URI, or an imported XSL Transformation Document, and there will be more possible matching UDDI entities, the call will fail because the mapping is considered ambiguous. It is the responsibility of the caller to provide specific directions for the publishing in such cases.

If a mapping entry specifies "create" as its publishing method, a new entity will be created to represent the particular part of the XSL Transformation Document. In this case the tModelKey of the mapping, if present, is used to provide a publisher-assigned key to the new entity.

If a mapping entry specifies "ignore" publishing method, the information is not propagated into the UDDI registry at all. When updating an existing XSL Transformation Document tModel, such information are purged. So when a XML namespace is "ignored", the publishing operation will remove the association between the XSL Transformation Document and the ignored XML Schema. Ignoring an element or type will delete the representing tModel entity from the UDDI.

Arguments

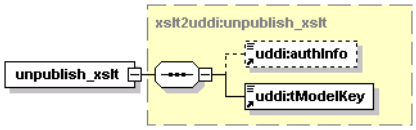
- `uddi:authInfo` - (Optional) - authentication
- `location` - XSLT identification (location) of the XSL Transformation Document.
- `uddi:tModelKey` - the tModelKey to be updated
- uddi:name - the name of the tModel, optional
- xslt2uddi:contentMapping
- `publishingMethod` - The publishing method for the XSLT itself (create, update). (default = update).
- `schemaMethod` - The publishing method for the referenced schemas (create, reuse, ignore). (Default = reuse).

Returns

This API call returns the xsltDetail on success.

unpublish_xslt

Syntax



Unpublish the XSL Transformation Document.

The contents of the UDDI Registry are checked whether there are referencies to this XSLT representant. If so, the operation fails with a disposition report that clearly shows tModelKeys of the referencing entities. Only references between XSLTs are checked (the uddi:uddi.org:resource:reference taxonomy).

Arguments

- `uddi:authInfo` - This optional argument is the string representation of the `uddi:authToken`.
- `uddi:tModelKey` - tModelKey of the XSLT.

Returns

This API call returns the xsltDetail on success.

WSDL

Xslt2uddi_v3.wsdl [${doc.web.root}/wsdl/xslt2uddi_v3.wsdl]

API Endpoint

You can find the XSLT2UDDI API endpoint at `http://<host name>:<port>/uddi/xslt2uddi`.

Java

2.2.12. Inquiry UI

The Inquiry UI API has been implemented for improving the performance of the Business Service Console. The basic idea is to retrieve data that appear in the Business Service Console using a single API call.

This API contains only one operation get_entityDetail. Its input includes a query specification and an output format:

- The **query specification** comprises one of the standard UDDI v3 API data structures: find_business, find_services, find_binding, find_tModel, get_businessDetail, get_serviceDetail, get_bindingDetail and get_tModelDetail.
- The **output format** defines which data structures will be returned and how they will be pruned.

The operation get_entityDetail returns a list of UDDI data structures. ACLs are also applied to retrieved data.

For example, if you specify the following inquiry:

```
<get_entityDetail xmlns="http://systinet.com/uddi/inquiryUI/6.0">
  <outputFormat>
    <businessEntityMask descriptionIncluded="true" identifierBagIncluded="true"/>
    <businessServiceMask descriptionIncluded="true"/>
  </outputFormat>
  <find_binding serviceKey="uddi:systinet.com:demo:hr:employeesList"
     xmlns="urn:uddi-org:api_v3"/>
</get_entityDetail>
```

You will receive the following output:

```
<entityDetail xmlns="http://systinet.com/uddi/inquiryUI/6.0">
  <businessEntity businessKey="uddi:systinet.com:demo:hr"
        xmlns="urn:uddi-org:api_v3">
    <name>HR</name>
    <description>HR department</description>
    <businessServices>
      <businessService serviceKey="uddi:systinet.com:demo:hr:employeesList"
            businessKey="uddi:systinet.com:demo:hr">
        <name>EmployeeList</name>
        <description>wsdl:type representing service</description>
      </businessService>
    </businessServices>
    <identifierBag>
      <keyedReference tModelKey="uddi:systinet.com:demo:departmentID"
            keyName="department id" keyValue="002"/>
    </identifierBag>
  </businessEntity>
</entityDetail>
```

If there are matching bindingTemplates accessible while associated businessServices are not (because of ACLs), such bindingTemplates will be included in the result in a separate list of bindingTemplates. The same behavior applies to accessible businessServices of inaccessible businessEntities.

Data Structures

The following structures are used by the Systinet Inquiry UI API:

- Section bindingTemplateMask
- Section businessEntityMask
- Section businessServiceMask
- Section contactMask
- Section entityDetail
- Section outputFormat
- Section tModelInstanceInfoMask
- Section tModelMask

bindingTemplateMask

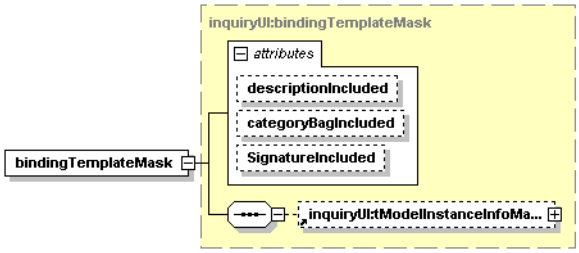


Table 44. Attributes

Attribute	Required
`descriptionIncluded`	No
`categoryBagIncluded`	No
`SignatureIncluded`	No

The `bindingTemplateMask` structure specifies the mask of the binding template of the outputFormat. Optional attributes define which elements will be returned in the entityDetail

businessEntityMask

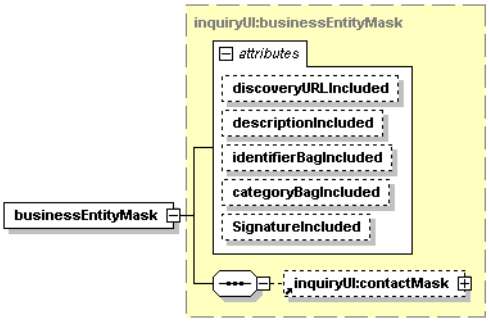


Table 45. Attributes

Attribute	Required
`discoveryURLIncluded`	No
`descriptionIncluded`	No
`identifierBagIncluded`	No
`categoryBagIncluded`	No
`SignatureIncluded`	No

The `businessEntityMask` structure specifies the mask of the business entity of the outputFormat. It also include a contactMask. Optional attributes define which elements will be returned in the entityDetail.

businessServiceMask

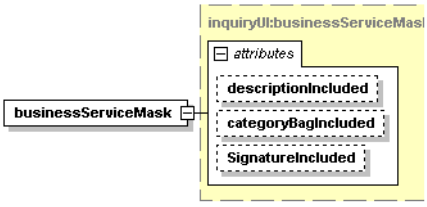
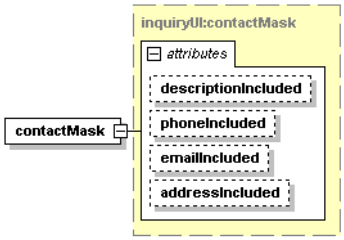


Table 46. Attributes

Attribute	Required
`descriptionIncluded`	No
`categoryBagIncluded`	No
`SignatureIncluded`	No

The `businessServiceMask` structure specifies the mask of the business service of the outputFormat. Optional attributes define which elements will be returned in the entityDetail.

contactMask

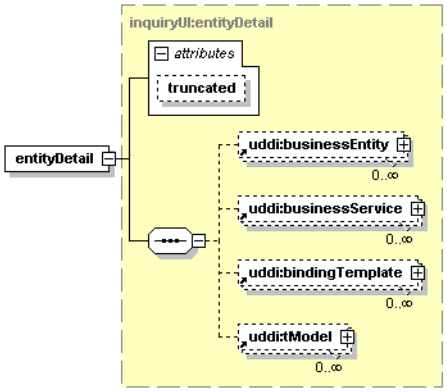


The `contactMask` structure specifies the submask of the business entity mask of the outputFormat. Optional attributes define which elements will be returned in the entityDetail

Table 47. Attributes

Attribute	Required
`descriptionIncluded`	No
`phoneIncluded`	No
`emailIncluded`	No
`addressIncluded`	No

entityDetail

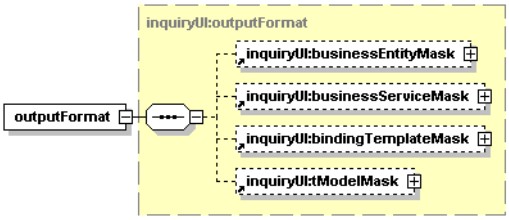


The `entityDetail` structure is returned by the get_entityDetail operation. The attribute `truncated` indicates a truncated result list.

Table 48. Attributes

Attribute	Required
`uddi:truncated`	No

outputFormat

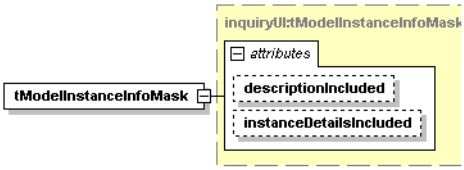


The `outputFormat` is a mask for data to be returned and can prune returned structures. The output format is defined by the following arguments.

Arguments

- inquiryUI:businessEntityMask
- inquiryUI:businessServiceMask
- inquiryUI:bindingTemplateMask
- inquiryUI:tModelMask

tModelInstanceInfoMask

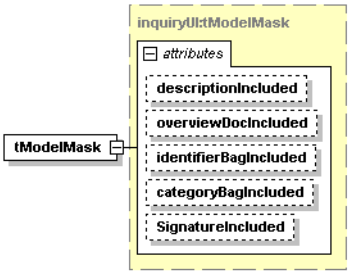


The `tModelInstanceInfoMask` structure specifies the mask of the tModel instance info of the outputFormat. Optional attributes define which elements will be returned in the entityDetail

Table 49. Attributes

Attribute	Required
`descriptionIncluded`	No
`instanceDetailsIncluded`	No

tModelMask



The `tModelMask` structure specifies the mask of the tModel of the outputFormat. Optional attributes define which elements will be returned in the entityDetail

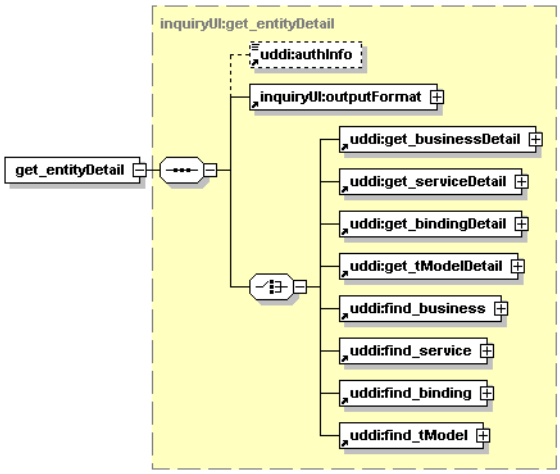
Table 50. Attributes

Attribute	Required
`descriptionIncluded`	No
`overviewDocIncluded`	No
`identifierBagIncluded`	No
`categoryBagIncluded`	No
`SignatureIncluded`	No

Operations

get_entityDetail

This is the core operation of the Inquiry UI API.



Arguments

- uddi:authInfo - This optional argument is an element that contains an authentication token.
- inquiryUI:outputFormat
- uddi:get_businessDetail, uddi:get_bindingDetail, uddi:get_tModelDetail, uddi:find_business, uddi:find_service, uddi:find_binding, uddi:find_tModel - standard UDDI v3 structures.

Returns

Upon successful completion, an entityDetail structure is returned.

WSDL

You can find the WSDL specification in the file inquiryUI.wsdl [${doc.web.root}/wsdl/inquiryUI.wsdl].

API Endpoint

You can find the Inquiry UI API endpoint at `http://<host name>:<port>/uddi/inquiryUI`.

Java

Systinet Java API is generated directly from WSDL. You are encouraged to browse `org.systinet.uddi.client.v3.ui.InquiryUIApi`.

2.2.13. Subscription Ext

The Subscription Extension API has been implemented to allow the user to create subscriptions in the *discovery registry* of the approval process. This means that subscription creation is not subject to the approval process; users can save subscriptions directly to the discovery registry. However, under this API, users are not allowed to save a bindingTemplate for the email address where notifications are sent. The Subscription Extension API allows the user to specify a bindingTemplate in the subscriptionExt structure in the save_subscription operation. This bindingTemplate is saved under the Notification Service Container of the operator's business entity. The Notification Service Container is a businessService with the key `uddi:systinet.com:subscription:notification_service_container`. This API can also be used for "read-only" registry. In that case, users are not allowed to publish their data to the registry. Their subscriptions can be saved with this API.

Data Structures

The following structures are used by the Subscription Extension API:

- Section Notification Service Container
- Section subscriptionExt

Notification Service Container

The `Notification Service Container` is a business service stored under the operator's business entity. It has the key: `uddi:systinet.com:subscription:notification_service_container`. This business service is imported together with other registry pre-deployed data.

subscriptionExt

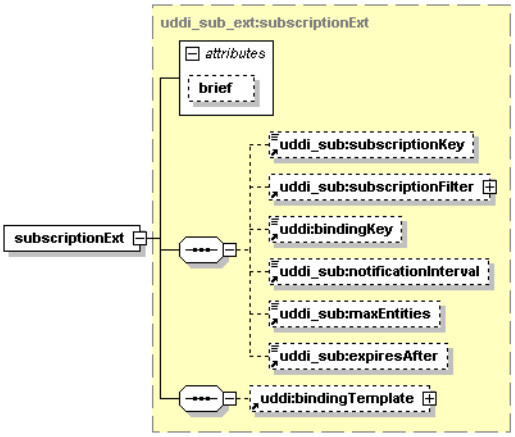


Table 51. Attributes

Attribute	Required
`brief`	No

The `subscriptionExt` structure substitutes the `uddi_sub:subscription` structure in the `save_subscription` structure of the standard UDDI v3 API.

Operations

The following operations extend the standard UDDI v3 API:

- Section save_subscription
- Section delete_subscription

save_subscription

- This operation is used when creating a new subscription. If the bindingTemplate is specified, then the subscription is saved under the caller's user account under the Notification Service Container. The bindingKey is generated by the registry, the other structures of the bindingTemplate remain untouched. The bindingKeys in both the subscription and the bindingTemplate are ignored. The subscription structure returns a bindingKey referencing the saved bindingTemplate, but not the bindingTemplate itself.
- Updating the existing subscription. The algoritm of the standard saving of subscriptions is extended with these steps:
  1. If the subscription refers to a bindingTemplate under the Notification Service Container, then the binding template will be deleted. See delete_subscription
  2. If the bindingTemplate is specified in the subscription, then the bindingTemplate is stored under the Notification Service Container

delete_subscription

If the subscription references a bindingTemplate which is under the Notification Service Container, then the bindingTemplate will be deleted.

WSDL

You can find the WSDL specification in the file uddi_sub_v3_ext.wsdl [${doc.web.root}/wsdl/uddi_sub_v3_ext.wsdl].

API Endpoint

You can find the Statistics API endpoint at `http://<host name>:<port>/uddi/subscriptionExt`.

Java

The Systinet Java API is generated directly from WSDL. You are encouraged to browse `org.systinet.uddi.client.subscription.v3.ext.UDDISubscriptionExtStub`.

2.3. Security APIs

Security APIs cover the following APIs:

- Account API - Systinet Account API is used to query and manage user accounts in BEA AquaLogic Service Registry.
- Group API - Systinet Group API is used to query and manage user groups in BEA AquaLogic Service Registry.
- Permission API - Systinet Permission API is used to query and manage permissions in BEA AquaLogic Service Registry.

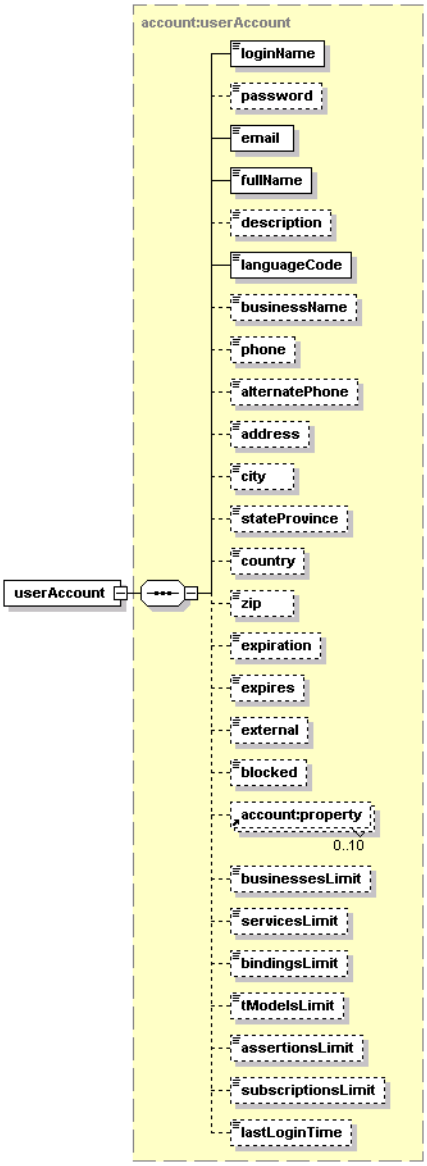
2.3.1. Account

Systinet Account API is used to query and manage user accounts in BEA AquaLogic Service Registry.

Data Structures

The following structures are used by the Systinet Account API:

userAccount



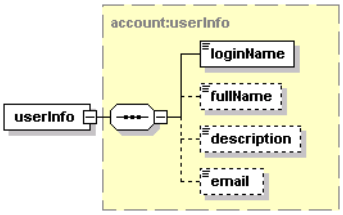
The `userAccount` element is container that holds the attributes of a user account in the BEA AquaLogic Service Registry. The required elements are:

- `loginName`
- `email`
- `fullName`
- `languageCode`

All other elements are optional.

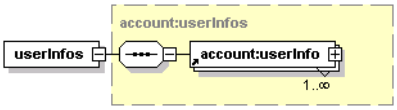
Element	Description
`loginName`	contains the login name of the user account
`password`	contains the password used to authorize the user
`email`	holds the user's email address
`fullName`	holds the user's full name
`description`	use for describing the user or the user's role
`languageCode`	the language the user speaks
`businessName`	name of organization where the user is employed
`phone`	telephone number used to contact the user
`alternatePhone`	second telephone number used to contact the user
`address`	
`city`	
`stateProvince`	
`country`	
`zip`	
`expiration`	may hold the time when the user account expires
`expires`	indicates whether the account may expire over time
`external`	a flag indicating whether the user account is external or stored in the UDDI registry
`blocked`	a flag indicating whether the user is blocked
`account:property`	an unspecified string; its meaning depends on UserStore type
`businessesLimit`	specifies how many business entities the user account may save
`servicesLimit`	specifies maximum number of business services within a single business entity that the user account may own
`bindingsLimit`	specifies how many bindingTemplates the user account may save within a single businessService
`tModelsLimit`	specifies the number of tModels the user account may save
`assertionsLimit`	specifies the number of publisherAssertions the user account may save
`subscriptionsLimit`	specifies the number of subscriptions the user account may save
`lastLoginTime`	contains information regarding when the user last logged into the registry

userInfo



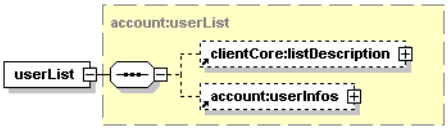
This element serves as a container for short information about single userAccount. It contains the required element `loginName`, and the optional elements `fullName`, `description`, and `email`.

userInfos



This element holds one or more `userInfo` elements.

userList



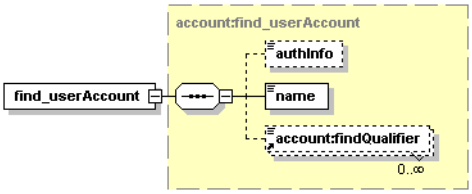
This element contains optional `listDescription` and `userInfos` elements.

Operations

find_userAccount

The find_userAccount API call is used to find user accounts in BEA AquaLogic Service Registry that match given criteria.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `name` - Name to be searched.
- `account:findQualifier` - The collection of findQualifier used to alter default behavior.

Behavior

The following findQualifiers affect behavior of the call:

- The findByLoginName findQualifier (default) is used to specify that user accounts shall be searched by loginName.

- With the findByFullName findQualifier, user accounts are searched by the fullName property.
- If the exactMatch findQualifier is present, an exact match is required.
- The default approximateMatch findQualifier enables SQL wildcard queries.
- If the findBlockedAccount findQualifier is present, only blocked accounts are returned.
- The sortByNameAsc (default) and sortByNameDesc findQualifiers controls the order in which the data is returned.

Returns

This API call returns the userList upon success.

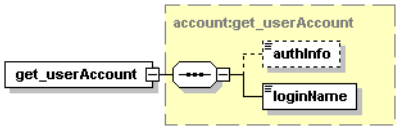
Permissions

This API call requires the API user permission for org.systinet.uddi.account.AccountApi and the action find_userAccount.

get_userAccount

The get_userAccount API call returns userAccount structure of selected user.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `loginName` - This required argument uniquely identifies the user account.

Returns

This API call returns userAccount upon success.

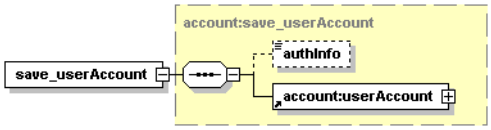
Permissions

This API call requires the API user permission for org.systinet.uddi.account.AccountApi and the action get_userAccount to get user's own account detail and API manager permission for org.systinet.uddi.account.AccountApi and the action get_userAccount to get other users' accounts.

save_userAccount

The save_userAccount API call is used to save or update userAccount in BEA AquaLogic Service Registry. Whether public registration is allowed or not depends on the BEA AquaLogic Service Registry configuration. It may be also configured to block registered account until it is enabled by code sent by email.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `account:userAccount` - The user account to be saved.

Returns

This API call returns userAccount upon success.

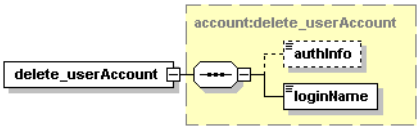
Permissions

This API call requires the API user permission for org.systinet.uddi.account.AccountApi and the action save_userAccount to save user's own account or register new account and API manager permission for org.systinet.uddi.account.AccountApi and the action save_userAccount to save other users' accounts.

delete_userAccount

The delete_userAccount API call causes selected user account to be removed from BEA AquaLogic Service Registry.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `loginName` - This required argument uniquely identifies the user account.

Returns

This API call returns UserAccount upon success.

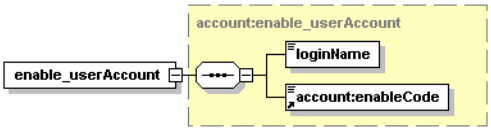
Permissions

This API call requires the API user permission for org.systinet.uddi.account.AccountApi and the action delete_userAccount to delete user's own account and API manager permission for org.systinet.uddi.account.AccountApi and the action delete_userAccount to delete other users' accounts.

enable_userAccount

The enable_userAccount API call is used to activate user account identified by loginName argument in BEA AquaLogic Service Registry.

Syntax



Arguments

- `loginName` - This required argument uniquely identifies the user account.

- `account:enableCode` - Confirmation string.

WSDL

You can find the WSDL specification in the file account.wsdl [${doc.web.root}/wsdl/account.wsdl].

API Endpoint

You can find the Account API endpoint at `http://<host name>:<port>/uddi/account` .

Java

The Systinet Java API is generated from Account WSDL. You are encouraged to browse `org.systinet.uddi.account.AccountApi` and to read and try Account demos.

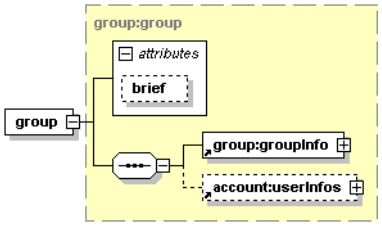
2.3.2. Group

Systinet Group API is used to query and manage user groups in BEA AquaLogic Service Registry.

Data Structures

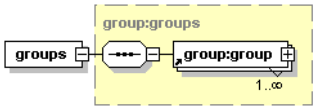
The following structures are used by the Systinet Group API:

group



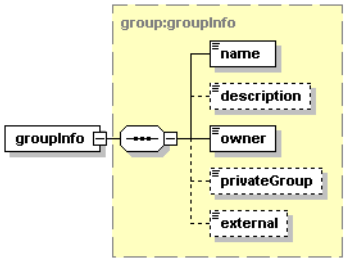
This element serves as a container for `groupInfo` and `userInfos` structures.

groups



This element serves as a container for one or more `group` structures.

groupInfo

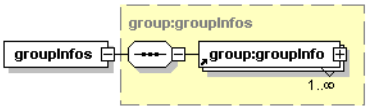


This element contains information about one user group:

- The required `name` element holds the name of the group.

- The optional `description` element is used to describe group and its usage.
- The `owner` element contains the loginName of the user who created this group.
- The `privateGroup` element indicates whether the group is public or private.
- The `external` element indicates whether the group is external (For example, in LDAP) or not.

groupInfos



This element serves as a container for one or more `groupInfo` elements.

groupList

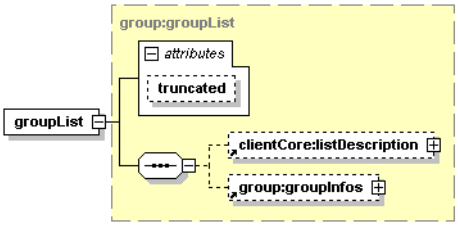


Table 52. Attributes

Attribute	Required
`truncated`	No

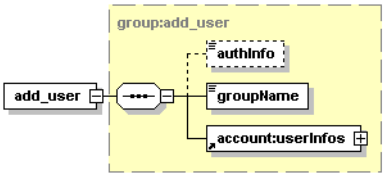
This structure server as a container for optional `listDescription` and optional `groupInfos` structures. The `truncated` attribute indicates whether the list of groupInfos is truncated.

Operations

add_user

The add_user API call is used to add a user to a user group.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `groupName` - the group to which the user will be added.
- `account:userInfos` - user that will be added to the group.

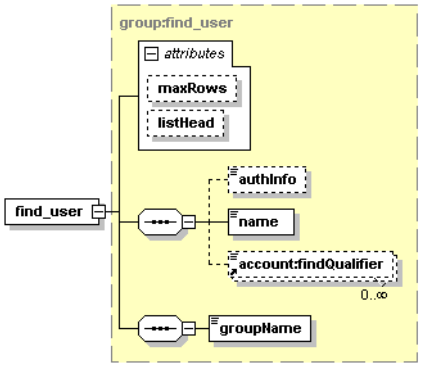
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action add_user.

find_user

The find_user API call is used to find user within the user group.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `name` - login name of the user
- `account:findQualifier` - find qualifier
- `groupName` - the group in which the user will be searched.

Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action find_user.

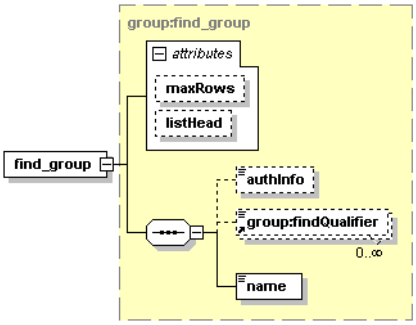
Returns

Upon successful completion, the UserList structure is returned.

find_group

The find_group API call is used to search groups in BEA AquaLogic Service Registry.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `group:findQualifier` - The collection of findQualifier used to alter default behavior.
- `name` - The required value contains name of the group to be searched.

Behavior

The following findQualifiers affect behavior of the call. The exactMatch findQualifier causes that exact match on group name is required, while default approximateMatch findQualifier enables SQL wildcard query. The findPrivateGroups findQualifier enables search between private groups, findPublicGroups enables search between public groups and find-MyGroups will cause the search to be performed only between groups owned by the user who executed this call. The sortByNameAsc and sortByNameDesc findQualifiers controls order, in which the data is returned.

If no findQualifier is defined, default findQualifier set contains approximateMatch, findPrivateGroups, findPublicGroups and sortByNameAsc findQualifiers.

Returns

Upon successful completion, the groupList structure is returned.

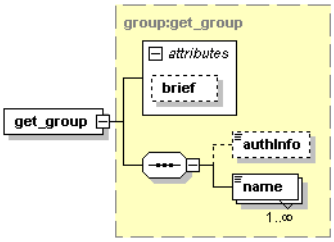
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action find_group.

get_group

The get_group API call is used to get details for one or more groups in BEA AquaLogic Service Registry.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `name` - The required value contains name of the group to be returned.
- `brief` - if you set this attribute, the result will not contain members of the group. Setting the attribute is useful when working with large groups with thousands of members.

Returns

Upon successful completion, the groups structure is returned.

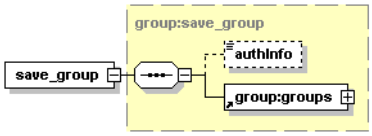
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action get_group. The user permission is needed to get user's own groups, the manager permission is required to get other users' groups.

save_group

The save_group API call is used to save collection of groups to BEA AquaLogic Service Registry.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `group:groups` - The groups to be saved.

Returns

Upon successful completion, the groups structure is returned.

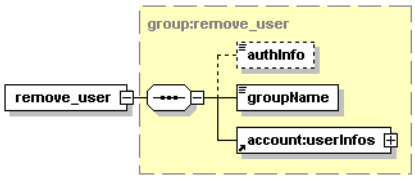
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action save_group. The user permission is needed to save user's own groups, the manager permission is required to update other users' groups.

remove_user

The remove_user API call removes user from the group.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `name` - login name of the user
- `groupName` - the group from which the user will be removed

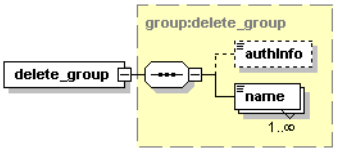
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action remove_user.

delete_group

The delete_group API call causes that groups identified by their names will be removed from BEA AquaLogic Service Registry.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.
- `name` - The required value contains names of the groups to be deleted.

Returns

Upon successful completion, the groups structure is returned.

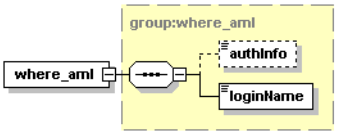
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action delete_group. The user permission is needed to delete user's own groups, the manager permission is required to delete other users' groups.

where_amI

The where_amI API call is there to return list of groups where the user executing this call is member. The call returns both private and public groups.

Syntax



Arguments

- `authInfo` - This optional argument is an element that contains an authentication token.

- `loginName` - This required argument uniquely identifies the user account.

Returns

Upon successful completion, the groupList structure is returned.

Permissions

This API call requires API user or manager permission for `org.systinet.uddi.client.group.GroupApi` and the action where_amI. The user permission is needed to get groups for the user himself, the manager permission is required to get groups for other user.

WSDL

You can find the WSDL specification in the file group.wsdl [${doc.web.root}/wsdl/group.wsdl].

API Endpoint

You can find the Group API endpoint at `http://<host name>:<port>/uddi/group`.

Java

The Systinet Java API is generated from Group WSDL. You are encouraged to browse `org.systinet.uddi.group.GroupApi` and to read and try Group demos.

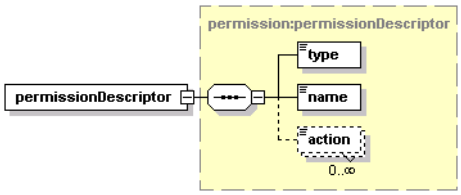
2.3.3. Permission

The Systinet Permission API is used to query and manage permissions in BEA AquaLogic Service Registry.

Data Structures

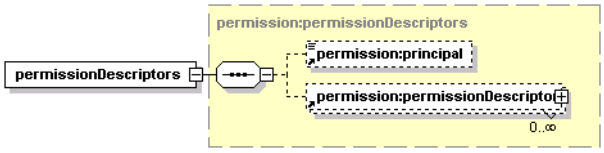
The following structures are used by the Systinet Permission API:

permissionDescriptor



This structure serves as a container for one `permission` and its actions. The `type` element contains the type of the permission. The `name` element contains the permission's name. Optional `action` elements are used to provide finer granularity to the permission and contain individual actions of this permission.

permissionDescriptors



This structure holds an optional `principal` element and zero or more `permissionDescriptor` structures.

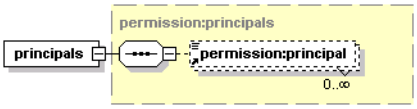
permissionDetail

This structure is a container for zero or more `permissionDescriptors` structures.

principal

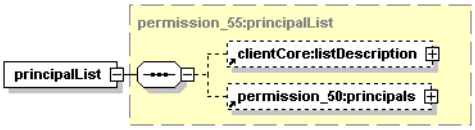
This element contains the optional attribute`principalType`, which may be assigned to a user or group. The element's text contains the loginName of the user, or the group name, depending on the principalType value.

principals



This structure serves as a container for zero or more `principal` elements.

principalList



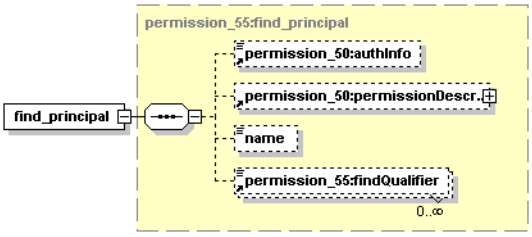
This structure serves as a list principals returned from the operation find_principal.

Operations

find_principal

This operation is used to find principals, it replaces the deprecared operation who_hasPermission .

Syntax



Arguments

- `permission:authInfo` - This optional argument is an element that contains an authentication token.
- `permissionDescriptor`
- `name` - name of the principal
- `findQualifier`

Returns

Upon successful completion, the principalList structure is returned.

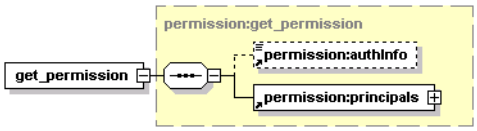
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.permission.PermissionApi` and the action get_permission. The user permission is needed to get permissions for the user himself, the manager permission is required to get permissions for other users.

get_permission

The get_permission API call is used to get permissions in BEA AquaLogic Service Registry, that have been assigned to users or groups identified by the principal's structure.

Syntax



Arguments

- `permission:authInfo` - This optional argument is an element that contains an authentication token.
- `permission:principals` - This mandatory structure contains list of users or groups to be searched.

Returns

Upon successful completion, the permissionDetail structure is returned.

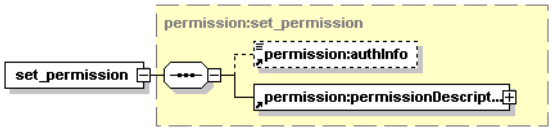
Permissions

This API call requires API user or manager permission for `org.systinet.uddi.permission.PermissionApi` and the action get_permission. The user permission is needed to get permissions for the user himself, the manager permission is required to get permissions for other users.

set_permission

The set_permission API call serves to set permissions in BEA AquaLogic Service Registry. Existing permissions for users or groups referenced in permissionDescriptors are overwritten by this call.

Syntax



Arguments

- `permission:authInfo` - This optional argument is an element that contains an authentication token.
- `permission:permissionDescriptors` - This mandatory structure holds permissions to be set.

Permissions

This API call requires API manager permission for `org.systinet.uddi.permission.PermissionApi` and the action set_permission.

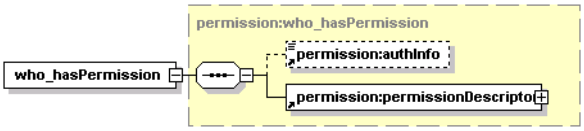
who_hasPermission

Important

The `who_hasPermission` operation is deprecated. We recommend to use the operation `find_principal` instead.

The who_hasPermission API call is used to find out which users or groups have the specified permissions.

Syntax



Arguments

- `permission:authInfo` - This optional argument is an element that contains an authentication token.
- `permission:permissionDescriptor` - This argument contains a description of permissions to be searched.

Returns

Upon successful completion, the principals structure is returned.

Permissions

This API call requires API manager permission for `org.systinet.uddi.permission.PermissionApi` and the action who_hasPermission.

WSDL

You can find the WSDL specification in the file permission.wsdl [${doc.web.root}/wsdl/permission.wsdl].

API Endpoint

You can find the Permission API endpoint at `http://<host name>:<port>/uddi/permission`.

Java

The Systinet Java API is generated from Permission WSDL. You are encouraged to browse its `org.systinet.uddi.permission.PermissionApi` and to read and try the Permission demos.

2.4. Registry Client

This section describes how to prepare your own client distribution. A client created this way allows you to access the BEA AquaLogic Service Registry API through a SOAP interface.

2.4.1. Client Package

Note

`CLIENT_HOME` refers to the directory in which the BEA AquaLogic Service Registry Client distribution will be created.

`REGISTRY_HOME` refers to the directory in which BEA AquaLogic Service Registry is installed

To create a client application distribution follow these steps:

1. Make sure BEA AquaLogic Service Registry is successfully installed.
2. In the `CLIENT_HOME` directory, create a subdirectory named `lib`.

   Copy the following files from `REGISTRY_HOME/lib` to `CLIENT_HOME/lib`

```
activation.jar
builtin-serialization.jar
core_services_client.jar
jaas.jar
jaxm.jar
jaxrpc.jar
jetty.jar
runner.jar
saaj.jar
security-ng.jar
security2-ng.jar
security_providers_client.jar
wasp.jar
wsdl_api.jar
xercesImpl.jar
xml-apis.jar
xmlParserApis.jar
```

3. In the `CLIENT_HOME` directory, create a subdirectory named `dist`.

   Copy the following files from `REGISTRY/dist` to `CLIENT_HOME/dist`:

```
account_client.jar
admin_utils_client.jar
approval_client_v3.jar
approval_content_checker_client_v3.jar
approval_management_client.jar
approval_production_client_v3.jar
category_client_v3.jar
configurator_client.jar
configurator_cluster_client.jar
group_client.jar
permission_client.jar
replication_client_v3.jar
statistics_client.jar
```

```
taxonomy_client_v3.jar
taxonomy_client_v31.jar
transformer_kr_client.jar
uddiclient_api_ext.jar
uddiclient_api_v1.jar
uddiclient_api_v2.jar
uddiclient_api_v3.jar
uddiclient_api_v3_ext.jar
uddiclient_core.jar
uddiclient_custody_v3.jar
uddiclient_subscription_listener_v3.jar
uddiclient_subscription_v3.jar
uddiclient_validate_values_v1.jar
uddiclient_validate_values_v2.jar
uddiclient_value_set_caching_v3.jar
uddiclient_value_set_validation_v3.jar
wsdl2uddi_client_v2.jar
wsdl2uddi_client_v3.jar
xml2uddi_client_v3.jar
xsd2uddi_client_v3.jar
xslt2uddi_client_v3.jar
```

4. In the `CLIENT_HOME` directory, create a subdirectory named `conf`. Copy the following files from REGISTRY_HOME/conf to CLIENT_HOME/conf:

```
clientconf.xml
log4j.config
```

Note

If you want to use the https connection in BEA AquaLogic Service Registry, you must import the certificate file into `clientconf.xml` using the `PStoreTool`. This file contains the certificate of the BEA AquaLogic Service Registry installation by default.

Tip

You do not have to copy client files to directories that have specific names (`lib`, `dist`, and `conf`). All client files can be copied to the flat directory `CLIENT_HOME`, for example. If you do this, however, replace `CONF_DIRECTORY`, `DIST_DIRECTORY`, and `LIB_DIRECTORY` with `CLIENT_HOME` in this section's instructions.

2.4.2. JARs on the Client Classpath

For each client package, the associated .jar files must be added to the classpath. These .jar files are listed in the appropriate sections below.

BEA AquaLogic Service Registry Runtime

To enable the BEA AquaLogic Service Registry Runtime client package, add these .jar files to the classpath.

```
activation.jar
```

```
builtin-serialization.jar;
core_services_client.jar;
jaas.jar;
jaxm.jar;
jaxrpc.jar
runner.jar
saaj.jar;
security-ng.jar;
security2-ng.jar;
security_providers_client.jar;
wasp.jar;
wsdl_api.jar
xercesImpl.jar;
xml-apis.jar;
xmlParserApis.jar;
```

UDDI API Client v1

To enable the UDDI API (v1) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.1.2, UDDI Version 1

```
uddiclient_api_v1.jar
uddiclient_core.jar
```

UDDI API Client v2

To enable the UDDI API (v2) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.1.3, UDDI Version 2.

```
uddiclient_api_v2.jar
uddiclient_core.jar
```

UDDI API Client v3

To enable the UDDI API (v3) client package, add these .jar files to the classpath. For more information on this client packages, please see Section 2.1.4, UDDI Version 3.

```
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI API Client v3 ext X

To enable the UDDI API (v3, ext X) client package, add these .jar files to the classpath.

```
uddiclient_api_v3_ext.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

Account Client

To enable the Account client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.3.1, Account.

```
account_client.jar
uddiclient_core.jar
```

Admin Utilities Client

To enable the Admin Utilities client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.5, Administration Utilities.

```
admin_utils_client.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

Approval Client v3

To enable the Approval (v3) client package, add these .jar files to the classpath. For more information on this client package, please seeSection 2.2.4, Approval.

```
approval_client_v3.jar
uddiclient_api_v3.jar
uddiclient_api_v2.jar
uddiclient_core.jar
```

Approval Content Checker Client v3

To enable the v3 Approval Content Checker client package, add these .jar files to the classpath.

```
approval_content_checker_client_v3.jar
uddiclient_core.jar
```

Approval Management Client

To enable the Approval Management client package, add these .jar files to the classpath.

```
approval_management_client.jar
uddiclient_core.jar
```

Category Client v3

To enable the Category (v3) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.3, Category

```
category_client_v3.jar
```

```
uddiclient_api_v3.jar
uddiclient_core.jar
```

Group Client

To enable the Group client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.3.2, Group.

```
group_client.jar
account_client.jar
uddiclient_core.jar
```

Permission Client

To enable the Permission client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.3.3, Permission.

```
permission_client.jar
account_client.jar
uddiclient_core.jar
```

Replication Client v3

To enable the Replication (v3) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.6, Replication.

```
replication_client_v3.jar
uddiclient_core.jar
```

Statistics Client

To enable the Statistics client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.7, Statistics.

```
statistics_client.jar
uddiclient_core.jar
```

Taxonomy Client v3

To enable the v3 Taxonomy client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.2, Taxonomy.

```
taxonomy_client_v3.jar
taxonomy_client_v31.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI Custody Client v3

To enable the v3 UDDI Custody client package, add these .jar files to the classpath. For more information on this client package, please see Section Custody.

```
uddiclient_custody_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI Subscription Client v3

To enable the v3 UDDI Subscription client package, add these .jar files to the classpath. For more information on this client package, please see Section Subscription.

```
uddiclient_subscription_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI Subscription Listener Client v3

To enable the v3 UDDI Subscription Listener client package, add these .jar files to the classpath. For more information on this client package, please see Section Subscription.

```
uddiclient_subscription_listener_v3.jar
uddiclient_subscription_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI Validate Values Client v1

To enable the UDDI Validate Values (v1) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.1, Validation.

```
uddiclient_validate_values_v1.jar
uddiclient_api_v1.jar
uddiclient_core.jar
```

UDDI Validate Values v2

To enable the UDDI Validate Values (v2) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.1, Validation.

```
uddiclient_validate_values_v2.jar
uddiclient_api_v2.jar
uddiclient_core.jar
```

UDDI Value Set Caching Client v3

To enable the UDDI Value Set Caching (v3) client package, add these .jar files to the classpath.

```
uddiclient_value_set_caching_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

UDDI Value Set Validation Client v3

To enable the UDDI Value Set Validation (v3) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.1, Validation.

```
uddiclient_value_set_validation_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

WSDL2UDDI Client v2

To enable the WSDL2UDDI (v2) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.8, WSDL Publishing

```
wsdl2uddi_client_v2.jar
uddiclient_api_v2.jar
uddiclient_core.jar
```

WSDL2UDDI Client v3

To enable the WSDL2UDDI (v3) client package, add these .jar files to the classpath. For more information on this client package, please see Section 2.2.8, WSDL Publishing

```
wsdl2uddi_client_v3.jar
uddiclient_api_v3.jar
uddiclient_core.jar
```

Resources publishing (XML, XSD, XSLT) Client

To enable the client package, add these .jar files to the classpath.

```
uddiclient_api_v3.jar
uddiclient_core.jar
xml2uddi_client_v3.jar
xsd2uddi_client_v3.jar
xslt2uddi_client_v3.jar
```

Classpath Examples

To run your BEA AquaLogic Service Registry client code you must add a config directory, wasp.jar, and client's jars to the classpath.

Note

`CLIENT_HOME=. CONF_DIRECTORY=CLIENT_HOME\conf DIST_DIRECTORY=CLIENT_HOME\dist LIB_DIRECTORY=CLIENT_HOME\lib`

- If you want to use only UDDI Version 3:

```
CONF_DIRECTORY;LIB_DIRECTORY\wasp.jar;DIST_DIRECTORY\uddiclient_api_v3.jar
```

- If you want to use only UDDI Version 3 and UDDI Subscription Version 3:

```
CONF_DIRECTORY;LIB_DIRECTORY\wasp.jar;DIST_DIRECTORY\uddiclient_api_v3.jar%;
DIST_DIRECTORY\uddiclient_subscription_v3.jar
```

- If you want to use only UDDI Version 3, UDDI Subscription Version 3, and Taxonomy:

```
CONF_DIRECTORY;LIB_DIRECTORY\wasp.jar;DIST_DIRECTORY\uddiclient_api_v3.jar%;
DIST_DIRECTORY\uddiclient_subscription_v3.jar;DIST_DIRECTORY\taxonomy_client_v3.jar
```

2.5. Client Authentication

In this section, we will show you how to create a BEA AquaLogic Service Registry client that uses HTTP Basic authentication. The example client will search the registry and publish a business entity to it. We use the demo user from demo data. We will also describe how to enable HTTP Basic authentication on the server side.

Establishing HTTP Basic authentication requires the following steps. They will be described in detail later in this section.

1. To prepare the client application, copy all necessary jar files from the subdirectories of `REGISTRY_HOME`. Compile the client class file with proper jars in the CLASSPATH.
2. Modify the file `REGISTRY_HOME/app/uddi/services/WASP-INF/package.xml` to enable the HttpBasic security provider on service endpoints and enable the HttpBasic interceptor.
3. Restart BEA AquaLogic Service Registry, then run the client application.

Now, we will describe the steps in detail:

1. Create the directory `CLIENT_HOME` and copy necessary jar files to it:
   a. Copy the following files from `REGISTRY_HOME/lib` to the `CLIENT_HOME` directory:
      - `activation.jar`

- `builtin_serialization.jar`
- `core_services_client.jar`
- `jaas.jar`
- `jaxm.jar`
- `jaxrpc.jar`
- `jce1_2_1.jar`
- `jcert.jar`
- `jetty.jar`
- `jnet.jar`
- `jsse.jar`
- `local_policy.jar`
- `log4j.jar`
- `mgmt_services_client.jar`
- `saaj.jar`
- `security-ng.jar`
- `security2-ng.jar`
- `security_providers_client.jar`
- `security_services_client.jar`
- `security_tools.jar`
- `sunjce_provider.jar`
- `US_export_policy.jar`
- `wasp.jar`
- `wasp_permissions.jar`
- `wsdl_api.jar`
- `xalan.jar`
- `xercesImpl.jar`
- `xml-apis.jar`
- `xmlParserAPIs.jar`

b. Copy the following files from `REGISTRY_HOME/dist` to the `CLIENT_HOME` directory:

- `uddiclient_core.jar`
- `uddiclient_api_ v3.jar`

c. Copy `security_providers.jar` from `REGISTRY_HOME/app/system` to the `CLIENT_HOME` directory

d. Copy the following files from `REGISTRY_HOME/conf` to `CLIENT_HOME/conf` directory:

- `clientconf.xml`
- `jaas.config`
- `package12.xsd`

2. Create a client class as shown in Example 3, ExampleHttpBasis.java and compile the class with all jars from the `CLIENT_HOME` directory in the CLASSPATH.

3. Modify `REGISTRY_HOME/app/uddi/services/WASP-INF/package.xml` to enable HTTP basic authentication as follows:

   a. Under `<processing name="UDDIv1v2v3PublishingProcessing"/>`, uncomment `<use ref="tns:HttpBasicInterceptor"/>`

   b. Under `<processing name="UDDIv1v2v3InquiryProcessing">`, add `<use ref="tns:HttpBasicInterceptor"/>`

   c. Add the attribute `accepting-security-providers="HttpBasic"` to all service-endpoints you wish to access via HTTP Basic authentication.

   A fragment of the `package.xml` is shown in Example 4, package.xml - HTTP Basic Enabled

4. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY_HOME/work` directory, and restart the registry.

5. Run the `ExampleHttpBasic.class` with the following parameter:

```
-Dwasp.location=CLIENT_HOME -Djava.security.auth.login.config=CLIENT_HOME\conf\jaas.config
```

2.5.1. Sample Files

Example 3. ExampleHttpBasis.java

```
// Copyright 2001-2005 Systinet Corp. All rights reserved.
// Use is subject to license terms.

import org.idoox.security.Credentials;
import org.idoox.wasp.SecurityHelper;
import org.idoox.wasp.WaspSecurity;
import org.systinet.wasp.Wasp;
import org.systinet.wasp.webservice.ServiceClient;

import org.systinet.uddi.client.v3.UDDIInquiryStub;
import org.systinet.uddi.client.v3.UDDIPublishStub;
import org.systinet.uddi.client.v3.UDDI_Inquiry_PortType;
import org.systinet.uddi.client.v3.UDDI_Publication_PortType;
import org.systinet.uddi.client.v3.struct.*;



public class ExampleHttpBasic {
    public static void main(String[] args) {

        String urlInquiry = "http://localhost:8080/uddi/inquiry";
        String urlPublishing = "http://localhost:8080/uddi/publishing";
        System.out.print("Using Inquiry at urlInquiry " + urlInquiry + " ..");
        ServiceClient serviceClient = ServiceClient.create();
        serviceClient.setServiceURL(urlInquiry);

        try {
            Wasp.init();
            Credentials credentials = WaspSecurity.acquireClientCredentials
                    ("demo_john", "demo_john", SecurityHelper.HttpBasic);
            WaspSecurity.setInitiatingProvider(serviceClient, SecurityHelper.HttpBasic);
            WaspSecurity.setCredentials(serviceClient, new Credentials[]{credentials});

            UDDI_Inquiry_PortType inquiry = UDDIInquiryStub.getInstance(serviceClient);
            System.out.println(inquiry.find_business(new Find_business()));

            serviceClient.setServiceURL(urlPublishing);
            UDDI_Publication_PortType publish = UDDIPublishStub.getInstance(serviceClient);
            System.out.println(publish.save_business(new Save_business
                    (new BusinessEntityArrayList(new BusinessEntity(new NameArrayList
                            (new Name("Business created by HttpBasic example")))))));

            System.out.println(" done");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Example 4. package.xml - HTTP Basic Enabled

```
.....
    <service-endpoint path="/inquiry" version="3.0" name="UDDIInquiryV3Endpoint"
        service-instance="tns:UDDIInquiryV3" processing="tns:UDDIv1v2v3InquiryProcessing"
          accepting-security-providers="HttpBasic">
        <wsdl uri="uddi_api_v3.wsdl" service="uddi_api_v3:UDDI_Inquiry_SoapService"/>
        <envelopePrefix xmlns="arbitraryNamespace" value=""/>
        <namespaceOptimization xmlns="arbitraryNamespace">false</namespaceOptimization>
    </service-endpoint>
   <service-instance implementation-class="com.systinet.uddi.publishing.v3.PublishingApiImpl"

        name="UDDIPublishingV3"/>
    <service-endpoint path="/publishing" version="3.0"
        name="UDDIPublishingV3Endpoint" service-instance="tns:UDDIPublishingV3"
        processing="tns:UDDIv1v2v3PublishingProcessing"
        accepting-security-providers="HttpBasic">
        <wsdl uri="uddi_api_v3.wsdl"
            service="uddi_api_v3:UDDI_Publication_SoapService"/>
        <envelopePrefix xmlns="arbitraryNamespace" value=""/>
        <namespaceOptimization xmlns="arbitraryNamespace">false</namespaceOptimization>
    </service-endpoint>

    <processing name="UDDIv3Processing">
      <use ref="uddiclient_v3:UDDIClientProcessing"/>
      <fault-serialization name="MessageTooLargeFaultSerializer"
     serializer-class="com.systinet.uddi.publishing.v3.serialization.MessageTooLargeFaultSeri-
alizer"
     serialized-exception-class="com.systinet.uddi.interceptor.wasp.MessageTooLargeException"/>

    </processing>

    <processing name="UDDIv1v2v3PublishingProcessing">
     <use ref="uddiclient_v3:UDDIClientProcessing"/>
     <use ref="uddiclient_v2:UDDIClientProcessing"/>
     <use ref="uddiclient_v1:UDDIClientProcessing"/>
     <!-- HttpBasic (without authtoken)         -->
     <use ref="tns:HttpBasicInterceptor"/>

       <interceptor name="MessageSizeCheckerInterceptor"
         implementation-class="com.systinet.uddi.interceptor.wasp.MessageSizeCheckerInterceptor"

          direction="in">
            <config:maxMessageSize>2097152</config:maxMessageSize>
       </interceptor>
    </processing>

    <processing name="UDDIv1v2v3InquiryProcessing">
        <use ref="tns:UDDIv3Processing"/>
        <use ref="tns:UDDIv2Processing"/>
        <use ref="tns:UDDIv1Processing"/>
        <use ref="tns:HttpBasicInterceptor"/>
    </processing>
.....
```

3. Server-Side Development

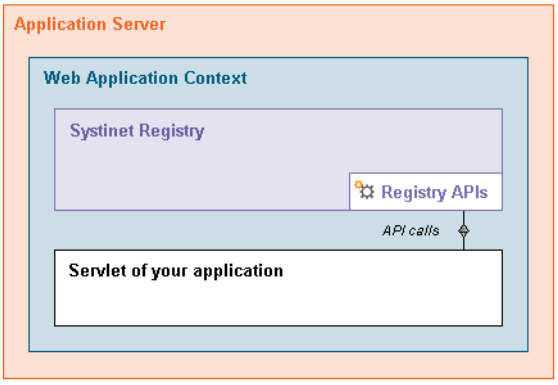
This chapter focuses on the server-side development of BEA AquaLogic Service Registry extensions. Possible ways of accessing BEA AquaLogic Service Registry are discussed including examples.

- Accessing backend APIs via servlet deployed on an application server.
- Custom BEA AquaLogic Service Registry Modules - how to create and deploy custom BEA AquaLogic Service Registry modules.
- Interceptors can monitor or modify the requests and responses of BEA AquaLogic Service Registry. Interceptors are at the lowest level of BEA AquaLogic Service Registry API call processing.
- Writing custom Validation services - BEA AquaLogic Service Registry provides several ways to define and use validation services for taxonomies or identifier systems inluding remotely and locally deployed validation services and an internal validation service. For details, please see User's Guide, Section 5.4, Taxonomy: Principles, Creation and Validation. This chapter focuses how to create a validation service.
- Writing subscription notification services - How to implement subscription notification service deployed on Systinet Server for Java.
- JSP Framework - This section covers the Systinet Web Framework.
- Business Service Console Framework - This section covers the Business Service Console Framework.

3.1. Accessing Backend APIs

This section will show you how to integrate BEA AquaLogic Service Registry with your application. Your application can be deployed as a servlet to the same context of the application server as the registry. In this case, the servlet of your application can access instances of BEA AquaLogic Service Registry APIs as shown in Figure 5.

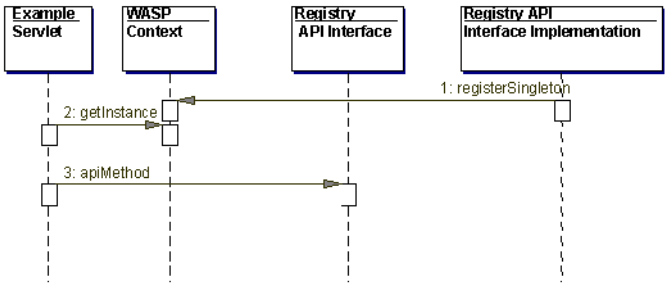
Figure 5. Accessing Backend Registry APIs - Architecture View



The sequence of steps that precedes access to the BEA AquaLogic Service Registry API is shown in Figure 6.

1. BEA AquaLogic Service Registry's API implementations are registered in the `WASP` context during the boot of the registry.
2. The example servlet deployed in the `WASP` context calls the `getInstance()` method with the required UDDI Registry interface as a parameter to obtain a reference of the interface implementation.
3. The example servlet can call the API methods of BEA AquaLogic Service Registry.

Figure 6. Accessing Backend Registry APIs - Sequence Diagram



Follow these steps to create and deploy the example servlet:

1. Create the example servlet class shown in Example 5, ExampleServet.java .

   Compile the `ExampeServlet.java` using:

   ```
   javac -classpath %REGISTRY_HOME%\dist\uddiclient_api_v3.jar;
   %REGISTRY_HOME%\dist\uddiclient_core.jar;
   %REGISTRY_HOME%\lib\wasp.jar;
   ```

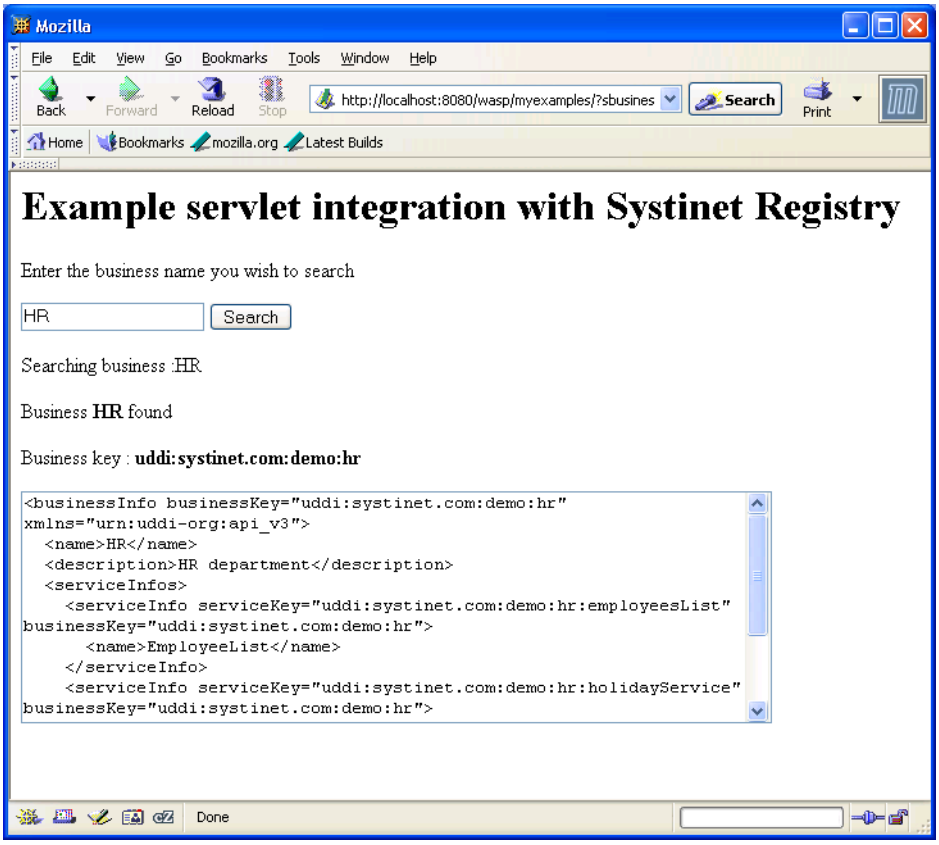
2. Copy `ExampleServlet.class` to the directory `webapps/wasp/Web-inf/classes/com/systinet/example/servlet`.

3. Add the example servlet to `webapps/wasp/Web-inf/web.xml` as shown in Example 6, Example Servlet's web.xml .

The example servlet will be available at `http://localhost:8080/wasp/myexamples`.

You can test it as shown at Figure 7.

Figure 7. Example Servlet Output



Example 5. ExampleServet.java

```
            package com.systinet.example.servlet;

import org.idoox.wasp.Context;
import org.idoox.wasp.InstanceNotFoundException;
import org.systinet.uddi.InvalidParameterException;
import org.systinet.uddi.client.v3.UDDIException;
import org.systinet.uddi.client.v3.UDDI_Inquiry_PortType;
import org.systinet.uddi.client.v3.struct.*;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Iterator;

public class ExampleServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        try {
            String searchedBusiness = request.getParameter("sbusiness");
            if (searchedBusiness == null) searchedBusiness = "";
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.println("<HTML>");
            out.println("<HEAD>");
            out.println("<H1>Example servlet integration with BEA AquaLogic Service Re-
gistry</H1>");
            out.println("<P>Enter the business name you wish to search");
            out.println("<FORM METHOD=GET ACTION=/wasp/myexamples/>");
            out.println("<INPUT NAME=sbusiness SIZE=20 VALUE=" + searchedBusiness + ">");
            out.println("<INPUT TYPE=SUBMIT VALUE=Search>");
            out.println("</FORM>");

            // get UDDI API V3 Inquiry implementation
            UDDI_Inquiry_PortType inquiry =
                (UDDI_Inquiry_PortType) Context.getInstance(UDDI_Inquiry_PortType.class);

            // prepare find_business call
            Find_business find_business = new Find_business();
            if (searchedBusiness.length() > 0) {
                find_business.addName(new Name(searchedBusiness));
                out.println("<P>Searching business :" + searchedBusiness);
                // call find_business
                BusinessList businessList = inquiry.find_business(find_business);
                // process the result
                BusinessInfoArrayList businessInfoArrayList
                    = businessList.getBusinessInfoArrayList();
                if (businessInfoArrayList == null) {
```

```
                    out.println("<P><B>Nothing found</B>");
                } else {

                    out.println("<P>Business <B>"+searchedBusiness+"</B> found");
                    for (Iterator iterator =
                       businessInfoArrayList.iterator(); iterator.hasNext();) {
                        BusinessInfo businessInfo = (BusinessInfo) iterator.next();
                        out.println("<P>Business key : <B>" +
                            businessInfo.getBusinessKey()+"</B>");
                        out.println("<P><TEXTAREA ROWS=10 COLS=70>");
                        out.println(businessInfo.toXML());
                        out.println("</TEXTAREA");

                    }

                }
            }
            out.println("</HTML>");
        } catch (InvalidParameterException e) {
        } catch (InstanceNotFoundException e) {
        } catch (UDDIException e) {
        }

    }
}
```

Example 6. Example Servlet's `web.xml`

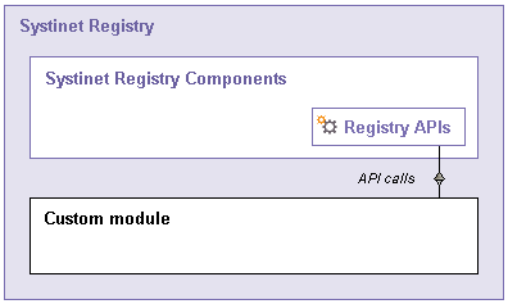
```
            <servlet>
    <servlet-name>ExampleServlet</servlet-name>
    <servlet-class>com.systinet.example.servlet.ExampleServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>ExampleServlet</servlet-name>
    <url-pattern>/myexamples/*</url-pattern>
</servlet-mapping>
```

3.2. Custom Registry Modules

In this section, we will show you how to extend BEA AquaLogic Service Registry functionality with your custom modules. Custom modules can be added to BEA AquaLogic Service Registry as shown in Figure 8.

Figure 8. Custom Registry Module - Architecture View



To create and deploy a registry module, follow these steps:

1. Write a class that implements `org.systinet.uddi.module.Module`.
2. Copy your module implementation class to the directory `REGISTRY_HOME/app/uddi/services/WASP-INF/classes`.
3. Create a configuration file for the module in `REGISTRY_HOME/app/uddi/conf`.
4. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY_HOME/work` directory, and restart the registry.

The main class of the custom module must implement `org.systinet.uddi.module.Module` interface that has these methods:

- `load()` is invoked as the first method of the module. You can put reading of the configuration file in here.
- `init()` is invoked after the `load()` method. Put the core implementation of your module in here. Write non-blocking code or start a new thread.
- `destroy()` is invoked just before the BEA AquaLogic Service Registry shutdown.

3.2.1. Accessing Registry APIs

To access the BEA AquaLogic Service Registry API you must obtain the API stub using the `getApiInstance()` method of the API implementation class. For example to obtain the stub of the Statistics API use:

```
StatisticsApi statapi = StatisticsApiImpl.getApiInstance();
```

Mapping between API interface classes and implementation classes is stored in the `REGISTRY_HOME/app/uddi/services/WASP-INF/package.xml` file. See Table 53, “Mapping API Interface and Implemenation Classes”.

Table 53. Mapping API Interface and Implemenation Classes

Interface class	Implementation class
org.systinet.uddi.client.v1.InquireSoap	com.systinet.uddi.inquiry.v1.InquiryApiImpl
org.systinet.uddi.client.v1.PublishSoap	com.systinet.uddi.publishing.v1.PublishingApiImpl
org.systinet.uddi.client.v2.Publish	com.systinet.uddi.publishing.v2.PublishingApiImpl
org.systinet.uddi.client.v2.Inquire	com.systinet.uddi.inquiry.v2.InquiryApiImpl
org.systinet.uddi.client.v3.UDDI_Security_PortType	com.systinet.uddi.v3.SecurityApiImpl
org.systinet.uddi.client.v3.UDDI_Publication_PortType	com.systinet.uddi.publishing.v3.PublishingApiImpl
org.systinet.uddi.client.v3.UDDI_Inquiry_PortType	com.systinet.uddi.inquiry.v3.InquiryApiImpl
org.systinet.uddi.client.subscription.v3.UDDI_Subscription_PortType	com.systinet.uddi.subscription.v3.SubscriptionApiImpl
org.systinet.uddi.client.custody.v3.UDDI_CustodyTransfer_PortType	com.systinet.uddi.custody.v3.CustodyApiImpl
org.systinet.uddi.replication.v3.ReplicationApi	com.systinet.uddi.replication.v3.ReplicationApiImpl
org.systinet.uddi.client.wsdl2uddi.v3.Wsdl2uddiApi	com.systinet.uddi.wsdl2uddi.v3.Wsdl2uddiApiImpl
org.systinet.uddi.client.wsdl2uddi.v2.Wsdl2uddiApi	com.systinet.uddi.wsdl2uddi.v2.Wsdl2uddiApiImpl
org.systinet.uddi.client.category.v3.CategoryApi	com.systinet.uddi.category.v3.CategoryApiImpl
org.systinet.uddi.client.taxonomy.v3.TaxonomyApi	com.systinet.uddi.taxonomy.v3.TaxonomyApiImpl
org.systinet.uddi.statistics.StatisticsApi	com.systinet.uddi.statistics.StatisticsApiImpl
org.systinet.uddi.admin.AdministrationUtilsApi	com.systinet.uddi.admin.AdministrationUtilsApiImpl
org.systinet.uddi.permission.PermissionApi	com.systinet.uddi.permission.PermissionApiImpl
org.systinet.uddi.group.GroupApi	com.systinet.uddi.group.GroupApiImpl
org.systinet.uddi.account.AccountApi	com.systinet.uddi.account.AccountApiImpl
org.systinet.uddi.configurator.ConfiguratorApi	com.systinet.uddi.configurator.cluster.ConfiguratorApiImpl

3.2.2. Custom Module Sample

This section includes step-by-step instructions how to create a registry module that counts the number of restarts of BEA AquaLogic Service Registry and saves the result to a configuration file.

Follow these steps:

1. Create Java file `ExampleModule.java` as shown in Example 7, ExampleModule.java
2. Compile the module using **java -classpath "%REGISTRY_HOME%\app\uddi\services\WASP-INF\lib\application_ core.jar; %REGISTRY_HOME%\lib\wasp.jar" ExampleModule.java**
3. Copy all module classes (`ExampleModule.class`, `ExampleModule$RestartConfig$Counter.class`, `ExampleModule$RestartConfig.class`) to the `REGISTRY_HOME/app/uddi/services/WASP-INF/classes/com/systinet/example/module` directory.
4. Create the configuration file `mymodule.xml` in `REGISTRY_HOME/app/uddi/conf` folder. For details, please see Example 8, Example configuration file for custom module.
5. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY_HOME/work` directory, and restart the registry.

The number of restarts will be printed in the window console in which you started BEA AquaLogic Service Registry. See also the configuration file of the module where a new element `counter` is created.

Example 7. `ExampleModule.java`

```
package com.systinet.example.module;

import org.idoox.config.Configurable;
import org.systinet.uddi.module.Module;

public class ExampleModule implements Module {
    private long restart = 0;
    private RestartConfig.Counter counter;

    interface RestartConfig {
        public Counter getCounter();
        public void setCounter(Counter counter);
        public Counter newCounter();
        interface Counter {
            public long getRestart();
            public void setRestart(long restart);
        }
    }

    public void load(Configurable config) {
        System.out.println("MY MODULE CONFIG READING");
        RestartConfig restartConfig = (RestartConfig) config.narrow(RestartConfig.class);
        if (restartConfig != null) {
            counter = restartConfig.getCounter();
            if (counter == null) {
                counter = restartConfig.newCounter();
                restartConfig.setCounter(counter);
            }
            try {
                restart = counter.getRestart();
            } catch (Exception e) {
                counter.setRestart(0);
            }
        }
    }

    public void init() {
        System.out.println("MY MODULE STARTED");
        counter.setRestart(++restart);
        System.out.println("UDDI REGISTRY: number of restarts = " + restart);
    }

    public void destroy() {
    }
}
```

Example 8. Example configuration file for custom module

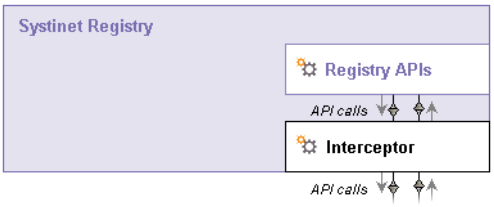
```
<?xml version="1.0" encoding="UTF-8"?>
<config name="myconf">
    <module loader="com.systinet.example.module.ExampleModule" name="MyModule">
    </module>
</config>
```

3.3. Interceptors

Interceptors can monitor or modify the requests and responses of BEA AquaLogic Service Registry as shown in Figure 9. They are at the lowest level of BEA AquaLogic Service Registry API call processing, and can be used for:

- Logging requests. See Section 3.3.2, Logging Interceptor Sample.
- Computing message statistics. See Section 3.3.3, Request Counter Interceptor Sample.
- Changing request arguments (adding default values)
- Prohibiting some API calls

Figure 9. Registry Interceptors



There are three types of BEA AquaLogic Service Registry interceptor:

- **Request Interceptor** Monitors or modifies request arguments, stops processing requests, or throws an exception. This type of interceptor accepts a called method object and its arguments.
- **Response Interceptor** Monitors or modifies response values or throws an exception. This interceptor accepts a called method object and its response value.
- **Exception Interceptor** Monitors, modifies, or changes an exception. This interceptor accepts a called method object and its thrown exception.

If you want to directly access the BEA AquaLogic Service Registry API see Section 3.2.1, Accessing Registry APIs for more information.

3.3.1. Creating and Deploying Interceptors

To create an Interceptor, follow these steps:

1. Write a class that implements the `org.systinet.uddi.interceptor` interface.
2. Copy your interceptor implementation class to the directory `REGISTRY_HOME/app/uddi/services/Wasp-inf/classes`.
3. Create a configuration file for your interceptor in the `REGISTRY_HOME/app/uddi/conf` directory. See Section Interceptor Configuration.

4. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY_HOME/work` directory, and restart the registry.

3.3.2. Logging Interceptor Sample

This section includes step-by-step instructions how to create the interceptor that logs requests.

To create a logging interceptor:

1. Create Java file `LoggingInterceptor.java` as shown in Example 9, Logging Interceptor Class.
2. Compile the interceptor using **Java -classpath "%REGISTRY_HOME%\app\uddi\services\Wasp-inf\lib\application_core.jar; %REGISTRY_HOME%\lib\wasp.jar" LoggingInterceptor.java**
3. Copy `LoggingInterceptor.class` to the `REGISTRY_HOME/app/uddi/services/Wasp-inf/classes/interceptor` directory.
4. Create the configuration file `Myinterceptor.xml` in `REGISTRY_HOME/app/uddi/conf` folder. For details, please see Example 10, Logging Interceptor Configuration File.
5. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY_HOME/work` directory, and restart the registry.

Example 9. Logging Interceptor Class

```
package interceptor;

import org.idoox.config.Configurable;
import org.idoox.wasp.WaspInternalException;
import org.idoox.wasp.interceptor.InterceptorChain;
import org.systinet.uddi.interceptor.ExceptionInterceptor;
import org.systinet.uddi.interceptor.RequestInterceptor;
import org.systinet.uddi.interceptor.ResponseInterceptor;
import org.systinet.uddi.interceptor.StopProcessingException;
import java.lang.reflect.Method;


public class LoggingInterceptor implements RequestInterceptor,
                     ResponseInterceptor, ExceptionInterceptor {

    public void load(Configurable config)
        throws WaspInternalException {
        // no initialization required
    }

    public void destroy() {
        // no destroy required
    }

    public void intercept(Method method,
                          Object[] args,
                          InterceptorChain chain,
                          int position)
        throws StopProcessingException, Exception {
        System.out.println("request: " + method.getName());
    }

    public Object intercept(Method method,
                            Object returnValue,
                            InterceptorChain chain,
                            int position)
        throws Exception {
        System.out.println("response: " + method.getName());
        return returnValue;
    }

    public Exception intercept(Method method,
                               Exception e,
                               InterceptorChain chain,
                               int position) {
        System.out.println("exception: " + method.getName());
        return e;
    }
}
```

Example 10. Logging Interceptor Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="MyInterceptorConfig">
  <UDDIInterceptorInstance name="LoggingInterceptorInstance"
    instancePerCall="false"
    className="interceptor.LoggingInterceptor"/>
  <UDDIInterceptor name="LoggingInterceptor"
    instanceName="LoggingInterceptorInstance"
    interceptorChain="inquiry_v3"
    request="true"
    response="true"
    fault="true" />
</config>
```

Interceptor Configuration

The configuration file must be present in the `REGISTRY_HOME/app/uddi/conf` directory. For details please see Example 10, Logging Interceptor Configuration File. Interceptors are called in the same order as they appear in the configuration file.

- config name - the unique (unambiguous) name of the configuration.
- `UDDIInterceptorInstance` - contains information about the implementation class and its instantiation.
  - `name` - The name of interceptor instance. This name is used as a link to the UDDIInterceptor/instanceName section of the configuration.
  - `instancePerCall` - If the `instancePerCall` attribute is set to true, then the class will be instantiated once per API call. Otherwise, this interceptor instantiates only once for all calls.
  - `className` - name of the class that implements the interceptor.
- `UDDIInterceptor` - The UDDIInterceptor contains references to UDDI Interceptors and their types.
  - `name` - name of the interceptor.
  - `instanceName` - this attribute contains the name of the UDDIInterceptorInstance section of the configuration file.
  - `interceptorChain` - UDDIInterceptorChains are defined for each API in their configuration files. This attribute contains a reference to the required API.
  - `request` - when set true, the interceptor catches requests.
  - `response` - when set true, the interceptor catches responses.
  - `fault` - when set true, the interceptor catches faults.

3.3.3. Request Counter Interceptor Sample

In this section, we will create an interceptor that counts requests and stores the number of request to a configuration file. The steps required to create a Request Counter Interceptor are the same as those in the Section 3.3.2, Logging Interceptor Sample.

Interceptor implementation is shown in Example 11, Request Counter Interceptor Class; the configuration file is shown in Example 12, Request Counter Interceptor Configuration File.

Example 11. Request Counter Interceptor Class

```
package interceptor;

import org.idoox.config.Configurable;
import org.idoox.wasp.WaspInternalException;
import org.idoox.wasp.interceptor.InterceptorChain;
import org.systinet.uddi.interceptor.RequestInterceptor;
import org.systinet.uddi.interceptor.StopProcessingException;
import java.lang.reflect.Method;

public class RequestCounterInterceptor implements RequestInterceptor {

    private long request = 0;
    private RequestCounterInterceptorConfig.Counter counter;

    /**
    * RequestCounterInterceptor config interface
    */
    interface RequestCounterInterceptorConfig {
        public Counter getCounter();
        public void setCounter(Counter counter);
        public Counter newCounter();
        interface Counter {
            public long getRequest();
            public void setRequest(long request);
        }
    }
    public void intercept(Method method,
                          Object[] args,
                          InterceptorChain chain,
                          int position)
        throws StopProcessingException, Exception {
        counter.setRequest(++request);
        System.out.println("request: " + request);
    }

    public void load(Configurable config)
        throws WaspInternalException {
        RequestCounterInterceptorConfig intinterceptorConfig =
                          (RequestCounterInterceptorConfig)
                           config.narrow(RequestCounterInterceptorConfig.class);
        if (intinterceptorConfig != null) {
            counter = intinterceptorConfig.getCounter();
            if (counter == null) {
                counter = intinterceptorConfig.newCounter();
                intinterceptorConfig.setCounter(counter);
            }
            try {
                request = counter.getRequest();
            } catch (Exception e) {
            counter.setRequest(0);
            }
        }
```

```
    }

    /**
    * Destroys the interceptor.
    */
    public void destroy() {
        // no destroy required
    }
}
```

Example 12. Request Counter Interceptor Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<config name="myInterceptors">
    <UDDIInterceptorInstance className="interceptor.RequestCounterInterceptor"
        instancePerCall="false" name="RequestCounterInterceptorSampleInstance">
    </UDDIInterceptorInstance>
    <UDDIInterceptor fault="false"
        instanceName="RequestCounterInterceptorSampleInstance"
        interceptorChain="inquiry_v3" name="RequestCounter" request="true"
        response="false"/>
</config>
```

3.4. Writing a Custom Validation Service

BEA AquaLogic Service Registry provides several ways to define and use validation services for taxonomies or identifier systems. For details about BEA AquaLogic Service Registry taxonomies, please see User's Guide, Section 5.4, Taxonomy: Principles, Creation and Validation. This chapter focuses on custom validation services that you can deploy:

- Locally on BEA AquaLogic Service Registry - Local validation service.
- Remotely to a SOAP server, for example the Systinet Server for Java - External validation service.

There are three different Java interfaces for validation services, one for each of the main UDDI data structures. These interfaces correspond to the WSDL Port Types of the Validation Service defined in the UDDI specification.

- UDDI v3 validation services must implement `org.systinet.uddi.client.valueset.validation.v3.UDDI_ValueSetValidation_PortType` .
- UDDI v2 validation services must implement `org.systinet.uddi.client.vv.v2.ValidateValues` .
- UDDI v1 validation services must implement `org.systinet.uddi.client.vv.v1.ValidateValues` .

These interfaces are similar enough that we will only describe v3 validation. Your validation service must implement the interface `UDDI_ValueSetValidation_PortType`. This interface only has the `validate_values` method which has only one parameter, Validate_values. This parameter is a wrapper for real parameters: optional authInfo and basic UDDI data structures (businessEntities, businessServices, bindingTemplates, tModels and publisherAssertions) to validate. The `validate_values` method returns `DispositionReport`. If validation passes successfully, the DispositionReport should contain only one `Result` with errNo equals `UDDIErrorCodes.E_SUCCESS`.

3.4.1. Deploying Validation Service

Once the validation service is implemented, you can deploy the validation service locally on BEA AquaLogic Service Registry. To deploy the validation service on BEA AquaLogic Service Registry

1. Create a `classes` subdirectory under `REGISTRY_HOME/app/uddi/services/WASP-INF` and copy the class file into this directory (with respect to subdirectories corresponding to packages).

2. Shutdown BEA AquaLogic Service Registry, delete the `REGISTRY/work` directory, and restart BEA AquaLogic Service Registry.

For more information, please see the Demos, Section 2.4, Validation. For details about the configuration of Validation Services, please see Administrator's Guide, Section 1.5, Taxonomy Management

To deploy an external validation service, you must create a deployment package.

3.4.2. External Validation Service

This section shows you how to implement and package an external validation service that will be deployed to Systinet Server for Java 5.5. We show you how to package and deploy the ISBN validation service from the validation demo described in Section 2.4, Validation. We assume you have already built the Validation demo.

Note

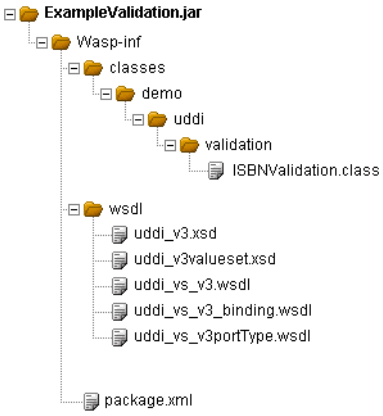
We also assume BEA AquaLogic Service Registry is installed in the `REGISTRY_HOME` folder and running at `http://localhost:8080/` and that

Systinet Server for Java is installed in `WASP_HOME` folder and running at `http://localhost:6060/`

To package and deploy a validation service to Systinet Server for Java:

1. Create a deployment package.

   Create the jar file `ExampleValidation.jar` with the following structure:



   Copy `ISBNValidation.class` from `REGISTRY_HOME/demos/advanced/validation/build/classes` to the package.

   Copy the wsdl and xsd files from `REGISTRY_HOME/doc/wsdl` to the package.

   Copy the `package.xml` file shown at Example 13, package.xml to the package.

2. Deploy the validation package with required BEA AquaLogic Service Registry client packages into Systinet Server for Java 5.5.

   a. **copy %REGISTRY_HOME%\dist\uddiclient_api_v3.jar %WASP_HOME%\app\system\uddi**

b. **copy %REGISTRY_HOME%\dist\uddiclient_value_set_validation_v3.jar %WASP_HOME%\app\system\uddi**

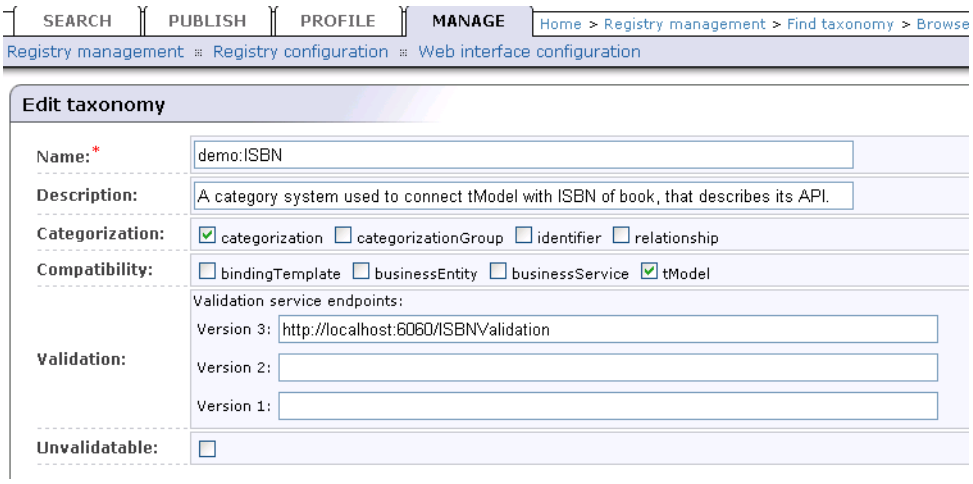
c. **copy ExampleValidation.jar %WASP_HOME%\app\system\uddi**

3. Shutdown the Systinet Server for Java, delete the WASP_HOME/work directory, and restart the Systinet Server for Java

Now you can upload the checked taxonomy from `REGISTRY/demos/advanced/validation/data`. For more information, please see User's Guide Section 1.5.5, Uploading Taxonomies.

Modify the validation service endpoint as shown in Figure 10

Figure 10. Validation for Checked Taxonomy



You can run and test the validation service using Validation demo described in Section 2.4, Validation.

3.4.3. Sample Files

Example 13. `package.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://systinet.com/wasp/package/1.2"
    xsi:schemaLocation=
       "http://systinet.com/wasp/package/1.2 http://systinet.com/wasp/package/1.2"
    targetNamespace="http://my.org" version="1.0"
    name="ISBNValidation" client-package="false" library="false"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tns="http://my.org"

    xmlns:UDDIClient-value-set-validation-v3=
        "http://systinet.com/uddi/client/value-set-validation/v3/5.0">

<dependency ref="UDDIClient-value-set-validation-v3:UDDIClient-value-set-validation-v3"
   version="5.0"/>
    <service-endpoint name="ISBNValidation"
        path="/ISBNValidation"
        service-instance="tns:ISBNValidationInstance"
    processing="UDDIClient-value-set-validation-v3:UDDIClientProcessing">
        <wsdl uri="uddi_vs_v3.wsdl" xmlns:wsdl="urn:uddi-org:vs_v3_binding"
           service="wsdl:UDDI_ValueSetValidation_SoapService"/>
    </service-endpoint>
    <service-instance name="ISBNValidationInstance"
        implementation-class="demo.uddi.validation.ISBNValidation"
        preload="false" ttl="600" instantiation-method="shared"/>
</package>
```

3.5. Writing a Subscription Notification Service

This section will show you how to implement a subscription notification service. When you create a BEA AquaLogic Service Registry subscription you can specify a notification listener service endpoint as described in Section 1.4, Subscriptions in BEA AquaLogic Service Registry. In this chapter, we describe the following use case: The user wants to create a service that will be executed when a subscription notification is sent. The listener notification service will be deployed on the Systinet Server for Java.

The procedure of creating and deploying the subscription notification consist of the following steps:

1. Create subscription notification service class. Package the notification service class with necessary wsdl, schema, and deployment descriptor files.

2. Deploy the service notification package with the required BEA AquaLogic Service Registry client packages into Systinet Server for Java.

3. Create a subscription using the Registry Console.

Note

We assume BEA AquaLogic Service Registry is installed in `REGISTRY_HOME` folder and running at `http://localhost:8080/`, and that

Systinet Server for Java is installed in `WASP_HOME` folder and running at `http://localhost:6060/`.

Now we will describe the process in detail:

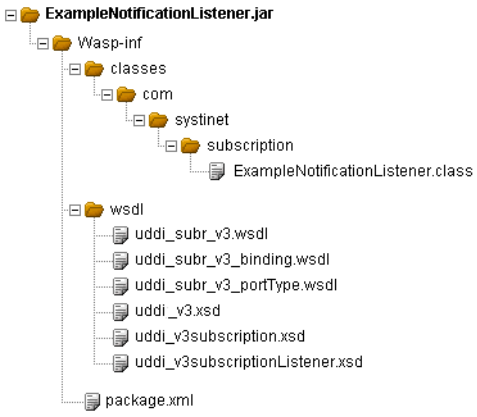
1. Create the subscription notification service class shown in Example 14, ExampleNotificationListener.java

2. Compile the `ExampleNotificationListener.java` using:

```
javac -classpath%REGISTRY_HOME%\dist\uddiclient_api_v3.jar;
%REGISTRY_HOME%\dist\uddiclient_core.jar;
%REGISTRY_HOME%\dist\uddiclient_subscription_listener_v3.jar;
%REGISTRY_HOME%\dist\uddiclient_subscription_v3.jar ExampleNotificationListener.java
```

3. Package the `ExampleNotificationListener.class` with necessary wsdl, schema and deployment descriptor file as follows:

   a. Create a jar file `ExampleNotificationListener.jar` with the following structure:



   b. Copy the wsdl and schema files from `REGISTRY_HOME/doc/wsdl` to the package.

   c. Copy the `package.xml` file shown in Example 15, package.xml to the package.

4. Deploy the service notification package with required BEA AquaLogic Service Registry client packages into Systinet Server for Java 5.5.

   a. **copy %REGISTRY_HOME%\dist\uddiclient_api_v3.jar %WASP_HOME%\app\system\uddi**

   b. **copy %REGISTRY_HOME%\dist\uddiclient_subscription_v3.jar %WASP_HOME%\app\system\uddi**

   c. **copy %REGISTRY_HOME%\dist\uddiclient_subscription_listener_v3.jar %WASP_HOME%\app\system\uddi**

   d. **copy ExampleNotificationListener.jar %WASP_HOME%\app\system\uddi**

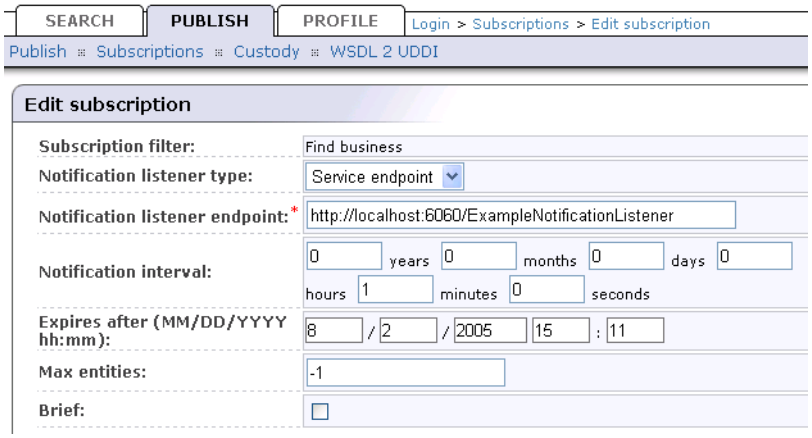
5. Shutdown the Systinet Server for Java, delete the WASP_HOME/work directory, and restart the Systinet Server for Java

6. Create a subscription using the Registry Console.

See Section Publishing Subscriptions for instructions on how to create a subscription.

7. Publish the subscription with the Notification listener type Service endpoint. Enter the Notification listener endpoint as `http://your.computer.name.com:6060/ExampleNotificationListener` as shown in Figure 11

Figure 11. Create Subscription



3.5.1. Sample Files

Example 14. `ExampleNotificationListener.java`

```
package com.systinet.subscription;

import org.systinet.uddi.client.subscription.listener.v3.UDDI_SubscriptionListener_PortType;
import org.systinet.uddi.client.subscription.listener.v3.struct.Notify_subscriptionListener;
import org.systinet.uddi.client.v3.UDDIException;
import org.systinet.uddi.client.v3.struct.DispositionReport;

public class ExampleNotificationListener implements UDDI_SubscriptionListener_PortType{

    public DispositionReport notify_subscriptionListener(Notify_subscriptionListener body)
       throws UDDIException {
            System.out.println(body.toXML());
            DispositionReport result = DispositionReport.DISPOSITION_REPORT_SUCCESS;
            return result;
    }
}
```

Example 15. `package.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<package xmlns="http://systinet.com/wasp/package/1.2"
    xsi:schemaLocation="http://systinet.com/wasp/package/1.2 http://systinet.com/wasp/pack-
age/1.2"
    targetNamespace="http://my.org" version="1.0"
    name="ExampleNotificationListener" client-package="false" library="false"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tns="http://my.org"

    xmlns:uddi_subr_v3="urn:uddi-org:subr_v3_binding"
    xmlns:uddiclient_subscription_listener_v3=
       "http://systinet.com/uddi/client/subscription/listener/v3/5.0">

    <dependency ref=
      "uddiclient_subscription_listener_v3:UDDIClient-subscription-listener-v3" version="5.0"/>


    <service-endpoint name="ExampleNotificationListener"
        path="/ExampleNotificationListener"
        service-instance="tns:ExampleNotificationListenerInstance"
        processing="uddiclient_subscription_listener_v3:UDDIClientProcessing">
        <wsdl uri="uddi_subr_v3.wsdl"
            service="uddi_subr_v3:UDDI_SubscriptionListener_SoapService"/>
    </service-endpoint>
    <service-instance name="ExampleNotificationListenerInstance"
        implementation-class="com.systinet.subscription.ExampleNotificationListener"
        preload="false" ttl="600" instantiation-method="shared"/>
</package>
```

3.6. Writing a Content Checker

In this section, we will show you how to create a content checker. The content checker provides an approver the ability to programmatically check data for approval. We assume you are familiar with the Approval Process, which is described in the following sections:

- User's Guide, Section 1.5, Approval Process in BEA AquaLogic Service Registry
- Administrator's Guide, Section 6, Approval Process Principles

We will show you how to create and deploy a content checker on the following example: an Approver set a rule that each business entity name must start with prefix "org_". Data that does not satisfy this rule cannot be approved (that is, copied to a *discovery registry*). The content checker is executed when a approver clicks on the **Approve** button in the Approve request page of the BEA AquaLogic Service Registry console.

To set up this optional content checking:

1. Write a class that implements the class `org.systinet.uddi.approval.checker.v3.CheckerApi`.
2. Deploy the implementation class to the BEA AquaLogic Service Registry.
3. Register the implementation of the content checker class in the BEA AquaLogic Service Registry's data.

Now, we will look at the steps in detail:

1. Write a class that implements the `org.systinet.uddi.approval.checker.v3.CheckerApi`

   a. Create the content checker class as shown in Example 16, Content Checker Implementation .

   b. Compile the `CheckerApiImpl.java`, and add jars from the directory `PUBLICATION_REGISTRY_HOME/dist` to the class path.

2. Deploy the implementation class to the BEA AquaLogic Service Registry.

   a. Copy the `CheckerApiImpl.class` to the file `PUBLICATION_REGISTRY_HOME/app/uddi/services/WASP-INF/lib/approval_staging_v3.jar` to the folder `com/systinet/uddi/approval/v3/approver` inside the jar file.

   b. Shutdown the *Publication Registry*, delete the `PUBLICATION_REGISTRY_HOME/work` directory, and restart the *Publication Registry*.

3. Register the implementation of the content checker class in the BEA AquaLogic Service Registry data.

   a. Log on to the *Publication Registry* as an approver. The content checker will be applicable to an approver who follows these steps:

   b. Publish the WSDL of the checker service:

      Publish the WSDL located at `http://<host_name>:<http_port>/uddi/doc/wsdl/approval_checker.wsdl` to a new or already existing business entity. Use the **Advanced publishing** mode and be sure to reuse the existing WSDL portType (tModel name:`CheckerApi`, tModel's key: `uddi:systinet.com:uddi:service:porttype:approvalchecker`). The WSDL service `approval_checker_SoapService` will be published under the business entity.

   c. Specify the checker in the access point of a new binding template under the `approval_checker_SoapService` service.

      Enter the value of access point which starts with the `class:` prefix and continue with the fully qualified class name. For example, `class:com.systinet.uddi.approval.v3.approver.CheckerApiImpl`.

Example 16. Content Checker Implementation

```
// Copyright 2001-2005 Systinet Corp. All rights reserved.
// Use is subject to license terms.
package com.systinet.uddi.approval.v3.approver;

import org.systinet.uddi.InvalidParameterException;
import org.systinet.uddi.approval.checker.v3.CheckerApi;
import org.systinet.uddi.approval.checker.v3.struct.CheckRequest;
import org.systinet.uddi.approval.v3.ApprovalErrorCodes;
import org.systinet.uddi.approval.v3.ApprovalException;
import org.systinet.uddi.approval.v3.struct.ApprovalEntitiesDetail;
import org.systinet.uddi.approval.v3.struct.EntitiesDetail;
import org.systinet.uddi.client.v3.struct.*;

/**
 * Checks if a BE starts with org_
 */
public class CheckerApiImpl implements CheckerApi {


    public DispositionReport checkRequest(CheckRequest checkRequest)
            throws ApprovalException {

        try {
            ResultArrayList resultArrayList = new ResultArrayList();

            ApprovalEntitiesDetail approvalEntitiesDetail =
                checkRequest.getApprovalEntitiesDetail();
            if (approvalEntitiesDetail != null) {
                EntitiesDetail entitiesDetail4Saving =
                    approvalEntitiesDetail.getEntitiesDetail4Saving();
                BusinessEntityArrayList businessEntityArrayList =
                    entitiesDetail4Saving.getBusinessEntityArrayList();
                if (businessEntityArrayList != null) {
                    for (int i = 0; i < businessEntityArrayList.size(); i++) {
                        BusinessEntity businessEntity = businessEntityArrayList.get(i);
                        if (businessEntity != null) {
                            NameArrayList nameArrayList =
                                businessEntity.getNameArrayList();
                            for (int j = 0; j < nameArrayList.size(); j++) {
                                Name name = nameArrayList.get(j);
                                if (name != null && !name.getValue().startsWith("org_")) {
                                  resultArrayList.add(
                                      new Result(ApprovalErrorCodes.INVALID_DATA,
                                            new ErrInfo(ApprovalErrorCodes.getCode(
                                      ApprovalErrorCodes.INVALID_DATA),
                                      "Only business entities whose name start with the " +
                                      "prefix \"org_\" are allowed" +
                                      " (BE [key: " + businessEntity.getBusinessKey() +
                                      ", name: " + name.getValue() + "])"),
                                            KeyType.businessKey));
                                }
                            }
```

```
                        }
                    }
                }
            }

            if (resultArrayList.size() > 0) {
                return new DispositionReport(resultArrayList);
            } else {
                return DispositionReport.DISPOSITION_REPORT_SUCCESS;
            }
        } catch (InvalidParameterException e) {
            // should not occur
            throw new ApprovalException(ApprovalErrorCodes.FATAL_ERROR, e.getMessage());
        }

    }
}
```

3.7. Systinet Web Framework

This section describes BEA AquaLogic Service Registry from the developer's point of view. It describes the BEA AquaLogic Service Registry Framework architecture and configuration.

- Section 3.7.1, Architecture Description
- Section 3.7.2, Directory Structure
- Section 3.7.3, Framework Configuration
- Section 3.7.4, syswf JSP tag library
- Section 3.7.5, Typical Customization Tasks

3.7.1. Architecture Description

The framework uses the Jasper engine, a part of the server. It is able to run on Jasper1 (Servlet API 2.3/JSP spec 1.2) or Jasper2 (Servlet API 2.4/JSP spec 2.0). It also uses a customized JSTL 1.0 tag library implementation which is based on Apache tag libraries from the Jakarta project [http://jakarta.apache.org/].

Applications using the Systinet Web Framework are composed of pages. Every page of the web has a URI where it can be accessed. In the Systinet Web Framework, we call each page of the web as a task.

The Systinet Web Framework uses a component model to build up the web application. Every task is assigned to a component which is the real entity behind the process that generates the resulting HTML page displayed to the user. Thus, every task references a component, but components need not be associated with tasks, as we will see later.

Each component is built from two parts:

- a JSP part
- a Java part

The JSP part serves as a template and takes care of parsing and visualization of the data that comes in a session, or in a request to which they are stored in the Java part of a component.

The framework functionality is accessible from the JSP parts of components through the Systinet custom JSP tag library. This library contains tags for creating references to tasks, nesting components, and tags for creating HTML form elements that support dynamic behavior.

Sometimes, a component is purely JSP-based as the one associated with this documentation page. But when the page must process user-entered information, or when data must be modified before presentation, you must use the Java part of the component.

To switch from one page to a another, use the `syswf:control` custom tag in the JSP part of the source task component. The `syswf:control` tag's `targetTask` attribute defines the task (that is, the page) the user should be transferred to. The custom tag is translated into a piece of JavaScript code responsible for correct page submitting.

Tasks can be accessed directly using a web browser. For example, if the registry's web interface runs on the address `http://localhost:8080/uddi/web`, a task with the URI `/findBusiness` can be accessed directly from the client browser at `http://localhost:8080/uddi/web/findBusiness`.

Component Java Interface Part

The Java part of the component must implement the `com.systinet.webfw.Component` interface from the Web Framework library. However, it usually extends its default implementation: `com.systinet.webfw.ComponentImpl`. For those components that do not declare their Java part, this default implementation is automatically used.

The interface consists of two methods:

- `void process(String action, Map params)`
- `void populate(String action, Map params)`

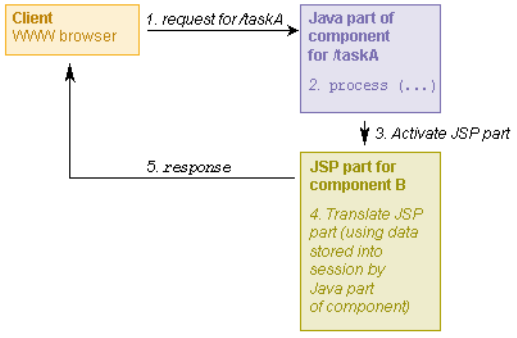
The `process()` method is called just before the translation of the component's JSP part is started, so it should take care of data preparation and it should also handle the actions requested by the user (react to pressed buttons, etc.).

The `populate()` method is called only when the POST request to the URI comes from the same URI , so it's a perfect place to modify the way data from a web page is populated back into objects. Actually, the target objects are always Java Beans which simplify their handling quite a bit.

Request Diagram

The diagram shown in Figure 12 demonstrates how requests for the page are handled by the Web Framework:

Figure 12. Request Diagram



1. The request is sent by the client browser from a different page than the page requested.

2. The `process()` method is called on taskA component's Java part. This method should perform actions triggered by controls in the web page and/or prepare data for taskA component's JSP part.

3. Processing of taskA component's JSP part is initialized.

4. While taskA component's JSP part is being processed, the resulting HTML is generated.

5. Processing of taskA component's JSP part finishes; the response is returned to the client's browser.

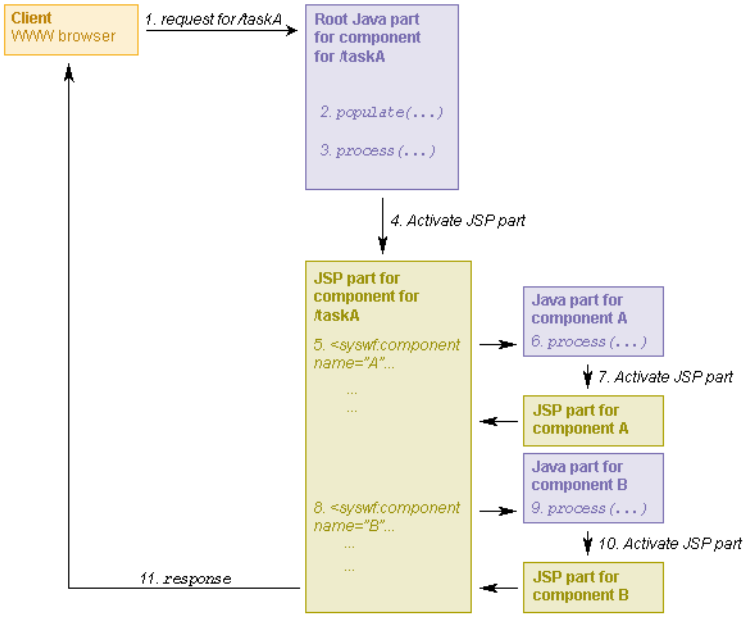
Note

If the request is sent by the client browser from the same page as the page requested (meaning the source and target tasks are the same), then the `populate()` method is called on the task component's Java part before the `process()` method.

Nesting Components

As we noted above, the component JSP part can include other components using the `syswf:component` custom tag right in the JSP code. The diagram shown in Figure 13 presents how a request is handled when there are such nested components. Note that now the request comes from the same task it is targeted to:

Figure 13. Nesting Components Diagram



1. The request is sent by the client browser from the same page as the page requested.

2. The `populate()` method is called on taskA component's Java part. This method is responsible for the transfer of data from web page form elements (input fields, radio buttons, etc.) to JavaBeans objects on the server.

3. The `process()` method is called on taskA component's Java part. This method should perform actions triggered by controls in the web page and/or prepare data for taskA component's JSP part.

4. Processing of taskA component's JSP part is initialized.

5. Request for insertion of component A is found.

6. The `process()` method is called on the Java part of component A. This method should prepare data for component presentation.
7. Processing of the JSP part of component A is performed. Once finished, the result is included in the parent JSP page.
8. Request for insertion of component B is found.
9. The `process()` method is called on the Java part of component B. This method should prepare data for component presentation.
10. Processing of the JSP part of component B is performed. Once finished, the result is included in the parent JSP page.
11. Processing of taskA component's JSP part finishes. The response is returned in the client's browser.

Component JSP Part

Example 17. Skeleton of the JSP Page

The following example displays the WSDL URL for a WSDL service.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@ taglib prefix="syswf" uri="http://systinet.com/jsp/syswf" %>

<syswf:page headerTemplate="pageHeader.jsp" footerTemplate="pageFooter.jsp">

    <syswf:wrap headerTemplate="design/pageHeader.jsp"
        footerTemplate="design/pageFooter.jsp">
    ...
    </syswf:wrap>

</syswf:page>
```

The core of the JSTL (standard tag library) together with the Registry Web Framework custom tag library are imported. The beginning of the page is declared (`syswf:page` tag); page header and footer represented as JSP pages are passed as attributes. These pages contain the basic HTML tags and declaration of Java Scripts that will be used in the page.

To enable automatic wrapping and resizing, all of the page's content is packed into the `syswf:wrap` tag to which page header and footer JSP pages are passed as attributes. The header and footer pages contain:

- The design part - the logo and menu, such as the labels at the top of this page under the product name
- The navigation path - shown in the top right corner of this page
- Text that should be displayed in the bottom of the page, such as copyright information.

Implicit Objects

Implicit objects allow you to interact with various framework parts, from Java code or JSP pages. A reference to an implicit object should be obtained from the `com.systinet.uddi.util.CallContext` class, or by using simple getter methods from `com.systinet.webfw.ComponentImpl`.

- **request** HTTP request interface; here you can read, for example, http headers included in user's request. Using request attributes is the preferred way to transfer data from Java to JSP pages.

- **response** HTTP response interface; can be used, for example, to set content type and other response header data or to send binary data back to client.
- **localSession** Contains the `java.util.Map` object, which is accessible from the current task only. For example, when you have tasks A and B in navigation history, each has a separate local session. When you return from task B to task A, the whole local session content of task B is discarded.
- **globalSession** Contains the `java.util.Map` object, which is shared among all tasks; this session can be used, for example, to store the current user's authToken, or other application-wide data.

Data Types

Data type classes are responsible for converting values between web page HTML form fields and underlying Java Beans objects. The Data type class must implement the simple interface `com.systinet.webfw.datatype.DataType` with two methods:

- `String objectToWeb(Object value)` provides conversion from arbitrary Java type to String usable in web pages.
- `Object webToObject(String value)` provides conversion in the opposite direction.

There are predefined implementations of this object for converting the simple Java data types string, int, long, and boolean.

Client-side Validators

Validators can be used to validate user input before a web page is submitted to a server. The validation is invoked by a specific page control (a button or a link). There is a predefined set of validators for common input field checks.

Table 54. Predefined Validators

Name	Description
required	Checks if the field is not empty.
uddiKey	Checks if the field content starts with the `uddi:` prefix.
length50, length80, length255, length4096, length8192	Checks if the field contains no more than the specified number of characters.
email	Checks if the field contains an email address.
long	Checks if the field contains a number of type long.
int	Checks if the field contains a number of type int.

To add a validator to an input field or a text area, use the sysfw:checker tag. To trigger the validation control, use the syswf:validate tag.

Example 18. Validators Usage

```
<syswf:input name="businessKey" value="">
    <syswf:checker name="required" action="viewBusinessV3"/>
    <syswf:checker name="uddiKey" action="viewBusinessV3"/>
</syswf:input>
...
<syswf:control action="viewBusiness" caption="View business" mode="button">
    <syswf:validate action="viewBusinessV3"/>
</syswf:control>
```

The Example 18, Validators Usage shows an input field with two checkers, the first one checks if the field is not empty and the second one checks if the field contains a string starting with the prefix `uddi:` (uddi key). Both checkers are invoked when a user clicks the **View business** button.

Validation is performed using a JavaScript function. The validator name is required to be defined in the JavaScript function with the name check_required. The return value from the validator is of the boolean type: true when the field content is valid, and false when content is invalid. In case of error, the validator displays an error message with the description of the allowed field content. This validator is also responsible for transferring the focus to the field with an error.

Example 19. Required Validator Implementation

```
// is required checker
function check_required (formID, fieldID)
{
  var value = getFieldValue(formID, fieldID);
  if (isEmpty(value))
  {
    alertRequired();
    setFocus(formID, fieldID);
    return false;
  }
  return true;
}
```

Custom validators should be can be added to the file `REGISTRY_HOME/app/uddi/web.jar/webroot/script/uddi.js`. Many functions for validation are defined in the file `REGISTRY_HOME/app/uddi/web.jar/webroot/script/wf.js`.

3.7.2. Directory Structure

JSP pages for the BEA AquaLogic Service Registry user interface are placed in the `REGISTRY_HOME/app/uddi/web.jar/jsp` directory. Static content, such as scripts and images, is stored in the `REGISTRY_HOME/app/uddi/web.jar/webroot` directory.

JSP Page Reference

Table 55. Root Files

File	Description
`error.jsp`	skeleton for error page
`home.jsp`	main page with welcome text
`login.jsp`	login page
`management.jsp`	page with buttons for all registry management tasks
`pageFooter.jsp`	page header containing required JavaScripts and HTML form. Do not write any design here; use `design/pageFooter.jsp` instead
`pageHeader.jsp`	contains mainly page hidden fields. Do not write any design here; use `design/pageHeader.jsp` instead
`uddiErrorComponent.jsp`	component responsible for displaying error messages

Table 56. Content of Page Directories

Directory	Description
`account`	All pages related to account management
`admin`	Administration tools for tModel deletion and key replacement
`approval`	Pages for approval process
`configuration`	Registry and web configuration pages
`custody`	User interface for custody transfer
`design`	Contains various design elements such as frames and tabs
`group`	Group management
`inquiry`	UDDI inquiry pages
`permission`	Permission management
`publishing`	UDDI publishing pages
`replication`	Replication management
`statistics`	Shows registry statistics
`subscription`	UDDI subscription pages
`taxonomy`	Taxonomy browsing and management
`util`	Various page components
`wsdl2uddi`	WSDL-to-UDDI mapping pages
`xml2uddi`	Inquiry and publishing pages for mapping of XML files to UDDI
`xsd2uddi`	Inquiry and publishing pages for mapping of XML schemas to UDDI
`xslt2uddi`	Inquiry and publishing pages for mapping of XSLT style sheets to UDDI

3.7.3. Framework Configuration

All needed configuration settings are stored in the file `REGISTRY_HOME/app/uddi/conf/web.xml`

Component

Specifies configuration of page components.

Table 57. Component Attributes

Attribute	Description	Required
`name`	Unique component identification	yes
`className`	Fully qualified class name of the component implementation class	no
`page`	Path to JSP page with component design; path is relative to root JSP directory.	no

Task

Contains definition of tasks.

Table 58. Task Attributes

Attribute	Description	Required
`URI`	Unique string used to call a task from controls or directly using http URL; the URI must start with a forward slash (/) character.	yes
`caption`	task description to be displayed, for example as page title	no
`component`	Name of task root component	yes

Table 59. Subelement

Element	Description	Required
`param`	Additional parameters to be passed to the root component; each parameter is specified as name-value pair.	no

Data Type

Contains the definition of the data types.

Table 60. Data Type Attributes

Attribute	Description	Required
`typeName`	Unique name of the data type; this name is used to reference a data type, for example from the syswf:input tag.	yes
`className`	Name of data type implementation class	yes

Other Configuration

Table 61. Configuration Elements

Element	Description
`url`	First part of the URL used to access BEA AquaLogic Service Registry without encryption (plain HTTP); this part should contain the http protocol prefix, hostname, and port.
`secureUrl`	First part of the URL used to access BEA AquaLogic Service Registry using encryption. This part should contain https protocol prefix, hostname and port.
`context`	Context part of the URL, used to access BEA AquaLogic Service Registry tasks; the default value is `uddi/web` for standalone registries and `wasp/uddi/web` for registries ported to an application server.
`dataContext`	Context part of the URL, used to access BEA AquaLogic Service Registry's static content, for example, images and cascading style sheets. The default value is `uddi/webdata` for standalone registries and `wasp/uddi/webdata` for registries ported to an application server.
`serverSessionTimeout`	Default timeout of server-side sessions (measured in seconds).
`uploadTempDir`	Directory used to store temporary files during the upload process; this path should be relative to service context directory.
`maxUploadSize`	Maximum size of uploaded files; larger files are rejected.
`jspDir`	Directory with JSP pages; the path should be relative to service context directory.
`jspEngine`	Contains JSP engine initialization parameters and the compilation classpath. A complete list of available Jasper initialization parameters can be found below.

Jasper Configuration

Table 62. Jasper init Configuration Parameters

Parameter name	Default value	Description
`checkInterval`	300	If the `development` parameter is false and `reloading` parameter is true, background compiles are enabled. `checkInterval` is the time in seconds between checks to see if a JSP page needs to be recompiled.
`compiler`	javac	Which compiler Ant should be used to compile JSP pages. See the Ant documentation for more information.
`classdebuginfo`	true	Indicates whether the class file should be compiled with debugging information
`development`	true	Indicates whether Jasper is used in development mode; checks for JSP modification on every access.
`enablePooling`	true	Determines whether tag handler pooling is enabled
`ieClassId`	clsid:8AD9C840-044E-11D1-B3E9-00805F499D93	The class-id value sent to Internet Explorer when using >jsp:plugin< tags.
`fork`	true	Tells Ant to fork compiles of JSP pages so that a separate JVM is used for JSP page compiles from the JVM.
`javaEncoding`	UTF8	Java file encoding to use for generating java source files.
`keepgenerated`	true	Indicates whether generated Java source code for each page is kept or deleted.
`logVerbosityLevel`	WARNING	The level of detailed messages to be produced by this servlet. Increasing levels cause the generation of more messages. Valid values are FATAL, ERROR, WARNING, INFORMATION, and DEBUG.
`mappedfile`	false	Indicates whether the static content is generated with one print statement per input line, to ease debugging.
`reloading`	true	Indicates whether Jasper checks for modified JSPs.

3.7.4. syswf JSP tag library

A JSP page using the syswf tag library must include this header `<%@ taglib prefix="syswf" uri="http://systinet.com/jsp/syswf" %>`

syswf:component

Includes the component with specified parameters.

Table 63. `syswf:component` Attributes

Attribute	Description	Required
`prefix`	All parameter names in component will be prefixed with this prefix; the prefix must be unique within each JSP page.	yes
`name`	Name of component, as written in the config file.	yes

Table 64. `syswf:component` Subelements

Element	Description	Required
`param`	When this parameter value is passed into a component, it will be accessible in the request scope in the component Java class and in the JSP page.	optional

The value of the parameter should be specified in two ways: As a value attribute or as a content of the value tag.

Example 20. Component Parameters

```
<syswf:component prefix="names" name="nameList">
   <syswf:param name="color1" value="white"/>
   <syswf:param name="color2">black</syswf:param>
</syswf:component>
```

syswf:page

Creates an HTML page form with all required internal fields. This must be the root element of all components used as tasks.

Table 65. `syswf:page` Attributes

Attribute	Description	Required
`headerTemplate`	The filename of the JSP page containing the page header, this file is designed to create elements required for framework functionality. Note that there should be no graphic design.	yes
`footerTemplate`	The filename of the JSP page containing the page footer, this file is designed to create elements required for framework functionality. Note that there should be no graphic design.	yes

syswf:wrap

This tag helps you to separate page functionality from its design. It includes specified header and footer templates before and after the body element. Header and footer templates should be parametrized using `syswf:param` tags.

Table 66. `syswf:wrap` Attributes

Attribute	Description	Required
`headerTemplate`	File name of JSP page containing the header.	no
`footerTemplate`	File name of JSP page containing the footer.	no

Table 67. `syswf:wrap` Subelements

Element	Description	Required
`param`	When you pass the parameter value into a component, this parameter will be accessible in the request scope in the component Java class and JSP page.	no

syswf:control

Creates a button or link, which should be used to trigger actions and transfers to other tasks.

Table 68. `syswf:control` Attributes

Attribute	Description	Required
`action`	Action to be passed to a control's parent component.	no
`mode`	Allowed values are `button`, `anchor`, `script`, or `image`. The script generates the `submit JavaScript` command, which can be used, for example, as a value for the HTML onClick attribute. Image is a graphic button.	yes
`targetTask`	URI of task to be called.	no
`targetDepth`	Specifies level in navigation path to be used.	no
`targetUrl`	Specifies the URL to be used to submit data; usable, for example, when you need to switch from http to https.	no
`caption`	control caption	required in anchor and button mode
`hint`	Help text, displayed as tooltip.	no
`disabled`	If set to `true`, button is disabled and link cannot be clicked.	no
`redirect`	If set to `true`, the task is only redirected to another task. This means that task data stored in a local session will also be accessible from the target task. Normal behavior is that a local session is not transferred between tasks.	no
`src`	Path to the image file used as graphic button.	required in image mode

Table 69. `syswf:control` Subelements

Element	Description	Required
`param`	Adds action parameters.	no
`attribute`	Adds attributes to created input or an HTML tag.	no

syswf:input

Inserts input field into JSP page.

Table 70. `syswf:input` Attributes

Attribute	Description	Required
`name`	Specifies the name of the accessible value of this input field.	yes
`value`	Specifies a value which appears in the input field, or a base object for the property attribute.	yes
`property`	Contains the property name of the object specified by the expression in the value attribute.	no
`hint`	Help text, displayed as a tooltip.	no
`dataType`	Data type which will be used to transform values between the underlying Java Bean object and the input field.	no
`disabled`	If set to `true`, the input field will be disabled.	no
`mode`	A possible value is password, used for password fields.	no

Table 71. `syswf:input` Subelements

Element	Description	Required
`attribute`	Appends a name and value pair as attribute to the resulting HTML tag; usable, for example, for the CSS class specification for an input field.	no

syswf:selectOne

Displays controls which enable the user to select one value from a list of available values.

Table 72. `syswf:selectOne` Attributes

Attribute	Description	Required
`name`	Specifies the name under which this value will be accessible; select one element.	yes
`mode`	Specifies visual style; possible values are `radio`, `check box`, and `menu`.	no
`value`	Specifies a value which will be selected, or a base object for the property attribute.	yes
`property`	Contains the property name of the object specified by expression in the value attribute.	no
`optionValues`	Specifies a comma-delimited list of available values, the expression of which evaluates either to String[], or to an array of object for the optionValuesProperty attribute.	yes
`optionValuesProperty`	Contains property name of objects specified by expression in the optionValues attribute.	no
`optionCaptions`	Specifies a comma-delimited list of available captions, the expression of which evaluates either to String[], or to an array of object for the optionCaptionsProperty attribute.	no
`optionCaptionsProperty`	Contains property name of objects specified by expression in the optionCaptions attribute.	no
`hint`	Help text, displayed as tooltip.	no
`dataType`	Data type which will be used to transform values between the underlying Java Bean object and the selected element.	no

Table 73. `syswf:selectOne` Subelements

Element	Description	Required
`attribute`	Appends a name/value pair as an attribute to resulting HTML tags.	no

syswf:selectMany

Displays controls which enable the user to select multiple values from list of available values.

Table 74. `syswf:selectMany` Attributes

Attribute	Description	Required
`name`	Specifies the name under which the value of this selectMany element will be accessible.	yes
`mode`	Specifies visual style possible values `check`, `box` and `menu`.	no
`value`	Specifies an array of values which will be selected, or base objects, for the property attribute.	yes
`property`	Contains property name of objects specified by expression in the value attribute.	no
`optionValues`	Specifies a comma-delimited list of available values the expression of which evaluates to String[], or to an array of object for the optionValuesProperty attribute.	yes
`optionValuesProperty`	Contains the property name of objects specified by expression in the optionValues attribute.	no
`optionCaptions`	Specifies a comma-delimited list of available captions, the expression of which evaluates to either String[], or to an array of object for the `optionCaptionsProperty` attribute.	no
`optionCaptionsProperty`	Contains a property name for objects specified by expression in the `optionCaptions` attribute.	no
`hint`	Help text, displayed as tooltip.	no

Table 75. `syswf:selectMany` Subelements

Element	Description	Required
`attribute`	Appends a name/value pair as an attribute to result HTML tags.	no

syswf:textArea

Creates a text area HTML component.

Table 76. `syswf:textArea` Attributes

Attribute	Description	Required
`name`	Specifies the name under which the value of this text area will be accessible.	yes
`value`	Specifies a value which appears in the text area, or a base object for the property attribute.	yes
`property`	Contains a property name of an object specified by expression in the value attribute.	no
`hint`	Help text, displayed as tooltip.	no
`dataType`	Data type which will be used to transform values between underlying the Java Bean object and the text area.	no
`disabled`	If set to `true`, the text area will be disabled.	optional

Table 77. `syswf:textArea` Subelements

Element	Description	Required
`attribute`	Appends a name/value pair as an attribute to the result HTML tag; usable, for example, for CSS class specification for the text area.	no

syswf:value

Evaluates the given expression and transform result using data type.

Table 78. `syswf:value` Attributes

Attribute	Description	Required
`value`	Specifies the expression which will be evaluated.	yes
`hint`	Help text, displayed as tooltip.	no
`dataType`	Data type which will be used to transform value.	no

syswf:size

This tag will fill the page attribute with size of given List, UDDIList, StringArrayList or Array.

Table 79. `syswf:size` Attributes

Attribute	Description	Required
`var`	Name of variable to store the size of a given list or array.	yes
`value`	Specifies an expression to be evaluated; the result must be List, UDDIList, StringArrayList or Array.	yes
`scope`	Scope of the variable to store the size of a given list or array. Allowed values are `request`, `session`, `application`, or `default`.	no

navigationPath

This component renders the history path (bread crumbs links)

navigationPath component in action

Example 21. Component Parameters

```
<syswf:component name="navigationPath" prefix="path"/>
```

3.7.5. Typical Customization Tasks

- **Q: Where can I find the code which generates the page header?** A: It is defined in the file `design/pageHeader.jsp`.
- **Q: How do I change the text displayed on a page's title bar?** A: Modify content of <title> tag in the file `pageHeader.jsp`.
- **Q: Where is the right place to include my own JavaScript files?** A: Reference to your files should be placed in `pageHeader.jsp`. Place your script files in the `REGISTRY_HOME/app/uddi/web.jar/webroot/script` directory.

- **Q: Where is it possible to change the text displayed in the page footer?** A: The page footer is defined in the file `design/pageFooter.jsp`.

3.8. Business Service Console Framework

This section describes the Business Service Console (BSC) from the developer's point of view. It describes the Business Service Console Framework architecture and configuration, and demonstrates how to customize the console.

The Business Service Console implementation and configuration are contained in the JAR file `bsc.jar` located in directory `REGISTRY_HOME/app/uddi`.

This section has the following subsections:

Section 3.8.1, Business Service Console Localization How to localize the Business Service Console, or the Registry Console.

Section 3.8.2, Directory Structure The directory structure of `bsc.jar`.

Section 3.8.3, Business Service Console Configuration Business Service Console configuration files in `bsc.jar`.

Section 3.8.5, Permission support Features to establish whether users have permission to perform operations.

Section 3.8.6, Components and Tags Components and tags in `bsc.jar` used to develop Business Service Console components.

3.8.1. Business Service Console Localization

BEA AquaLogic Service Registry is ready for localization. This chapter is focused on localization of web applications such as the Business Service Console and Registry Console. It provides information on BEA AquaLogic Service Registry localization support and how to write localizable web applications.

Basic concepts

The localization support is built upon standard Java resource bundles and the JSP formatting tag library.

Locale detection

The user language-detection routine is invoked for each HTTP request. When the user is logged in, the userAccount's languageCode is used, if it is set. Otherwise the browser's preferred language is used. The system then finds the resource bundle for the chosen locale or uses a default resource bundle, if there is no such localized resource bundle. See the ResourceBundle javadocs for details of the algorithm.

The system uses UTF-8 encoding by default, but it can be configured to use a custom locale-encoding mapping in the file `web.xml`:

```
<webFramework>
    <encoding>
        <map locale="en" encoding="UTF-8"/>
        <map locale="zh" encoding="Big5"/>
    </encoding>
</webFramework>
```

Resource bundles

There is one resource bundle common to all JSP files serving as a dictionary - `com.systinet.uddi.bui.standard.BUIMessages`. It contains keys for common words like "OK", "Cancel" or names of entities (Provider, Service). Then each top-level directory in the `jsp` directory has a unique resource bundle for its files and subdirectories. The resource bundles for

Business Service Console are located within the `src` directory and are copied to the `WASP-INF/classes` directory during build phase.

Resource keys naming convention

The resource key is composed of JSP file name (without suffix) and an English identifier in camel notation. (Capital letters are used to indicate the start of words, instead of a space or underscore.) If the JSP file is located in some subdirectory of the top-level directory, the subdirectory name is also encoded in the resource key. For example resources for JSP file `search/interfaces/simple.jsp` are stored in the file `com.systinet.uddi.bui.standard.component.search.SearchMessages.properties` and all keys have the prefix *interfaces.simple_*.

In some configuration files it is necessary to use a custom resource bundle instead of the default bundle. There is a way to encode the custom resource bundle name into the resource key. If the resource key contains the character *$*, then the part before it will be treated as the resource bundle identifier and the rest of the resource key as actual resource key. For example *customBundle$resourceKey*.

Localization of Configuration

The configuration files are localizable too. For example the file `conf/bsc.xml` has texts in the resource bundle `com.systinet.uddi.bui.framework.BSCMessages.properties`. The attributes like `caption` and `hint` have their localizable alternatives `captionKey` and `hintKey`, which have precedence over the original attributes providing text. The exception to this rule is the `task` element in the file `conf/web_component.xml`, where `caption` attribute has precedence over new `captionKey` attribute.

JSP localization

The localization of JSP files uses the standard formatting tag library. Every JSP must start with import of this library and setting of the locale for the current user, if he is logged in. The user's language is stored in the session variable `userDefaultLanguage`.

Example 22. Example of localization

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<c:if test="${not empty globalSession['userName']}">
   <fmt:setLocale value="${globalSession['userDefaultLanguage']}" scope="page"/>
</c:if>
<fmt:setBundle basename="com.systinet.uddi.bui.standard.component.search.SearchMessages"
var="search_Message"/>
<fmt:message key="interfaces.simple_operationProperty" bundle="${search_Message}"/>
```

In addition to the full power of the standard formatting library there are several Systinet extensions that complement localization needs.

ParseResourceKey tag

The parseResourceKey tag is used, when the resource key can contain an embedded resource bundle. It detects such a situation and introduces two new variables that will hold the values of resource bundle and resource key to be used.

Table 80. `ParseResourceKey` tag Parameters

Param	Description	Required
`key`	The resource key that may contain an embedded custom resource bundle.	yes
`defaultBaseName`	Default resource bundle to use if no custom bundle is detected.	yes
`varBundle`	Name of variable that will hold the name of the bundle for this resource.	yes
`varResource`	Name of variable that will hold resource key.	yes

Example 23. ParseResourceKey tag - Usage Example

```
<syswf:parseResourceKey key="${captionKey}"
    defaultBaseName="com.systinet.uddi.bui.framework.WebComponentMessages"
    varBundle="bundleName" varResource="finalCaptionKey"/>
<fmt:setBundle basename="${bundleName}" var="dynamic_Message"/>
<fmt:message key="${finalCaptionKey}" var="dialogCaption" bundle="${dynamic_Message}"/>
```

LocalizedFileName tag

LocalizedFileName tag finds the name of the localized file for the current locale. It uses the same heuristic search as resource bundle loading. For example if there is a file `scripts.js` and the french locale is set, then `scripts_fr.js` may be returned.

Table 81. `localizedFileName` tag Parameters

Param	Description	Required
`basedir`	Prefix to be concatenated to fileName to access a resource from the servlet context.	yes
`fileName`	Name of file whose localized version is needed.	no
`var`	Name of variable that will hold the file name for the current locale.	yes

Example 24. localizedFileName - Usage Example

```
<syswf:localizedFileName basedir="/../webroot/" fileName="js/bui.js" var="jsBui"/>
                <script language="JavaScript" src="<c:out value="${jsBui}"/>"></script>
```

LocalizedInclude tag

Sometimes it is necessary to localize very long text and it would not be practical to store it in a resource bundle as a key, especially when the text contains formatting information. For this purpose there is a tag `localizedInclude`, which writes to output the content of file selected in the current locale. The rules for file selection are same as for Resource bundles.

Table 82. `localizedInclude` tag Parameters

Param	Description	Required
`baseName`	Path to resource with the text to be written to output.	yes

Example 25. localizedInclude - Usage Example

```
<syswf:localizedInclude baseName="publish/service/generic/selectInterfaces.html"/>
```

Java localization

The localization of web applications uses standard resource bundles. It is necessary to use `com.systinet.web-fw.util.BundleHelper` instead of `java.util.ResourceBundle` to retrieve a resource bundle otherwise different rules for locale selection will be used in the java code and JSP files, which results in page with portions in different languages.

3.8.2. Directory Structure

The following table summarize the directories inside `bsc.jar`.

Table 83. `bsc.jar` Directories

Directory	Description
conf	Configuration files of the Business Service Console. See Section 3.8.3, Business Service Console Configuration
jsp	JSP files
src	Source Java files
WASP-INF	Compiled Java and JSP classes, libraries, and Systinet Server for Java-related configuration files
webroot	Static content of Business Service Console pages such as HTML, Javascript, graphics and CSS.

The `bsc.jar` package depends on the UDDI-service package. So services in the UDDI-service package are available to Business Service Console developers.

If you want to edit and modify any of the Business Service Console's source JSP or Java files, perform the following steps:

1. Unzip `bsc.jar` to a temporary location.
2. Edit the source files.
3. Compile the Java sources against the libraries in the `REGISTRY_HOME/lib` directory and the client libraries from the `REGISTRY_HOME/dist` directory.
4. Copy the resulting `.class` files into the `WASP-INF/classes` directory of the unzipped JAR.
5. Stop BEA AquaLogic Service Registry
6. To preserve any changes made to the Business Service Console configuration at runtime, copy the contents of directory `REGISTRY_HOME/work/uddi/bsc.jar/conf` to the `conf` directory of the unzipped JAR.
7. Zip the JAR again and deploy it over the original file in the `REGISTRY_HOME/app/uddi` directory.

If you intend to change the JSP files only for testing purposes, you do not have to redeploy the `bsc.jar`. It is sufficient to modify the JSP files in `REGISTRY_HOME/work/uddi/bsc.jar/jsp`. You must reload pages in the browser before any change is visible. Note that files under `REGISTRY_HOME/work` are liable to be overwritten or deleted when packages are re-deployed.

`conf` Directory

This directory contains the following configuration files:

Table 84. `conf` Directory Contents

File	Description
`bsc.xml`	The Business Service Console configuration file. This contains the configuration of tabs, user profiles, URLs, paging limits, enterprise classifications, and settings for the approval process and subscription components. Also API endpoints and a flag determining whether SOAP communication is used for these. See Section 3.8.3, Business Service Console Configuration.
`web.xml`	The deployment configuration file. This contains Business Service Console deployment information such as web interface URLs and contexts. It also defines the location of JSP files, their pre-compiled versions and declared libraries for the JSP engine.
`web_component.xml`	The web framework configuration file. This contains the web framework's static settings including definitions of components, tasks and data types, and configuration for menus, context menus, trees and customizable taxonimies.
`component_description.xml`	This describes components in terms of their roles, relationships and interfaces.

`jsp` directory

This directory contains the JSP files that constitute the base of the Business Service Console and the following subdirectories:

Table 85. `jsp` Directory Contents

Directory	Contents (JSP files)
`account`	Account management
`approval`	Approval process interface (part of tools section).
`browse`	Report section of console, includes also entity details pages
`catalog`	Catalog section of console
`common`	Common pages for table component actions
`configuration`	Content of configuration section
`design`	Design including miscellaneous page and frame headers and footers
`editor`	Component editor components
`publish`	Catalog section of the console
`query`	Query framework components
`search`	Search section of console
`table`	Table framework components
`taxonomy`	Taxonomy framework components
`tools`	Tools section components
`util`	Utility components such as navigationPath
`view`	Entity list view pages of console
`WEB-INF`	Configuration files for JSP pages including declaration of use, tag libraries, etc.
`wizard`	The wizardIterator framework component

`src` directory

This directory contains the source files of the Business Service Console

Table 86. `src` Directory Contents

Enclosing Package	Description
`com.systinet.uddi.bui.framework`	Source Java files for the Business Service Console framework.
`com.systinet.uddi.bui.standard`	Source Java files for Business Service Console default implementation

`WASP-INF` directory

This directory contains the `package.xml` file for the Business Service Console, and the subdirectories listed in the following table:

Table 87. `WASP-INF` Subdirectories

Directory	Contents
`classes`	Compiled Java classes of the Business Service Console (including the Java parts of components and several utility classes)
`jsp-classes`	Pre-compiled JSP pages (JSP parts of components) from the `jsp` directory
`lib`	Libraries for the web application, including JSP, JSTL supporting libraries, etc.

`webroot` Directory

Contains subdirectories listed in the following table:

Table 88. `webroot` Subdirectories

Directory	Contents
`gui`	Resource files such as CSS, graphics, HTML
`gfx`	A deprecated directory that contained miscellaneous graphic files such as icons, logos, etc.
`script`	A deprecated directory that contained Java Scripts and the `bui.css` file for the Business Service Console

3.8.3. Business Service Console Configuration

The `bsc.jar` file in directory `REGISTRY_HOME/app/uddi` contains the configuration files for the entire Business Service Console. They are located in the `conf` subdirectory, the contents of which were summarized in Directory Structure. In this section, we focus on the file `bsc.xml`.

- Section BEA AquaLogic Service Registry API Endpoint URL
- Section Result Filtering
- Section Main Menu Tabs
- Section User Profiles
- Section Entity List Views
- Section Browsable Taxonomies

- Section Paging Limits

BEA AquaLogic Service Registry API Endpoint URL

This configuration part contains the endpoint URLs used by the Business Service Console to communicate with BEA AquaLogic Service Registry:

```
<url>http://localhost:8080</url>
<secureUrl>https://localhost:8443</secureUrl>
<useSoap>false</useSoap>
<uddiEndpoints accountApiPath="/uddi/account" approverApiPath="/uddi/approver"
    categoryApiPath="/uddi/category" configuratorApiPath="/uddi/configurator"
    inquiryPath="/uddi/inquiryExt" inquiryUIApiPath="/uddi/inquiryUI"
    publishingPath="/uddi/publishingExt" requestorApiPath="/uddi/requestor"
    securityPath="/uddi/security" subscriptionPath="/uddi/subscriptionExt"
    taxonomyApiPath="/uddi/taxonomy" wsdlApiPath="/uddi/wsdl2uddi"
    xml2UddiApiPath="/uddi/xml2uddi" xsd2UddiApiPath="/uddi/xsd2uddi"
    xslt2UddiApiPath="/uddi/xslt2uddi"/>
```

The endpoint URL is composed of two parts:

- The prefix, taken from the `url` element (or `secureURL` element for a secure endpoint)
- The relative part, taken from the specified `uddiEndpoints` attribute (depending on the type of the endpoint).

If you want to use a different target registry, it is usually sufficient to change the prefix (the absolute part of the URL).

Important

The `useSoap` element indicates whether to use SOAP to access BEA AquaLogic Service Registry, or to ignore the declared API endpoints and make the calls directly through the Java virtual machine.

Result Filtering

Use this section to filter data that should not be displayed in the Business Service Console. For example, to hide the Operational business entity, you can set up a filter in the `businessUI` element. Note that child elements of an element you filter will not be displayed in the Business Service Console.

The following sample will hide the Operational business entity in the Business Service Console:

```
<filteredKeys>
      <businessKey>uddi:systinet.com:uddinodebusinessKey</businessKey>
</filteredKeys>
```

Main Menu Tabs

```
<tab tabId="home" taskId="/home">
    <captionKey>bsc.tab_home</captionKey>
    <hintKey>bsc.tab_homeHint</hintKey>
</tab>
<tab tabId="search" taskId="/search">
    <captionKey>bsc.tab_search</captionKey>
    <hintKey>bsc.tab_searchHint</hintKey>
</tab>
<tab tabId="catalog" taskId="/publish">
```

```
    <captionKey>bsc.tab_catalog</captionKey>
    <hintKey>bsc.tab_catalogHint</hintKey>
</tab>
<tab tabId="tools" taskId="/tools">
    <captionKey>bsc.tab_tools</captionKey>
    <hintKey>bsc.tab_toolsHint</hintKey>
</tab>
<tab tabId="report" taskId="/browse">cat
    <captionKey>bsc.tab_reports</captionKey>
    <hintKey>bsc.tab_reportsHint</hintKey>
</tab>
<tab tabId="configure" taskId="/configure">
    <captionKey>bsc.tab_configure</captionKey>
    <hintKey>bsc.tab_configureHint</hintKey>
</tab>
```

Each `tab` element contains the definition of one main menu tab. These tabs are rendered in the top left area of the Business Service Console, under the product logo.

Important

Some navigation tabs should not be visible in all cases. Tab visibility depends on the profile selected for the current user. The BEA AquaLogic Service Registry administrator should define tab visibility using the Configuration main menu tab.

Table 89. `Tab` element attributes

Attribute	Description	Required
`captionKey`	Resource bundle key to caption visible for users.	yes
`tabId`	Unique tab identification.	yes
`taskId`	URI of the task called when a user clicks on this tab.	yes
`hintKey`	Resource bundle key to descriptive text displayed as tab tooltip.	no

User Profiles

```
<profile defaultTab="home" profileId="default" captionKey="bsc.profile_anonymousUserProfile">
    <visibleTab>search</visibleTab>
    <visibleTab>report</visibleTab>
    <visibleTab>home</visibleTab>
    <defaultView viewId="Common" viewType="providers"/>
    <defaultView viewId="Common" viewType="interfaces"/>
    <defaultView viewId="Common" viewType="endpoints"/>
    <defaultView viewId="Common" viewType="bindings"/>
    <defaultView viewId="Common" viewType="services"/>
</profile>
```

Each `profile` element contains the definition of one user profile. The user profile defines the visibility of tabs, default navigation tab, and default views for various Business Service Console entity lists.

Users are able to change their profiles using the **My Profile** link from the Home page, unless an administrator has prohibited this via the **Configure** tab.

Table 90. `Profile` Element Attributes

Attribute	Description	Required
`caption`	Tab caption visible for users.	yes
`profileId`	Unique profile identification.	yes
`defaultTab`	TabId of the tab which will be displayed after user login. This attribute must contain the identification of one of the visible tabs defined for this profile.	yes
`visibleTab`	Id of navigation which will be visible for this user profile.	at least one
`defaultView`	Specifies the `viewId` of the view used as default when the user enters a page with a result list. The attribute `viewType` defines the type of list and `viewId` defines view identification.	one for each viewType

Entity List Views

```
<!-- list of available views -->
<view captionKey="bsc.view_business" viewId="Business"/>
<view captionKey="bsc.view_common" viewId="Common"/>
<view captionKey="bsc.view_technical" viewId="Technical"/>
<view captionKey="bsc.view_operation" viewId="Operation"/>

<!-- list of available view types -->
<viewType captionKey="bsc.viewType_providers" viewTypeId="providers"/>
<viewType captionKey="bsc.viewType_services" viewTypeId="services"/>
<viewType captionKey="bsc.viewType_endpoints" viewTypeId="endpoints"/>
<viewType captionKey="bsc.viewType_interfaces" viewTypeId="interfaces"/>
<viewType captionKey="bsc.viewType_bindings" viewTypeId="bindings"/>
```

The `view` element defines a list of available views. The `viewType` element defines a list of available entity list types.

Important

The viewId and viewType values are used to determine which component will be used for view presentation. The Business Service Console automatically checks the existence of all available view components, and only existing views will be presented to user. The View component name has the following format: `[viewType][viewId]Results`. For example, the component rendering the business view on a list of providers is named `providersBusinessResults`

Browsable Taxonomies

```
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_usage" tModelKey="uddi:systinet.com:tax-
onomy:usage"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_endpoint" tModelKey="uddi:systinet.com:tax-
onomy:endpoint:status"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_status" tModelKey="uddi:systinet.com:tax-
onomy:interface:status"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_namespace" tMod-
elKey="uddi:uddi.org:xml:namespace"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_localName" tModelKey="uddi:uddi.org:xml:loc-
alName"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_certification" tModelKey="uddi:systin-
et.com:taxonomy:service:certification"/>
```

```
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_availability" tModelKey="uddi:systin-
et.com:taxonomy:endpoint:availability"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_wsiCompliance" tModelKey="uddi:65719168-
72c6-3f29-8c20-62defb0961c0"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_milestone" tModelKey="uddi:systinet.com:ver-
sioning:milestone"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_releaseDate" tModelKey="uddi:systin-
et.com:versioning:releasedate"/>
<browsableTaxonomy captionKey="bsc.browsableTaxonomy_version" tModelKey="uddi:systinet.com:ver-
sioning:version"/>
```

This section holds information about the list of browsable taxonomies. Each taxonomy is displayed as one node in the Business Service Console **Reports** tab. This list is also used when displaying the **Classifications** tab on entity detail pages.

Table 91. `BrowsableTaxonomy` Attributes

Attribute	Description	Required
`caption`	Taxonomy display name used when rendering the navigation tree on the **Report** tab.	yes
`tModelKey`	The key of the taxonomical tModel in BEA AquaLogic Service Registry.	yes

Paging Limits

```
<pagingLimits component="resourcesXsltResults" pageSize="10" pageCount="20"/>
<pagingLimits component="resourcesWsdlResults" pageSize="10" pageCount="20"/>
<pagingLimits component="default" pageSize="10" pageCount="20"/>
```

In this section, the limits of rows displayed on one page and the number of pages displayed are defined. Each component should define its specific settings, or that default values are used when settings are not found.

3.8.4. Entity Configuration

In this section, we will explain how the Business Service Console can be configured to recognize UDDI data as Business Service Console Entities and how to hook into standard actions for those entities.

Overview

The UDDI specification recognizes 4 entity types. However Business Service Console needs to present the UDDI data in terms of the user's business. Because of various mappings of resources, business artifacts etc. to UDDI, a single UDDI entity may correspond to several business-level entities.

The Entity Configuration defines how to recognize individual business artifacts. UDDI *categorization* is used to annotate the UDDI data with information about their role or type. So for example, a tModel is just a *resource* for the Business Service Console GUI. But when the tModel contains, for example, the `uddi:uddi.org:resource:type` category with value `xslt`, it represents an *XSL Transformation document*. Depending on business needs for the artifacts, different presentations, actions, or relationships may be available.

The Entity Configuration further specifies *Views* to be used for the particular entity. A View is a web page, or a portion of it, customized for the particular Entity. Views correspond to abstract operations that make sense on several entities. For example, pages for **Delete** action are different for Services and Providers but both pages perform the same abstract operation.

Configuration

The configuration is stored in the file `conf/bsc.xml`, inside the `entityViews` element. New Entities can be created by adding new Entity definitions to that configuration, or behaviour of existing Entities can be changed. In this release, we support creation of new Entities and customization of Entities based on tModels (TM).

The following example shows a commented configuration of a "Categorization" entity, derived from a tModel. It corresponds to a taxonomy tModel used by the Registry. The individual parts of the configuration will be described below

Example 26. Definition of an XML document

```
<!--
    Definition of a new entity, based on a TModel (TM). We specify an icon,
     a (localizable) name (caption) and (localizable) description.
-->
<entity entityId="xsd" type="TM" icon="xsd.gif"
   captionKey="bsc.entityViews_xsd_caption" descriptionKey="bsc.entityViews_xsd_description">

    <!--
        Categorization together with "type" attribute (above) tells the
        framework how to identify this type of entities
    -->
    <categorization>
        <keyedReference tModelKey="uddi:uddi.org:resource:type" keyValue="xsd"/>
    </categorization>

    <!--
        Views tells the BSC which components and tasks should be used
        to display information about the entity or to manipulate with
        the entity
    -->
    <views>
        <view type="list" task="/browse/resources/xsds">
            <parameter paramName="entityId" paramValue="${entityId}"/>
            <parameter paramName="editableMode" paramValue="${editableMode}"/>
        </view>
        <view type="listMy" task="/browse/resources/xsds">
            <parameter paramName="entityId" paramValue="${entityId}"/>
            <parameter paramName="editableMode" paramValue="${editableMode}"/>
            <parameter paramName="filterMyEntities" paramValue="true"/>
        </view>
        <view type="create" task="/publish/resources/xsds/createXSDResource">
           <parameter paramName="requiredCategories" paramValue="${categoryBag.KeyedReference-
ArrayList}"/>
        </view>
        <view type="edit" task="/publish/resources/xsds/editXSDResource">
            <parameter paramName="tModelKey" paramValue="${entityKey}"/>
        </view>
        <view type="find" task="/search/resources/schemas">
            <parameter paramName="editableMode" paramValue="${editableMode}"/>
        </view>
        <view type="searchResults" component="resourcesXsdResults">
            <parameter paramName="query" paramValue="${query}"/>
            <parameter paramName="var" paramValue="${var}"/>
        </view>
        <view type="detail" task="/browse/xsdDetail">
            <parameter paramName="tModelKey" paramValue="${entityKey}"/>
        </view>
        <view type="treeContextMenu" component="contextMenu_xsdList"/>
        <view type="pageMenu" task="/catalog/xsdMenu"/>
        <view component="xsdsSubscriptionView" type="subscriptionChangeView"/>
        <view type="delete" task="/publish/resources/xsds/unpublishXSDResource">
            <parameter paramName="tModelKey" paramValue="${entityKey}"/>
```

```
        </view>
    </views>

    <!--
        References defines how to make associations with this type of entity,
        what keyedReferences to use and who can make the association
    -->
    <references>
        <!-- One or more references, leading to this entity type. -->
        <reference refName="schema"
                 captionKey="bsc.entityViews_xsd_refSchema_caption"
                 descriptionKey="bsc.entityViews_xsd_refSchema_description">
            <!-- originTypes may be either entityIds, or UDDI entity types.
                 No origin means all entities match
            <originType>xml</originType>
            -->
            <originType>xml</originType>
           <keyedReference tModelKey="uddi:uddi.org:resource:reference" keyName="definition"/>

        </reference>
        <reference refName="schemaOfSource"
                captionKey="bsc.entityViews_xsd_refSchemaOfSource_caption"
                descriptionKey="bsc.entityViews_xsd_refSchemaOfSource_description">
            <originType>xslt</originType>
            <keyedReference tModelKey="uddi:uddi.org:resource:reference"
                keyName="transformation-source"/>
        </reference>
        <reference refName="schemaOfDestination"
                captionKey="bsc.entityViews_xsd_refSchemaOfDestination_caption"
                descriptionKey="bsc.entityViews_xsd_refSchemaOfDestination_description">
            <originType>xslt</originType>
            <keyedReference tModelKey="uddi:uddi.org:resource:reference"
                keyName="transformation-destination"/>
        </reference>
        <reference refName="dependencyOnXSD"
                captionKey="bsc.entityViews_dependencyOnXSD_caption"
                descriptionKey="bsc.entityViews_dependencyOnXSD_description">
            <keyedReference tModelKey="uddi:systinet.com:dependency" keyName="tModel"/>
        </reference>
    </references>
</entity>
```

Entity Definition

The Business Service Console Entity definition element introduces a new Entity recognized by the Business Service Console. The entity has an *id*, a title, an optional description and an icon.

Table 92. Entity definition attributes

Attribute	Description	Required
`entityId`	An unique identifier that identifies this entity type. It should start with lowercase letter, and use only alphanumeric characters.	Yes
`captionKey`	This string serves as a key to the resource bundle, which stores to actual string used for the entity caption. See below regarding handling of singular and plural forms.	Yes
`descriptionKey`	The key into the resource bundle, for the string that provides a short decription of the entity type. The description may contain HTML markup.	No

When the Business Service Console needs to print a noun, that describes a collection of entities, it uses the string denoted by the `captionKey` resource bundle key. In the case the Business Service Console needs to print a *singular* noun, which stands for the entity type, it uses the key with `_single` suffix. All strings are taken from the resource bundle `src/BSCMessages.properties`, unless specified otherwise by the `captionKey` attribute (see Section 3.8.1, Business Service Console Localization for details).

The `icon` attribute is relative to directory `webroot/gfx/tree`. The icon is displayed in navigation trees to provide an unique visual appearance for the entity type.

Example 27. Definition of a XML document

```
<!--
    The "XML Document" entity is derived from UDDI TModel.

    The definition also defines what name and icon should display for this
    type of data.
-->
<entity entityId="xml" type="TM" icon="xml.gif"
   captionKey="bsc.entityViews_xml_caption"  descriptionKey="bsc.entityViews_xml_description">

    <categorization>
        <!--
            "XML Documents" are characterized by a keyedReference for the
            uddi:uddi.org:resource:type taxonomy, with "xml" value.
        -->
        <keyedReference tModelKey="uddi:uddi.org:resource:type" keyValue="xml"/>
    </categorization>
</entity>
```

Entity Categorization

As noted in the overview, an UDDI data structure may be used to represent several abstractions - Entities. An *Entity* is characterized by two things:

- *basic UDDI type*
- *categorization*

The basic UDDI type is one of:

BE

Business Entity

BS

Business Service

BT

Binding Template

TM

tModel

The UDDI entity needs to be of the specified type in order to be recognized as the particular BSC Entity. In addition, you may specify mandatory `keyedReferences`, which the UDDI entity needs to have. The BSC Entity that has most `keyedReferences` matching the UDDI data will be selected. If there remains a choice, one is chosen at random.

Zero or more `keyedReferences` can be specified. When no categorization is present, all appropriate UDDI structures match, regardless of their contents. When specified, each `keyedReference` entry can have the following attributes:

Table 93. `keyedReferenceAttributes`

Attribute	Description	Required
`tModelKey`	A tModel key of the *taxonomy* used for categorization	yes
`keyName`	The `keyName` of the required `keyedReference`. If the attribute is omitted, keyNames are ignored.	no
`keyValue`	The `keyValue` of the required `keyedReference`. If the attribute is omitted, keyValues are ignored (any matches).	no

Example 28. Definition of a XML document

```
<!--
    This is a definition of a WSDL service. It is derived from the Business
    Service UDDI structure (BS)
-->
<entity entityId="service" type="BS"  icon="service.gif"
    captionKey="bsc.entityViews_service_caption"
    descriptionKey="bsc.entityViews_service_description">
    <categorization>
        <!--
            A WSDL service is characterized by having the
            "uddi:uddi.org:wsdl:types" category with "service" value,
            according to the WSDL to UDDI mapping Technical Notes
        -->
        <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
                        keyValue="service"/>
    </categorization>
</entity>

<!--
    This is a specification of a XSL Transformation entity. It is derived from
    a TModel UDDI structure (TM)
-->
<entity entityId="xslt" type="TM" icon="xslt.gif"
    captionKey="bsc.entityViews_xslt_caption"
    descriptionKey="bsc.entityViews_xslt_description">
    <categorization>
        <!--
            The XSLT resource is characterized by the resource:type
            category which must have the "xslt" value, according to
            the proposed mapping Technical Note.
        -->
        <keyedReference tModelKey="uddi:uddi.org:resource:type"
                        keyValue="xslt"/>
    </categorization>
</entity>
```

Note

There must be an uncategorized Entity defined for each of the UDDI structures, to serve as a "catch-all" for data that does not match any specific entity. The default Business Service Console configuration provides such Entities.

Entity Views

A *View* stands for a visualization of some aspect, or an abstract task, that is available for the entity. Some tasks may or may not be available, depending on whether an appropriate View is available for the rendered data. The Business Service Console implementation uses View definitions to lookup tasks and components, which are appropriate for handling the data presentation, or to perform operations on the data.

A View is identified by a `viewType`. There can be at most one View for the particular `viewType` defined for the given entity. If such View is not defined, the Entity does not support the relevant visualization, or operation. The following table summarizes the supported `viewTypes`.

Table 94. Predefined View types

viewType	Description	Required
searchResults	Embeddable component, that provides a search results for a given type of Entity. The Component should accept a query, and render the matching results on the screen. These Components are used in Reports, Quicksearch etc.	No
edit	Task for editing a specific entity. The task accepts an entity key, and produces a screen (form, wizard) suitable for editing the entity.	No
detail	Provides a task that displays detailed information for an entity. The task accepts the key of the entity to display. This View is mandatory to ensure that information about any entity can be reached.	Yes
find	Provides a searching task for the entity. The task is supposed to display a form and results of the search.	No
subscriptionChangeView	Provides a Component to render subscription results for the particular entity type. The Component accepts the list of subscriptions to filter and display as a parameter	No
create	Provides a Task with a Wizard or a form to create a new entity. The Task may process a parameter that identifies a parent structure where the new entity should be stored.	No
delete	Provides a Task for deleting the entity. The Task should accept a single key, or a collection of keys as a parameter, and it should handle deletion of a single or several entities.	No
list	Provides a task, which displays all entities of the particular type. The Task accepts a parameter, which turns edit functions on/off. These tasks should not require login.	No
listMy	Provides a task, which displays all entities of the type owned by the logged-in user; otherwise, the function is just as with the `list` view.	No
treeContextMenu	Provides a context menu for the Catalog tree. If missing, there will not be a context menu for the entity.	No
pageMenu	Provides a Task that displays the entity's menu when the Entity is selected in the Catalog tree. If missing, the entity will not be shown in Catalog at all.	No

Each view can take some parameters. The parameters are passed by the code that invokes the View, and the framework passes them to the View's implementation component or task. The caller must be able to use the same parameters for invoking a View on different Entity types to remain independent of implementation details of individual Entities. To achieve this, the View definition not only contains parameter names, but also uses a simple mechanism to translate View's parameters to the implementation Component or Task parameters.

This is achieved by allowing JSTL EL expressions as parameter values. The parameter definition in the View configuration specifies the name of the parameter passed to the implementation Component or Task (`paramName`) and EL expression to construct the value from the parameter(s) passed by the caller (`paramValue`). Those EL expressions are evaluated in the context of a special component used to invoke Views, so all *parameters, request and session variables* can be used to create the resulting value.

The following example shows a definition for the "Detail" View for the "Service" entity. Note how the general "entityKey" parameter, which is applicable to all Detail Views, translates to a specific parameter of the particular implementation Task.

Example 29. Classification of data in Java

```
<!--
    We declare a view of type "detail", which is implemented by the
    Task /browse/serviceDetail
-->
<view type="detail" task="/browse/serviceDetail">
    <!--
        The implementation task accepts "serviceKey" parameter,
        we have to adapt the View's parameter to the custom name.
    -->
    <parameter paramName="serviceKey" paramValue="${entityKey}"/>
</view>
```

References

Entities may have some relationships or associations between them. An association between A and B is established by creating a `keyedReference`, with the `tModelKey` that identifies the type of the relationship and a `keyValue` which holds the `entityKey` of the other side of the association. Only directed associations between two UDDI entities are supported, however because of Registry query capabilities, it is also possible to navigate in the reverse direction of an association - and Business Service Console supports that with the "Referenced By" action.

A reference is defined by:

refName

An identifier that identifies this reference.

keyedReference

A `keyedReference` which is used to represent the association in the Registry. The `tModelKey` is mandatory, the `keyName` tag is optional: if present, the `keyedReference` must have such `keyName` value in order to form this reference.

originType

Zero or more `originTypes` can be specified to restrict which Entities can establish associations. If no `originType` is present, the association can originate at any type of entity. When `originType` is present, only the listed entity types can serve as origins for the association. Multiple `originType` values are supported.

The permitted values are the values of the `id` Entity definition attribute. In addition, values that represent the UDDI structure types are permitted (`BE, BS, BT, TM`). When an UDDI structure type is specified, the Reference can originate from any entity derived from that UDDI structure.

Note

The permitted origins should be a *subset* of the relationship Taxonomy compatibility list. If you permit an `originType`, whose UDDI structure is incompatible with the relationship Taxonomy, you will not be able to add such references (associations) to entities.

The Business Service Console presents References to other entities on Detail pages of entities, and provides "Referenced By" action for an entity to discover where the entity is referenced from. References defined in this configuration can also be added by the Business Service Console user using the Add Reference Wizard.

The following example shows how Policies can be associated with an arbitrary Entity. We define a reference to the "policy" entities, with a certain tModelKey (according to the WS-Policy specification), and we do not restrict who can use such a reference.

Example 30. Policy Entity

```
<!--
   Definition of the "Policy" entity
-->
<entity entityId="policy" type="TM" icon="policy.gif"
      captionKey="bsc.entityViews_policies" descriptionKey="bsc.entityViews_policies">
      <!-- Some categorization that identifies the entity -->
      <categorization>
         <keyedReference tModelKey="uddi:schemas.xmlsoap.org:policytypes:2003_03"
             keyValue="policy" keyName="policy"/>
      </categorization>
      <views>
         <!-- List of views, not important for this example -->
         ...
      </views>

      <references>
         <!--
            We define a Reference named "refLocalPolicy", with a (localizable) caption and de-
scription.
            originTypes specifier is missing, so this Reference can originate from any type of

            Entity.
         -->
         <reference refName="refLocalPolicy"
               captionKey="bsc.entityViews_policy_refLocalPolicy_caption"
               descriptionKey="bsc.entityViews_policy_refLocalPolicy_description">
            <!--
               This Reference will be stored using a keyedReference, that have tModelKey set
               to "uddi:schemas.xmlsoap.org:localpolicyreference:2003_03" and keyName set to
               "Associated Policy"
            -->
            <keyedReference tModelKey="uddi:schemas.xmlsoap.org:localpolicyreference:2003_03"

               keyName="Associated Policy"/>
         </reference>
      </references>

</entity>
```

How to classify UDDI data

If a Component wants to smoothly integrate, it should ask the Entity Configuration to classify the data it works with. Then it can write proper nouns to the web page, and use tasks and components configured for the entity instead of using hardcoded links. The first step is obviously to find out what Entity the data correspond to.

In Java, you will use the `EntityHelper` to determine the classification:

Example 31. Classification of data in Java

```
UDDIObject fromInstance;

/**
    Assume, that the "fromInstance" variable is initialized to an UDDIObject
    instance
 */

// Extract CategoryBag from whatever UDDI structure we have
CategoryBag fromCatBag = BscObjectUtilities.getCategoryBag(fromInstance, true);
// Get the list of KeyedReferences
KeyedReferenceArrayList fromKr = fromCatBag.getKeyedReferenceArrayList();
// Lookup the appropriate Entity definition from Entity Configuration
EntityHelper.Entity myEntity = helper.findEntityByCategorization(fromKr, fromType);
```

The code snipped provides you with an `EntityHelper.Entity` instance, which describes the data type. Please refer to API documentation for details how to use the retrieved data.

Using Entities in JSP pages

The `EntityHelper` API class is designed for simple usage from JSPs. For classification, you may use the following snippet:

Example 32. Classification of data in JSP

```
<!--
    The "instance" variable should be initialized to some UDDIObject
    instance. The "entityType" variable will be created and set to the
    appropriate EntityHelper.Entity instance by the tag.
-->
<bsc:setEntityClassification var="entityType" instance="${instance}"/>
```

The `bsc:setEntityClassification` is a JSP alternative to call the `findEntityByClassification` method of the `EntityHelper` class. Note the usage of the `bscEntityClassifier`. This is session variable, provided by the Business Service Console Framework so the `EntityHelper` API is accessible from JSP pages.

If you are given an `entityId` instead of a data structure, you may easily refer to the `EntityHelper.Entity` instance using an EL expression in the JSP:

Example 33. Classification of data in JSP

```
<!--
    The "entityId" variable should be initialized
    to one of the entity types as defined in bsc.xml

    The "bscEntityClassifier" contains a framework-provided instance
    of the EntityHelper API, which provides a Map of available entities
    for easy lookup from JSP.
-->
<c:set var="bscEntityType" value="${bscEntityClassifier.entities[entityId]}"/>
```

In order to use the Entity's caption or description, the procedure described in Localization guide must be used, to make use of the appropriate localized string. We recommend using the following pattern:

Example 34. Classification of data in JSP

```
<!--
    First, get the entity type for the given entityId, we are expected to
    work with
-->
<c:set var="bscEntityType" value="${bscEntityClassifier.entities[entityId]}"/>

<!--
    Handle embedded bundle path specification, see localization guide for
    the details. The evaluated bundle name and key name will be placed
    into named request variables
-->
<syswf:parseResourceKey key="${bscEntityType.captionKey}"
    defaultBaseName="com.systinet.uddi.bui.framework.BSCMessages"
    varBundle="bundleName" varResource="finalCaptionKey"/>

<!--
    Load the bundle, which actually contains the key.
    Note that the bundle name may not be known at design time,
    as it may be embedded in the generalized resource bundle key
-->
<fmt:setBundle basename="${bundleName}" var="dynamic_Message"/>

<!--
    Setup two variables, one holding plural noun for entity caption,
    the other will hold the singular
-->
<fmt:message key="${finalCaptionKey}" var="entityCaption"
             bundle="${dynamic_Message}"/>
<fmt:message key="${finalCaptionKey}_single" var="entityCaption_single"
             bundle="${dynamic_Message}"/>

<!--
    Finally, format some message (properly localizing it through a bundle),
    and substitute the entity nouns in it. Note that the message itself
    can control whether plural or singular is used - it can use {0} to denote
    plural and {1} for singular noun.
-->
<fmt:message key="some_message_key" bundle="${myBundle}">
    <fmt:param value="${entityCaption}"/>
    <fmt:param value="${entityCaption_single}"/>
</fmt:message>
```

The snippet first parses the resource bundle key provided as `entity.captionKey` property, then loads the appropriate `ResourceBundle` using the `fmt:setBundle` standard tag. Note the bundle key naming convention used to load the singular and plural nouns for the Entity type.

Using Views

When working with some data structure, you may directly invoke a Component, using `syswf:component`, or make a link to a specific task using `syswf:control`. If you work on a mixture of data structures, each structure may require a different Component to display itself, or a different Task to perform the action. When the Entity Configuration changes, so that, for example, the task URI of the **Edit** operation changes, pages which use hardcoded component names or task URIs may become inconsistent with the rest of the UI.

You may perform the operation in an abstract way, using the `invokeEntityView` Component. You need to pass in enough information to identify the entity type and you need to specify the type of invoked View (see above for the overview of supported view types). Parameters defined by the View specification will be forwarded to the View component or task. You may pass *additional* parameters, but you have to prefix them with the prefix `view_` so that they are recognized and forwarded.

Example 35. Invoking a Component configured in Entity Configuration

```
<!--
    The following code invokes a "searchResults", which produces a table
    of results for the entity and the passed query.
    The code is taken out from Reports tab implementation
-->
<syswf:component prefix="${tabId}" name="invokeEntityView">
    <!--
        The desired viewType
    -->
    <syswf:param name="viewType" value="searchResults"/>
    <!--
        The query to process, taken from a prepared Map
        of queries for individual entity types
    -->
    <syswf:param name="query" value="${entityQueries[type.id]}"/>
    <!--
        Output parameter, component stores the result list in a temporary
        to allow the caller to find out whether the result list is
        empty
    -->
    <syswf:param name="var" value="references_tmp"/>
    <!--
        Propagates the type of the entity, to cover the case
        the view is reusable and is used for multiple entity
        types
    -->
    <syswf:param name="entityId" value="${type.id}"/>
</syswf:component>
```

Example 36. Linking to a Task configured in Entity Configuration

```
<!--
    This snippet invokes a Create Wizard for the given entity.
-->
<syswf:component name="invokeEntityView" prefix="create">
    <syswf:param name="entityId" value="${entityId}"/>
    <syswf:param name="viewType" value="create"/>

    <!-- A HTML link will be generated -->
    <syswf:param name="mode" value="anchor"/>
    <!-- Text for the hyperlink -->
   <syswf:param name="caption" value="Link text"/>
</syswf:component>
```

You may also need to determine whether a certain View is available. The `EntityHelper.Entity` provides you with all supported views as a `java.util.Map`, so you use the contents from a JSP easily:

Example 37. Linking to a Task configured in Entity Configuration

```
<!-- Set the entity type into a variable, for convenience -->
<c:set var="bscEntityType" value="${bscEntityClassifier.entities[entityId]}"/>

<!-- Check whether the desired view is available -->
<c:if test="${not empty bscEntityType.views['create']}">
    <!-- Do some fancy stuff -->
    ...
</c:if>
```

The presence of a View indicates, that a certain function is available for an entity. You may conditionally change the page appearance based on such an indication.

Linking to a Detail page

In places where an entity is mentioned, it is often appropriate to link to the entity Detail page. There is a special component showEntityName for this purpose. It renders the entity's name as a hyperlink to the entity's Details.

Example 38. Linking to entity details

```
<!--
    This example shows how to create a link to the detail
    page of an Entity. The entity is given by its key,
    UDDI type and the keyedReferences.
-->
<syswf:component name="showEntityName" prefix="name1">
    <syswf:param name="entityKey" value="${key}"/>
    <syswf:param name="uddiType" value="TM"/>
    <syswf:param name="keyedReferences" value="${keyedReferenceArrayList}"/>
</syswf:component>


<!--
    The following example shows how to use UDDI structure itself,
    if it is available to link to the relevant entity detail
-->
<syswf:component name="showEntityName" prefix="n_${row.key}">
    <syswf:param name="entityInstance" value="${theStructure}"/>
    <!--
 We override the rendered string with a custom value.
 If this was omitted, the entity name would be printed
 as the hyperlink text
    -->
    <syswf:param name="instanceName" value="Some string"/>
</syswf:component>
```

A description of the component and its parameters can be found in file `jsp/browse/showEntityName.jsp`, which you can find in `bsc.jar` or in the BSC work directory.

3.8.5. Permission support

Business Service Console contains powerful support for user permission evaluation on selected objects. The developer can easily find out, if the current user is allowed to manipulate some object. This feature takes into consideration object ownership, Access Control Lists, groups and API permissions.

Data classes

The API contains two important Java types. The first is the class `com.systinet.uddi.bui.framework.component.util.permission.UserContext`. An instance is created automatically, when a user logs into the Business Service Console and it is available in the global session under key `userContext`. The instance holds the groups that the user is member of and a list of his permissions.

Then there is an interface `com.systinet.uddi.bui.framework.component.util.permission.DataFeeder`. It is the developer's responsibility to create and feed an instance of its implementation. There are two implementations available. `com.systinet.uddi.bui.framework.component.util.permission.UDDIDataFeeder` is initialized with list of UDDI keys and it fetches specified UDDI structures from BEA AquaLogic Service Registry. If these structures are already available, then it is better to use `com.systinet.uddi.bui.standard.component.util.permission.BuiDataFeeder` for performance reasons.

PermissionEvaluator

To check user permissions in Java code you must use class `com.systinet.uddi.bui.framework.component.util.permission.PermissionEvaluator`. It contains public methods to check whether the user can create a business service in

the given business entity, or binding template in the given business service, and to check whether the user can update or delete a specified business entity, business service, binding template or tModel. These methods take a UDDI key, the `UserContext` and a `DataFeeder` implementation as arguments.

Example 39. ParseResourceKey tag - Usage Example

```
boolean allowed = PermissionEvaluator.checkPermissionDeleteTM(tModelKey, userContext,
dataFeeder);
```

checkPermission tag

To check user permissions in JSP, there is a tag `checkPermission`. In addition to a UDDI key, the `UserContext` and a `DataFeeder` implementation, it accepts `operation` and `var` attributes as arguments. It specified variable receives the result of the check.

Table 95. `checkPermission` tag Parameters

Param	Description	Required
`var`	Name of the variable that will hold the result of the check.	yes
`scope`	Scope for the new variable.	no
`operation`	Operation identifier. One of *create*, *edit* and *delete*.	yes
`key`	The key of the UDDI structure for which we want to check permissions.	yes
`userContext`	Container for user account specific data. Typically available in global session, if the user is logged in.	yes
`dataFeeder`	Data object holding information about UDDI structures on this page.	yes

Example 40. ParseResourceKey tag - Usage Example

```
<bsc:checkPermission var="permission"
    operation="edit" key="${row.key}"
    userContext="${globalSession['userContext']}"
    dataFeeder="${dataFeeder}"/>
<c:if test="${permission}">
    <syswf:control targetTask="/publish/endpoints/editEndpoint"
            caption="Delete" mode="image" src="gfx/icon/i_edit.gif">
        <syswf:param name="bindingKey" value="${row.key}"/>
    </syswf:control>
</c:if>
```

3.8.6. Components and Tags

This section describes selected components and tags of the Business Service Console (BSC) framework and components of the Business Service Console. The BSC Framework is a set of components and tags used by developers to develop Business Service Console components. For complete documentation of these components, see the Java Doc located in the `REGISTRY_HOME/doc/bsc-api` directory.

- Framework Components
- Framework Tags
- Business Service Console Components

Framework Components

This section describes the following component types:

Query
Wizard
Result
Taxonomy
Util

Query

In this section, we will show you how Query components are used in Business Service Console. We explain query components on the page shown in Figure 14 with a page from a wizard for creating a new business service

Figure 14. Query Components

The service name in the drop down list under the **rewrite service** option is produced via the Entity chooser component

The following fields in Figure 14 are produced via Taxonomy filters components:

- The **Usage**, **Release date**, **Version** and **Milestone** fields are produced by inputCategorySetter component.
- The **Certification** field is produced by the selectCategorySetter component

Entity Choosers

The Entity Chooser component is used to select one entity from a list of entities obtained by a query.

There are four types of Entity Choosers, each representing a UDDI data structure:

- `businessChooser` for selecting business entities
- `bindingChooser` for selecting binding templates.
- `serviceChooser` for selecting business services
- `tmodelChooser` for selecting technical models.

All these choosers have the similar functionality.

Table 96. `entityChooser` Parameters

Param	Type	Description	Required	I|O
`filter`	`Find_entity` (Find_business, Find_service, Find_binding or Find_tModel)	Filter used for getting a list of entities. If it is not specified, all entities will be fetched.	optional	in
`resultObject`	Object	The bean where key of the selected entity will be saved.	required	in
`resultProperty`	String	The property of the resultObject bean, into which the key will stored.	required	in
`sort`	String	The sorting mode can be `asc` (for acsending) or `desc` (for descending). Entities are sorted in the list according to their name.	optional	in
`hint`	String	String used as a hint which appears if the pointer is on the component view area.	optional	in
`changeAction`	String	Action sent to the parent component when a selection has changed.	optional	in
`detailTask`	String	Task used for rendering the detail of a selected entity.	optional	in
`entityKeyName`	String	Name of the entity key used in detailTask for getting details on an entity.	optional	in
`emptyMessage`	String	Value displayed if there are no entities to be displayed.	optional	in
`pageSize`	Integer	Maximum number of entities to be displayed in the list; default value is 50.	optional	in
`entitiesTruncatedMessage`	String	Value displayed in the list if a query generates more entities than pageSize allows. Default value is "..."	optional	in
`mandatoryPermission`	String	Permission the user must have on an entity in order for it to be selected. In other words an additional filter criterion. Possible values are `create`, `edit` or `delete`.	optional	in

Example 41. `entityChooser` Example

```
<%-- Import the Systinet framework custom tag library. --%>
<%@ taglib prefix="syswf" uri="http://systinet.com/jsp/syswf" %>
...
<syswf:component prefix="business" name="businessChooser">
    <syswf:param name="resultObject" value="${resultBusiness}"/>
    <syswf:param name="resultProperty" value="key"/>
    <syswf:param name="sort" value="ascending"/>
    <syswf:param name="changeAction" value="business"/>
    <syswf:param name="detailTask" value="/browse/providerDetail"/>
    <syswf:param name="entityKeyName" value="businessKey"/>
</syswf:component>
```

Taxonomy Filters

Note

Taxonomy filters have been obsoleted by categorySetters are are deprecated. See the introduction above.

Taxonomy filter components are used for selecting one category or a subset of all categories of the given taxonomy. The result of the selection is stored in the given CategoryBag.

Taxonomy Filter

The taxonomy filter is used for selecting one or many categories of the given taxonomy specified by its tModel key.

Table 97. `taxonomyFilter` Parameters

Param	Type	Description	Required	I|O
`taxonomyTModelKey`	String	Categories from this taxonomy are rendered as selection options.	required	in
`categoryBag`	CategoryBag	Serves as storage for the result set of the selected categories. The categoryBag component stores and returns the current status of the selection. This parameter can be common for more taxonomies or selectors. For one taxonomy there is an exclusivity of selection, meaning that a new selection replaces of the previous selection for a particular taxonomy.	required	in|out
`selectMode`	String	Defines selection mode as `one` or `many`. If mode `one` is supported, it will be possible to select just one category of the given taxonomy. If mode `many` is supported, it will be possible to select a subset of of the given taxonomy's categories.	optional	in
`viewMode`	String	Defines view modes `radio`, `menu`, or `checkbox`. If `radio` or `checkbox` modes are used, the selection will be rendered as a set of checks or radio buttons (depending on whether selectMode is `one` or `many`) where one button represents one category of the given taxonomy. If `menu` mode is used, a list box is rendered with a select one or multi-select property, depending on the supported selectMode. Each line item of the list box represents one selectable category.	optional	in
`categoryList`	String or Category[] (array) or CategoryArrayList	A list of selectable categories. If the parameter has type String, it must be a list of comma-separated category values. This feature is useful if either the subset of all categories of the given taxonomy is intended to be selectable, or the taxonomy does not have all selectable categories specified. For example, an unchecked taxonomy.	optional	in
`fakeNil`	String	Adds the functionality of the empty selection choice useful for selectionMode = `one`. If the parameter is used and is not empty, its String value is treated as no selection and is added to the list of selectable categories. If no category is selected, this nil value is visually selected. If this nil value is selected, no category of the given taxonomy is really selected. This feature is useful when selectMode = `one` is supported and a `select one or nothing`" feature is actually desired.	optional	in

Example 42. Taxonomy Filter - Usage Example

```
<%-- Import the Systinet framework custom tag library. --%>
<%@ taglib prefix="syswf" uri="http://systinet.com/jsp/syswf" %>
...
<syswf:component prefix="filter" name="taxonomyFilter">
    <syswf:param name="taxonomyTModelKey"
        value="uddi:uddi.org:categorization:types"/>
    <syswf:param name="categoryList" value="yes,no"/>
    <syswf:param name="categoryBag" value="${categoryBag}"/>
    <syswf:param name="viewMode" value="menu"/>
    <syswf:param name="selectMode" value="one"/>
    <syswf:param name="fakeNil" value="nil"/>
</syswf:component>
```

Taxonomy Pure Filter

The Taxonomy Pure Filter component is intended for incrementally adding to a list of selected categories of the given taxonomy. It is useful for taxonomies without defined categories.

It renders input fields representing the specification of the category which will be added to the list.

Table 98. `taxonomyFilterPure` Parameters

Param	Type	Description	Required	I|O
`taxonomyTModelKey`	String	Specifies the taxonomy the categories of which can be set to the resulting categoryBag.	required	in
`categoryBag`	Category-Bag	A storage of the result set of the selected categories. The categoryBag component stores and returns the current state of the selection. This parameter can be common for more taxonomies or selectors. For one taxonomy there is an exclusivity of selection. This means that a new selection of categories replaces the previous selection for a particular taxonomy.	required	in|out
`restricted`	String	If entered and not empty, it renders one input field for the specification of the value specifying category. Neither captions nor a key name input field are presented.	optional	in
`reuse`	String	If the reuse parameter is not present, the category specified by input fields is simply added to the given categoryBag (if it is not there already). If the reuse parameter is present and not empty, categoryBag is used for redefinition of the category it stores for the taxonomy given as the component parameter. This means that if the categoryBag already stores a category of the given taxonomy, this category is used and input fields are prefilled using this category. If a new specification of input fields gives the specification of a new category, this category will replace the old one stored in categoryBag.	optional	in

Example 43. Taxonomy Filter Pure - Usage Example

```
<%-- Import the Systinet framework custom tag library. --%>
<%@ taglib prefix="syswf" uri="http://systinet.com/jsp/syswf" %>
...
<syswf:component prefix="filterPure" name="taxonomyFilterPure">
    <syswf:param name="taxonomyTModelKey" value="uddi:uddi.org:wsdl:types"/>
    <syswf:param name="categoryBag" value="${categoryBag}"/>
    <syswf:param name="reuse" value=""/>
</syswf:component>
```

Wizard

wizardIterator

This component enables a wizard scenario and handles the navigation between the wizard steps. It renders the wizard navigation buttons as **Next**, **Previous**, **Cancel**, and **Finish**. It is also able to render the complete list of step names with an active step name highlighted.

There are two actions this component sends to the root component: Cancel and Finish. The Cancel action is sent as a reaction when the user presses the **Cancel** button. The Finish action is sent as a reaction when the user presses the **Finish** button. Both actions are declared in the Java part of the wizardIterator component of `com.systinet.uddi.bui.framework.component.wizard.WizardIterator` as final static fields: `CANCEL` and `FINISH`.

Table 99. `wizardIterator` Parameters

Param	Description	Required
`componentNames`	This value must hold a comma-separated list of Strings. Each String must refer to the name of a component. Each component then represents a step of the wizard. The order of the items in the list is the order of the wizard steps.	required
`stepNames`	The value of this parameter must hold a comma-separated list of Strings. Its length must be equal to the length of the list passed to the componentNames parameter. Each item of the list represents the title of the step that will be rendered at the top of the resulting wizard step page. The order of the items should correspond to the order of the items in the componentNames parameter.	required
`form`	The value of this parameter must hold an instance of Object. This instance will be passed as a parameter to every component that represents a wizard step . This is the main entity the wizard iterates over.	required
`showStepList`	The value of this parameter must hold a boolean. When set to true, a list of the step names with an active step highlighted will be displayed in the left part of the wizard's window.	optional
`showDisabledButtons`	The value of this parameter must hold a boolean. When set to true, disabled navigation buttons will also be displayed during the iteration (that is,, the Back button in the first step, the Next button in the last step, and the Finish button before the final step). By default only enabled navigation buttons are displayed.	optional

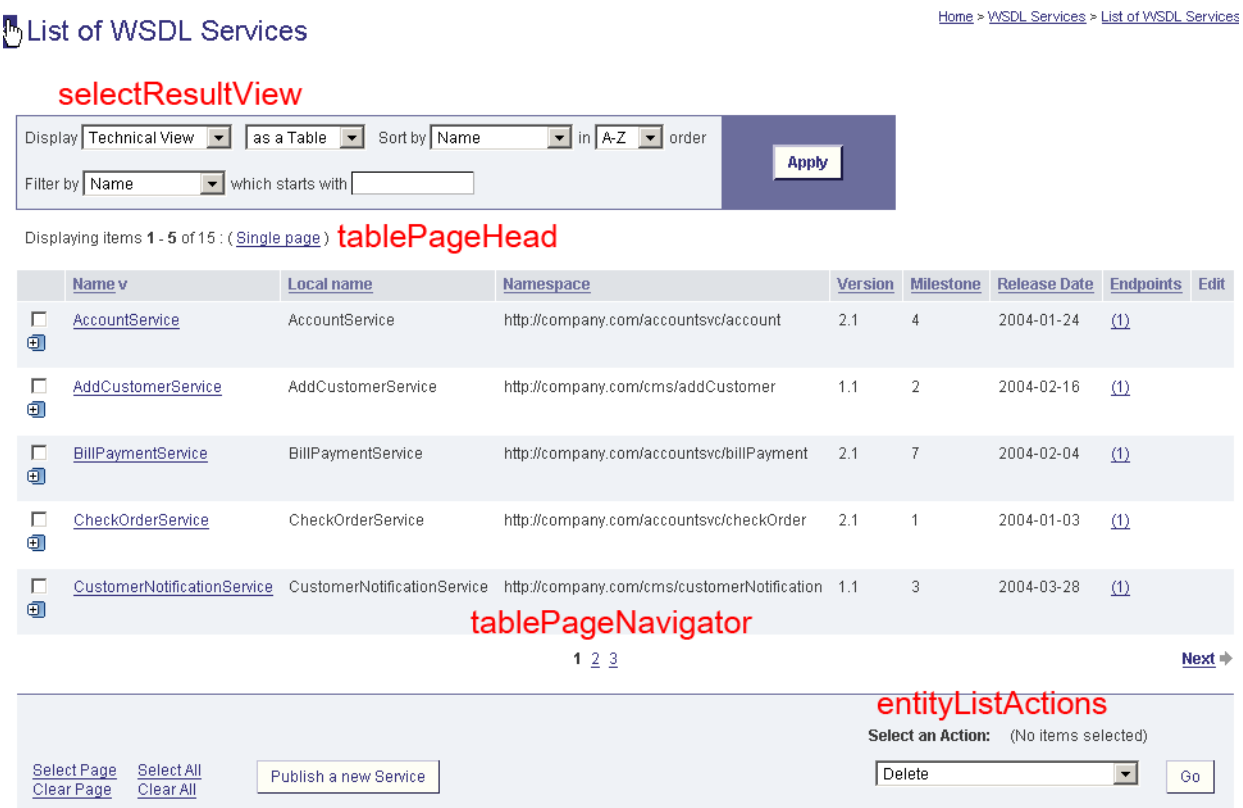
Example 44. wizardIterator - Usage Example

```
<syswf:component name="wizardIterator" prefix="wizard">
    <syswf:param name="componentNames" value="${wizardComponents}"/>
    <syswf:param name="stepNames" value="${wizardNames}"/>
    <syswf:param name="form" value="${form}"/>
    <syswf:param name="showStepList" value="true"/>
    <syswf:param name="showDisabledButtons" value="true"/>
</syswf:component>
```

Result

In this section, we will show how Result components are used in the Business Service Console. We explain result components in Figure 15, which displays list of services

Figure 15. Result Components



The selectResultView is responsible for rendering the **Result View** with a drop down list containing a list of available result views.

The tableFilter component renders fields for filter specification including the **Filter** button.

The columnHeader component renders a single column header in the result table.

selectResultView

This component takes a list of components and displays the first component (or a component specified by the defaultView parameter). SelectResultView also renders a drop down list, so the user can switch to another view from the list.

Table 100. `selectResultView` Parameters

Param	Description	Required
`views`	A comma-separated list of component names.	yes
`titles`	A comma-separated list of titles for the components from the views parameter. They will be displayed in the drop down list.	yes
`defaultView`	The name of a component to be rendered by default.	no

This component accepts other parameters to be passed to the rendered component. In this way, for example, you can pass the sorting column name, set a prefix, or set data to be displayed.

Example 45. selectResultView - Usage Example

```
<syswf:component prefix="providers" name="selectResultView">
    <syswf:param name="views" value="providersCommonResults,providersBizResults" />
    <syswf:param name="titles" value="Common,Business" />
    <syswf:param name="defaultView" value="${defaultView}"/>
    <syswf:param name="prefix" value="providers" />
    <syswf:param name="sortedBy" value="${sortedBy}"/>
    <syswf:param name="resultList"
        value="${availableProviders.businessEntityArrayList}"/>
</syswf:component>
```

tableFilter

This component renders a filter for the table identified by the parameter table.

Table 101. `tableFilter` Parameters

Param	Description	Required
`table`	Value must hold an instance of the `com.systinet.uddi.bui.framework.view.Table` object.	yes
`sortTablePrefix`	The prefix of the component that actually performs the filtering. See the processTable component.	yes

Example 46. TableFilter - Usage Example

```
<syswf:component prefix="filter" name="tableFilter">
    <syswf:param name="table" value="${table}"/>
    <syswf:param name="sortTablePrefix" value="sortTable" />
</syswf:component>
```

processTable

This component filters and sorts the rows of the table passed as a parameter. The component is not visual, it only handles actions and manipulates with data structures.

Table 102. `processTable` Parameters

Param	Description	Required
`table`	Value must hold an instance of the `com.systinet.uddi.bui.framework.view.Table` object.	yes
`defaultSortColumn`	The name of the column to be used for sorting the table provided the user has not chosen a sorting column.	no

Example 47. processTable - Usage Example

```
<syswf:component prefix="sortTable" name="processTable">
    <syswf:param name="table" value="${table}"/>
    <syswf:param name="defaultSortColumn" value="${defaultSortColumn}"/>
</syswf:component>
```

columnHeader

This component renders the column title for a single column. If a column is sortable, then its title is made clickable, so it can be used to sort the table by this column.

Table 103. `columnHeader` Parameters

Param	Description	Required
`table`	Value must hold an instance of the `com.systinet.uddi.bui.framework.view.Table` object.	yes
`column`	Value must hold an instance of the `com.systinet.uddi.bui.framework.view.Column` object.	yes
`sortTablePrefix`	The value of this parameter must be equal to the prefix used in the processTable component, otherwise sorting will not work.	yes

Example 48. columnHeader - Usage Example

```
<syswf:component prefix="providerName" name="columnHeader">
    <syswf:param name="table" value="${table}"/>
    <syswf:param name="column" value="${table.columns[0]}"/>
    <syswf:param name="sortTablePrefix" value="sortTable" />
</syswf:component>
```

Taxonomy

taxonomyTree

This component fetches all referenced categories from a selected internal taxonomy, and constructs a JavaScript tree, and adds it to the selected parent as a child node. Categories are considered to be referenced when they are used in some published keyedReference.

Table 104. `taxonomyTree` Parameters

Param	Description	Required
`tModelKey`	Key of internal taxonomy's tModel. If the taxonomy is not internal, it has no categories and thus no nodes in the tree under the parent will be added.	yes
`keyValue`	Optional parameter that in conjunction with tModelKey constructs one concrete category. Only referenced child categories of this category will be fetched from the registry.	no
`parent`	Name of a JavaScript variable that will serve as parent for the created tree.	yes

Example 49. taxonomyTree - Usage Example

```
<syswf:component prefix="iso3166" name="taxonomyTree">
    <syswf:param name="tModelKey" value="uddi:uddi.org:ubr:categorization:iso3166"/>
    <syswf:param name="targetTask" value="/browse/views/other/enterprise/taxonomy"/>
    <syswf:param name="parent" value="iso3166"/>
</syswf:component>
```

collectCategories

This component allows you to add and remove categories from the list. It consists of the two column areas. The left column contains the taxonomy tree structure; the right contains the list of currently selected values.

Table 105. `collectCategories` Parameters

Param	Description	Required
`taxonomyTModelKey`	Must contain the valid tModel key of the checked internal taxonomy. The values of this taxonomy will be displayed in the tree on the left side.	required
`categoryBag`	The value of this parameter must contain an instance of the org.systinet.uddi.client.v3.struct.CategoryBag class. This class is used as a holder for the selected categories, which are displayed in the right-hand column.	required

Example 50. collectCategories - Usage Example

```
<syswf:component name="collectCategories" prefix="unspsc">
    <syswf:param name="categoryBag" value="${form.business.entity.categoryBag}"/>
    <syswf:param name="taxonomyTModelKey"
        value="uddi:uddi.org:ubr:categorization:unspsc"/>
</syswf:component>
```

selectableTaxonomyTree

This component displays the taxonomy values in the tree-like structure. Each tree node also contains a check box, which can be used to select a specified value. The tree is constructed on a per-node basis, so it can handle potentially large taxonomy structures.

Table 106. `selectableTaxonomyTree` Parameters

Param	Description	Required
`taxonomyTModelKey`	Must contain the valid tModel key of the checked internal taxonomy. The values of this taxonomy will be displayed in the tree.	required

Tip

Usage of the selectableTaxonomyTree component: Note that taxonomy data are usually very large, so it is a good idea to restrict the area occupied by this component using the HTML DIV tag with a specified size. The example bellow displays the tree in a scrollable square area of 300 by 300 pixels.

Example 51. Component Parameters

```
<div style="width:300px; height:300px; overflow:scroll; clip:rect(0px 0px 0px 0px);">
    <syswf:component prefix="taxonomy" name="selectableTaxonomyTree">
       <syswf:param name="taxonomyTModelKey" value="${taxonomyTModelKey}"/>
    </syswf:component>
</div>
```

Util

tabbedFrame (deprecated)

This component renders content divided into several tabs. These tabs are displayed down the right side of screen. The `tabbedFrame` component is substituted for `TabsComponent` class.

Table 107. `tabbedFrame` Parameters

Param	Description	Required
`tabNN_component`	The value of this parameter must refer to the name of the component. The component then represents the content of a tab. The string NN in the parameter name stands for some number used for ordering the tabs. For example, tab1_component will be rendered before tab2_component.	at least one
`tabNN_id`	The value of this parameter must hold the tab's unique identifier. Its value is also used to determine which icons should be rendered as tab handles. For example, a tab with the id `webpaging` will require the images `webroot\gfx\tabs\webpaging_0.gif` and `webroot\gfx\tabs\webpaging_1.gif` to be present in `bsc.jar`. The first image represents unselected tab and the second image contains selected version. The prefix of the parameter name must match the corresponding `tabNN_component` parameter.	at least one
`defaultTab`	The value of this parameter must hold the id of the tab which will be active by default. This value is used only when user displays a page with a tabbed component for the first time. The next time identification of the active tab is obtained from a browser cookie rather than from this parameter.	no

Example 52. tabbedFrame configuration - Component Parameters

```
<config name="web" savingPeriod="5000">
    <webFramework>
    ...
       <component name="configuration_tabs" className="com.systinet.uddi.bui.framework.compon-
ent.util.TabbedFrame" page="util/tabbedFrame.jsp">
            <parameter paramName="tab1_id" paramValue="webtabs"/>
            <parameter paramName="tab1_component" paramValue="configuration_webtabs"/>
            <parameter paramName="tab2_id" paramValue="webviews"/>
            <parameter paramName="tab2_component" paramValue="configuration_webviews"/>
            <parameter paramName="tab3_id" paramValue="webtaxonomy"/>
            <parameter paramName="tab3_component" paramValue="configuration_webtaxonomy"/>
            <parameter paramName="tab4_id" paramValue="webpaging"/>
            <parameter paramName="tab4_component" paramValue="configuration_webpaging"/>
            <parameter paramName="tab5_id" paramValue="webui"/>
            <parameter paramName="tab5_component" paramValue="configuration_webui"/>
        </component>
        <component name="configuration_webtabs" page="configuration/webtabs.jsp"/>
        <component name="configuration_webviews" page="configuration/webviews.jsp"/>
        <component name="configuration_webtaxonomy" page="configuration/webtaxonomy.jsp"/>
        <component name="configuration_webpaging" page="configuration/webpaging.jsp"/>
        <component name="configuration_webui" page="configuration/webui.jsp"/>
        ...
    </webFramework>
...
</config>
```

TabsComponent

TabsComponent is a semi-complete component that is used to display a list of tabs defined in the configuration file `web_component.xml`. It consists of:

- class `com.systinet.uddi.bui.framework.component.util.TabsComponent` that reads the configuration;
- JSP file `util/tabsComponent.jsp` that renders the tabs.

The developer creates a new component with these and the parameter `tabs`.

The `tabs` parameter contains its configuration in the form of XML stored within a `paramValue` element. It must contain root element `tabs`, which contains `tab` elements.

Table 108. `tab` attributes

Attribute	Description	Required
`tabId`	The value of this attribute defines a unique identifier for this tab within this set of tabs. It can be used in `tabs_disable_list` component parameter.	yes
`tabComponent`	This value must be a reference to an existing component that represents the content of the tab.	yes
`captionKey`	The value of this attribute is a reference to a resource bundle with text that will be rendered as the caption of the tab.	yes
`hintKey`	The value of this attribute is a reference to a resource bundle with text that will be rendered as a hint for this tab. It will be displayed when the user points to the tab caption.	no

The component accepts a parameter `tabs_disable_list` with a comma separated list of tab identifiers that will be skipped during tab rendering. Leading and trailing commas are ignored. This way developers may dynamically disable some tabs that are not available in the current context.

Example 53. TabsComponent configuration

```
<component
        name="recentChanges_tabs"
        className="com.systinet.uddi.bui.framework.component.util.TabsComponent"
        page="util/tabsComponent.jsp">
    <parameter paramName="tabs">
        <paramValue>
            <tabs>
                <tab tabId="providers" tabComponent="changes_providers"
                    captionKey="webcomponent.recentChanges_tabs_caption_providers"
                    hintKey="webcomponent.recentChanges_tabs_hint_providers"/>
                <tab tabId="services" tabComponent="changes_services"
                    captionKey="webcomponent.recentChanges_tabs_caption_services"
                    hintKey="webcomponent.recentChanges_tabs_hint_services"/>
                <tab tabId="endpoints" tabComponent="changes_endpoints"
                    captionKey="webcomponent.recentChanges_tabs_caption_endpoints"
                    hintKey="webcomponent.recentChanges_tabs_hint_endpoints"/>
                <tab tabId="interfaces" tabComponent="changes_interfaces"
                    captionKey="webcomponent.recentChanges_tabs_caption_interfaces"
                    hintKey="webcomponent.recentChanges_tabs_hint_interfaces"/>
            </tabs>
        </paramValue>
    </parameter>
</component>
```

Example 54. TabsComponent usage

```
<syswf:component name="recentChanges_tabs" prefix="tabs">
    <syswf:param name="tabs_disable_list" value="endpoints,interfaces"/>
</syswf:component>
```

TreeComponent

TreeComponent is used to display a static tree. Elements of the tree are links to actions and tasks. Sub-trees can be expanded and collapsed. An icon can be shown next to each link.

TreeComponent is a semi-complete component. It consists of

- class com.systinet.uddi.bui.framework.component.util.TreeComponent that reads the configuration from `web_component.xml`;
- JSP `util/treeComponent.jsp` that renders the tree;

All elements of the tree are described in the component configuration, so different trees require different components. The component configuration includes:

- configuration common to different trees:
  - Java class file;
  - JSP file;
- configuration specific to the particular tree:
  - a parameter containing the tree layout;

Example 55. A tree configuration

```
<component name="publishTree"
        className="com.systinet.uddi.bui.framework.component.util.TreeComponent"
        page="util/treeComponent.jsp">
    <parameter paramName="treeContent">
        <paramValue>
            <node nodeId="publish" captionKey="publish" icon="publish.gif">
                <control action="" targetTask="/publish" targetUrl=""/>
                <node nodeId="providers" captionKey="publish_providers" icon="provider.gif">
                    <control targetTask="/catalog/providerMenu"/>
                    <contextMenuReference component="contextMenu_providersList"/>
                </node>
                <node nodeId="services" captionKey="publish_services" icon="service.gif">
                    <control targetTask="/catalog/serviceMenu"/>
                    <contextMenuReference component="contextMenu_servicesList"/>
                </node>
                <node nodeId="resources" captionKey="publish_resources" icon="resources_0.gif"
 openIcon="resources_1.gif">
                    <control targetTask="/publish/resources"/>
                    <node nodeId="wsdl" captionKey="publish_resources_wsdl" icon="wsdl_0.gif">

                        <control targetTask="/catalog/wsdlMenu"/>
                        <contextMenuReference component="contextMenu_wsdlList"/>
                        <node nodeId="portTypes" captionKey="publish_resources_wsdl_portTypes"
 icon="porttype.gif">
                            <control targetTask="/catalog/wsdl/portTypeMenu"/>
                            <contextMenuReference component="contextMenu_portTypeList"/>
                        </node>
                        <node nodeId="ports" captionKey="publish_resources_wsdl_ports"
icon="port.gif">
                            <control targetTask="/catalog/wsdl/portMenu"/>
                            <contextMenuReference component="contextMenu_portList"/>
                        </node>
                    </node>
                    <node nodeId="xsd" captionKey="publish_resources_xsd" icon="xsd.gif">
                        <control targetTask="/catalog/xsdMenu"/>
                        <contextMenuReference component="contextMenu_xsdList"/>
                    </node>
                </node>
            </node>
        </paramValue>
    </parameter>
 </component>
```

Table 109. node attributes

Attribute	Description	Required
nodeId	An identifier for the node, must be unique in the tree.	yes
captionKey	Resource-bundle key for caption of the link.	yes
icon	A relative path to the icon for the node. Default is file.png.	no
openIcon	Like icon attribute but for open nodes. Default is same as icon.	no

Table 110. `node` sub-elements

Element	Description	More than once	Required
node	Node element of subtree.	yes	no
control	Specifies syswf:control like links.	no	yes
elementMenuReference	Link to Section ContextMenuComponent.	no	no

Table 111. `control` attributes

Attribute	Description	Required
action	An action to syswf:control	no
targetTask	A targetTask to syswf:control	no
targetDepth	A targetDepth to syswf:control, with default 0.	no
targetUrl	A targetUrl to syswf:control	no

A control element can contain parameters described in a parameter element. These parameters will be available in the called task/component.

Table 112. `parameter` attributes

Attribute	Description	Required
paramName	An identifier	yes
paramValue	Any string.	yes

A contextMenuReference element links the tree node to a Section ContextMenuComponent, which is activated by right-clicking the node. It contains a component attribute, which is used to:

- create an identifier for linking the node and the actual menu;
- call a component of that name to render a contextMenu component. The rendered component is hidden until the menu is activated;

Up to 9 parameters can be specified in the contextMenuReference via parameter elements. They will be merged with the parameters specified in the configuration of the called contextMenu component and used within the syswf:control element.

A TreeComponent may be called without parameters.

ContextMenuComponent

ContextMenuComponent is used to display a context menu. Elements of the menu are links to actions and tasks. An icon can be shown next to each link.

ContextMenuComponent is a semi-complete component. It consists of

- class com.systinet.uddi.bui.framework.component.util.ContextMenuComponent that reads the configuration from component parameters in `web_component.xml`;
- `util/contextMenuComponent.jsp` that renders the menu.

All elements of the menu are described in the component the configuration, so different context menus require different components. The component configuration includes:

- configuration common to different context menus:
  - Java class file;
  - JSP file;
- configuration specific to the particular context menu:
  - a parameter containing menu items;

Example 56. A context menu configuration

```
<component name="contextMenu_providersList" className="com.systinet.uddi.bui.framework.compon-
ent.util.ContextMenuComponent" page="util/contextMenuComponent.jsp">
    <parameter paramName="menu">
        <paramValue>
            <contextMenu captionKey="providersList_caption">
                <menuItem captionKey="providersList_item_publish_providers"
icon="ctx_bsn_add.gif">
                    <control targetTask="/publish/providers">
                        <parameter paramName="editableMode" paramValue="true"/>
                    </control>
                </menuItem>
                <menuItem captionKey="providersList_item_search_providers"
icon="ctx_bsn_add.gif">
                    <control targetTask="/search/providers">
                        <parameter paramName="editableMode" paramValue="true"/>
                    </control>
                </menuItem>
                <menuItem captionKey="providersList_item_publish_myProviders"
icon="ctx_bsn_add.gif">
                    <control targetTask="/publish/myProviders">
                        <parameter paramName="editableMode" paramValue="true"/>
                    </control>
                </menuItem>
                <menuItem captionKey="providersList_item_publish_providers_createProvider"
icon="ctx_bsn_add.gif">
                    <control targetTask="/publish/providers/createProvider"/>
                </menuItem>
            </contextMenu>
        </paramValue>
    </parameter>
</component>
```

A `contextMenu` element contains attribute `captionKey` specifying the resource bundle key for the caption of the menu. Its content is a list of `menuItem` elements that describe each menu item. Menu items have inks to actions and tasks.a `captionKey` attribute and an `icon` attribute specifying the relative path to icon file. Menu items also contain a `control` element, which is exactly same as the one described in Section TreeComponent.

The component can be called without any parameters, but is not usually called directly in code, but from components such as TreeComponent that reference a context menu from their configuration.

Framework Tags

This section describes the Business Service Console web framework tag library.

- bsc:setLocalizedNames - selects names from a list in specified language
- bsc:setLocalizedDescriptions - selects descriptions from a list in a specified language
- bsc:setSelectedContacts - selects contacts of a certain useType from the given list
- bsc:setCategories - selects KeyedReferences from a specified list
- bsc:parseUddiQuery - sets UDDI query to a specified variable
- Table related tags: bsc:table, bsc:column, bsc:tableActions, bsc:row, bsc:cell , bsc:attribute

bsc:setLocalizedNames

This tag is used to set localized Names from a given list of names. The output JSP variable will contain names that match given criteria. The algorithm selects all Names with a langCode equal to the attribute langCode, if it is defined. Otherwise, Names with default (empty) langCodes are chosen. If there is no such Name at all, then the first Name is selected from the list.

Table 113. `setLocalizedNames` Parameters

Param	Description	Required
`var`	Output variable holding a list of names in the required language.	yes
`value`	This parameter must hold an instance of the `org.systinet.uddi.client.v3.struct.NameArrayList` object. This object will be searched for localized Names.	yes
`scope`	Identifies the scope in which the variable will be set. It accepts the following values: `request`, `session`, and `application`. If it is not defined or has a different value, then the `page` scope is used.	no
`langCode`	Code of the preferred language. Names with this langCode will be selected from the value parameter.	no

Example 57. setLocalizedNames - Usage Example

```
<bsc:setLocalizedNames var="DEFAULT_NAMES" value="${row.nameArrayList}" />
<c:out value="${DEFAULT_NAMES[0].value}"/>
```

bsc:setLocalizedDescriptions

This tag is used to set localized Descriptions from a given list. It creates a new JSP variable holding a DescriptionArrayList of Descriptions that match the given criteria. The algorithm selects all Descriptions with langCode equal to the attribute langCode, if it is defined. Otherwise, Descriptions with default (empty) langCodes are chosen. If there is no such Description at all, then the first Description is selected from the list.

Table 114. `setLocalizedDescriptions` Parameters

Param	Description	Required
`var`	The name of the output variable holding the list of descriptions in the required language	yes
`value`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.DescriptionArrayList` object. This object is searched for localized Descriptions.	yes
`scope`	This parameter identifies the scope in which the variable are to be set. It accepts the following values: `request`, `session`, and `application`. If it is not defined or has a different value, then the `page` scope is used.	no
`langCode`	Code of the required language. Names with this langCode will be selected from the value parameter.	no

Example 58. setLocalizedDescriptions - Usage Example

```
<bsc:setLocalizedDescriptions var="descriptions" value="${row.descriptionArrayList}"
    langCode="${userDefaultLanguage}"/>
<c:out value="${descriptions[0].value}"/>
```

bsc:setSelectedContacts

This tag is used to set Contacts of a certain useType from the given list. It creates a new JSP variable holding ContactArrayList of Contacts that matches given criteria.

The optional findQualifer parameter determines whether an exact match of the useType is required or if a useType containing a regular expression is to be used. Regular expressions used must conform to UDDI syntax, that is, it accepts ? and % as wildcards.

Table 115. `setSelectedContacts` Parameters

Param	Description	Required
`var`	The name of the output variable.	yes
`value`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.ContactArrayList` object; this object will be searched for the selected Contacts.	yes
`scope`	This parameter identifies the scope, where the variable shall be set. It accepts the following values: `request`, `session` and `application`. If it is not defined or has a different value, then the `page` scope is used.	no
`useType`	This parameter holds the value of useType that will be searched in the list of Contacts. If the approximateMatch findQualifier is used, then ? and % characters have the special meaning of wild card characters, as described in the UDDI specification.	yes
`findQualifier`	The findQualifier parameter determines, whether the useType shall be used for exact match (exactMatch) or whether it contains wild card characters (approximateMatch). If it is not specified, exactMatch is used.	no

Example 59. setSelectedContacts - Usage Example

```
<bsc:setSelectedContacts var="contact" value="${row.contactArrayList}"
    useType="%" findQualifier="approximateMatch" />
<c:out value="${contact[0].personNameArrayList[0].value}"/>
```

bsc:setCategories

This tag is used to set KeyedReferences from a given list. It creates a new JSP variable holding a KeyedReferenceArrayList of KeyedReferences that match given criteria. The tModelKey parameter specifies a tModelKey in which each KeyedReference must be selected. The optional keyValue parameter acts as a secondary filter for KeyedReferences.

Table 116. `setCategories` Parameters

Param	Description	Required
`var`	The name of the output variable	yes
`value`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.KeyedReferenceArrayList` object. This object will be searched for matching KeyedReferences. If the value is not defined, the variable will be unset.	yes
`scope`	This parameter identifies the scope in which the variable shall be set. It accepts the following values: `request`, `session`, and `application`. If it is not defined or has a different value, then the `page` scope is used.	no
`tModelKey`	Holds the value of the tModelKey in which each selected KeyedReference must be contained.	yes
`keyValue`	This optional parameter serves as an additional filter. If it is specified then each KeyedReference must contain it.	no
`keyName`	If specified, only keyedReferencies whose keyName equals to the attribute value are copied to the result variable.	no

Example 60. setCategories - Usage Example

```
<bsc:setCategories var="unspc" value="${row.categoryBag.keyedReferenceArrayList}"
    tModelKey="uddi:uddi.org:ubr:categorization:unspsc" />
<c:out value="${unspc[0].keyName}"/>
```

bsc:parseUddiQuery

This tag is used to set a UDDI query to a specified variable. It can parse XML containing one of the following operations:

find_binding
find_business
find_service
find_tModel
get_binding
get_business
get_service
get_tModel
If the value parameter is not set, then the tag body is evaluated and used as the value.

Table 117. `parseUddiQuery` Parameters

Param	Description	Required
`var`	The name of the introduced variable.	yes
`value`	Must hold a valid XML representation of the following UDDI operations: find_binding, find_business, find_service, find_tModel, get_binding, get_business, get_service and get_tModel. Note that namespaces must not be omitted!	no

Example 61. parseUddiQuery - Usage Example

```
<bsc:parseUddiQuery var="findQuery" scope="request">
    <find_tModel xmlns="urn:uddi-org:api_v3">
        <categoryBag>
            <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
                keyValue="portType" />
        </categoryBag>
    </find_tModel>
</bsc:parseUddiQuery>
```

bsc:table

The table tag is a key component in the Business Service Console. It is used to define the Table object, which holds data for query results, columns, and unfolded rows. Many components depends on this object, such as tableFilter and columnHeader.

The behavior of this tag depends on whether the Table object can be found in the selected scope. If the Table is missing, a new object is created and the tag body is evaluated. If the object is present in this scope, then the body is skipped. The table also refreshes, if the `request` contains the `tableRefresh` attribute. This prevents unnecessary data conversions when just redisplaying the same page

For example: when a different view is selected in the selectResultView component, then the tableRefresh attribute is pushed to the request and Table is recreated.

Table 118. `table` Parameters

Param	Description	Required
`var`	The name of the variable holding the Table object.	yes
`scope`	Identifies the scope in which the variable shall be set. It accepts the following values: `request`, `session`, and `application`. If it is not defined or has a different value, then the `page` scope is used.	no

Example 62. Table tag - Usage Example

```
<bsc:table var="${tableName}" scope="session">
    <bsc:column caption="Provider name" filterCaption="Provider name"
    name="providerName" sortable="true" filterable="true"/>
    <bsc:column caption="Keyword Name" filterCaption="Keyword Name"
    name="keywordName" sortable="true" filterable="true"/>
    <bsc:column caption="Keyword Value" filterCaption="Keyword Value"
    name="keywordValue" sortable="true" filterable="true"/>
    <bsc:column caption="Services" name="serviceCount" sortable="true"
        filterable="false"/>
    <c:forEach items="${resultList}" var="row" varStatus="status">
        <bsc:row>
            <bsc:attribute key="businessKey" value="${row.businessKey}" />
            <bsc:attribute key="services" value="${row.businessServiceArrayList}" />
            <bsc:setCategories var="keywords"
                value="${row.categoryBag.keyedReferenceArrayList}"
                tModelKey="uddi:uddi.org:categorization:general_keywords" />
            <bsc:cell trimWhitespace="yes">
                <bsc:setLocalizedNames var="DEFAULT_NAMES"
                    value="${row.nameArrayList}" />
                <c:out value="${DEFAULT_NAMES[0].value}"/>
            </bsc:cell>
            <bsc:cell trimWhitespace="yes">
                <c:out value="${keywords[0].keyName}"/>
            </bsc:cell>
            <bsc:cell trimWhitespace="yes">
                <c:out value="${keywords[0].keyValue}"/>
            </bsc:cell>
            <bsc:cell trimWhitespace="yes">
                <syswf:size var="SERVICE_COUNT"
                    value="${row.businessServiceArrayList}"/>
                <c:out value="${SERVICE_COUNT}"/>
            </bsc:cell>
        </bsc:row>
    </c:forEach>
</bsc:table>
```

bsc:tableActions

The `bsc:tableActions` tag initializes its output variable with a structure that describes actions that are rendered on the page by the browser. The tag does not produce any HTML output, just the data. It is provided for convenient and simple initialization of other UI components.

The list of actions is populated by `bsc:action` tags, nested in the `bsc:tableActions` tag. The standard UI uses attributes of `bsc:action` to populate and initialize HTML page controls. The standard UI behavior will be explained in attribute descriptions.

To support extensibility, `bsc:action` can specify an `insertInto` attribute, that directs the action into a specified action list. This feature can be utilized by extensions that build on a basic UI.

Table 119. bsc:tableActions attributes

Attribute	Description	Required
var	The name of the output variable that will receive the list of actions constructed by the tag.	yes
scope	The scope of the variable. See the JSP specification [http://jcp.org/aboutJava/communityprocess/final/jsr152/] for a list of scope names.	no

Table 120. bsc:action attributes

Attribute	Description	Required
task	URI of the task associated with the action.	yes
action	The action string associated with the UI action. This string can be sent to the associated task.	no
default	A boolean value of `true` or `false`.	yes
insertInto	The value must be of the type `com.systinet.uddi.bui.framework.view.TableActions`. The action produced by the tag will be appended to that list of actions.	

bsc:column

The column tag appends a new Column to the list of Columns in Table. This tag must be nested within the Table tag.

Table 121. `column` Parameters

Param	Description	Required
`caption`	The caption for this column. Used by the columnHeader component.	yes
`filterCaption`	The caption for this column in the tableFilter component. If empty, no caption is used.	no
`name`	The identifier of this column for sorting and filtering. If the column is used for filtering or sorting rows, then this parameter is mandatory.	only if used for filtering or sorting
`sortable`	The boolean property which determines, whether this column can be used for sorting rows of the Table. A case-insensitive match to yes and no is performed.	no
`filterable`	The boolean property which determines, whether this column can be used for filtering rows of the Table. The values are `yes` or `no`.	no

bsc:row

This tag appends a new Row to the list of Rows in Table. It must be nested within the Table tag. This tag supports storing of attributes via a directly nested attribute tag. The `key` parameter is the unique identifier of the row. The identifier must implement `java.io.Serializable`. If no key value is given, a key value is generated when the row is inserted into a table. The row can contain <bsc:attribute> tags which populate the row attributes property.

bsc:cell

The cell tag appends a new Cell to the current Row in the Table. This tag must be nested within the Row tag. If the caption parameter is not specified, then the tag body is evaluated and body content is used. The trimWhiteSpace attribute determines whether white spaces at both ends of the body content string shall be removed. This tag supports the storing of attributes via directly nested attribute tags.

Table 122. `cell` Parameters

Param	Description	Required
`caption`	The caption for this cell.	no
`trimWhiteSpace`	The boolean property which determines, whether white space characters from body content shall be removed. Used only if caption is not defined. A case-insensitive match to yes and no is performed.	no

bsc:attribute

The attribute tag is used to decorate the parent tag with additional data. The parent tag hierarchy is searched for the first tag that implements the Attributive interface. This parent will receive this tag value via the method `void addAttribute(string key, String value)`.

If the value parameter is not set, then the tag body is evaluated and used as the value.

Table 123. `attribute` Parameters

Param	Description	Required
`key`	Name of the attribute	yes
`value`	The string value for the specified key.	no

Business Service Console Components

This section describes selected components of the Business Service Console.

- providerSearchResults - executes a find_business UDDI query and displays providers that match the query.
- executeFindProviders - Executes a find_business UDDI query and produces a list of providers that match the query.
- serviceSearchResults - Executes a find_service UDDI query and displays services that match the query.
- executeFindServices - Executes a find_service UDDI query and displays services that match the query.
- endpointSearchResults - Executes a find_binding UDDI query and displays Endpoints that match the query.
- executeFindEndpoints - Executes a find_binding UDDI query that processes the results that match the query.
- interfaceSearchResults - Executes a find_tModel UDDI query and displays Interfaces that match the query.
- executeFindInterfaces - Executes a find_tModel UDDI query and displays Interfaces that match the query.
- bindingSearchResults - Executes a find_tModel UDDI query and displays Bindings that match the query.
- executeFindBinding - Executes a find_tModel UDDI query in order to find Bindings.
- getOperations - Fetches a list of operations for a Binding or a PortType from a WSDL.
- getDocumentation - Extracts the useType documentation from an org.systinet.uddi.client.v3.struct.OverviewDocArrayList.
- getServiceEndpoints - Analyzes the Binding Templates of a Business Service, and creates a list of valid WSDL Endpoints mapped to those Binding Templates.
- selectCategory - Takes an existing query and adds a KeyedReference into the query's CategoryBag.

providerSearchResults

This component executes a find_business UDDI query and displays providers that match the query. Alternatively, the component may be given a list of Businesses to display. This alternative approach is recommended when the result cannot be produced by an UDDI query directly, such as when there is some post-processing involved.

The results are displayed in a standard layout that allows selection of a view from a list of supported views.

Table 124. `providerSearchResults` Parameters

Param	Description	Required
`providerList`	Must hold an instance of one of the following: import org.systinet.uddi.client.v3.struct.BusinessList org.systinet.uddi.client.v3.struct.BusinessDetail org.systinet.uddi.client.v3.struct.BusinessEntityArrayList The query will not be executed, rather Providers (Businesses) which are given in the list will be displayed.	yes, exclusive with query
`defaultView`	The component name of the view component that should be displayed by default. Valid names are: • providerCommonResults - common results view • providerBusinessResults - business results view If the parameter is not present, the providersCommonResults view will be displayed.	no
`sortedBy`	The name of the column for the initial sort order. The list of applicable values depends on the selected default view Component.	no
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_business` object. The query will be executed in UDDI using the logged user's credentials.	yes, exclusive with providerList
`var`	A variable that will also receive the results. An anticipated use of this parameter is to enable the caller to detect if there are no results.	no

Example 63. providerSearchResults - Usage Example

This example shows how to display all Providers (UDDI business entities) whose name starts with "A".

```
<bsc:parseUddiQuery var="providersQuery" scope="session">
    <find_business xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
            <findQualifier>
                uddi:uddi.org:findqualifier:approximatematch
            </findQualifier>
        </findQualifiers>
        <name>A%</name>
    </find_business>
</bsc:parseUddiQuery>

<syswf:component prefix="providers" name="findProviders">
    <syswf:param name="query" value="${providersQuery}"/>
</syswf:component>
```

executeFindProviders

This Component executes a find_business UDDI query and produces a list of providers that match the query. The results will be placed into a specified result variable in the local session.

The specified result variable is a cache for the result value. If the variable is not empty, the query is not executed. You must clear the variable in order to get a fresh result set. The cache is automatically cleared when a task is selected.

Table 125. `executeFindProviders` Parameters

Param	Description	Required
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_business` object. The query will be executed in UDDI using the logged user's credentials.	yes
`var`	This value will be used as the name of the result variable in the local session where the component will store the result of the query. The stored result will be of type `org.systinet.uddi.client.v3.struct.BusinessDetail`.	yes

Example 64. executeFindProviders - Usage Example

This example shows how to display the names of all Providers (UDDI business entities) whose name starts with "A".

```
<bsc:parseUddiQuery var="providersQuery" scope="session">
    <find_business xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
            <findQualifier>
                uddi:uddi.org:findqualifier:approximatematch
            </findQualifier>
        </findQualifiers>
        <name>A%</name>
    </find_business>
</bsc:parseUddiQuery>

<syswf:component prefix="providers" name="executeFindProviders">
    <syswf:param name="query" value="${providersQuery}"/>
    <syswf:param name="var" value="searchResults"/>
</syswf:component>

Providers whose names start with "A":
<c:forEach items="${searchResults}" var="provider">
    <c:out value="${provider.names[0].value}"/><br/>
</c:forEach>
```

serviceSearchResults

This component executes a find_service UDDI query and displays services that match the query. Alternatively, the component may be given a list of Services to display. This alternative approach is recommended when the result can not be produced by a UDDI query directly, for example, if there is some post-processing required.

The results are displayed in a standard layout that allows the user to select a view from among the supported ones (common, business, etc.).

Table 126. `serviceSearchResults` Parameters

Param	Description	Required
`serviceList`	Must hold an instance of one of the following: import org.systinet.uddi.client.v3.struct.ServiceList org.systinet.uddi.client.v3.struct.ServiceDetail org.systinet.uddi.client.v3.struct.BusinessServiceArrayList The query will not be executed, rather Services which are given in the list will be displayed.	yes, exclusive with query
`defaultView`	The component name of the view component that should be displayed by default. Valid names are: • serviceCommonResults - common results view • serviceBusinessResults - business results view • serviceTechnicalResults - technical results view If the parameter is not present, the serviceCommonResults view will be displayed.	no
`sortedBy`	The name of the column for the initial sort order. The list of applicable values depends on the selected default view Component.	no
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_service` object. The query will be executed in UDDI using the logged user's credentials.	yes, exclusive with serviceList

Example 65. serviceSearchResults - Usage Example

This example displays all services that are categorized within the Certification taxonomy.

```
<bsc:parseUddiQuery var="serviceQuery" scope="session">
    <find_service xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
            <findQualifier>
                uddi:uddi.org:findqualifier:approximatematch
            </findQualifier>
        </findQualifiers>
        <categoryBag>
            <keyedReference
                tModelKey="uddi:systinet.com:taxonomy:service:certification"
                keyValue="%"
            />
        </categoryBag>
    </find_service>
</bsc:parseUddiQuery>

<syswf:component prefix="services" name="findServices">
    <syswf:param name="query" value="${serviceQuery}"/>
</syswf:component>
```

executeFindServices

This component executes a find_service UDDI query and displays services that match the query. The results of the query will be post-processed using `org.systinet.uddi.client.v3.struct.ServiceDetail` so that information about the owning Business Entity is easily available. The component will create wrapper structures (of type `com.systinet.uddi.bui.standard.util.Service`) to link the Service to its parent Business Entity instance. The wrappers will be placed into a specified result variable in the local session.

The specified result variable is a cache for the result value. If the variable is not empty, the query is not executed. You must clear the variable in order to get a fresh result set. The cache is automatically cleared when a task is selected.

Table 127. `executeFindServices` Parameters

Param	Description	Required
`query`	Value must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_service` object. The query will be executed in UDDI using the logged user's credentials.	yes
`var`	Used to name the result variable in the local session, where the component will store the result of the query. The stored result will be of type `com.systinet.uddi.bui.standard.util.Service[]`.	yes

endpointSearchResults

This component executes a find_binding UDDI query and displays Endpoints that match the query. Alternatively, the component may be given a list of Endpoints to display. This alternative approach is recommended when the result can not be produced by an UDDI query directly, for example, if there is some post-processing required.

The component will load more information from the UDDI Registry, for the interfaces and bindings available on the matching Endpoints. These queries will be executed in UDDI using the logged user's credentials.

The results are displayed in a standard layout that allows the user to select a view from among the supported ones (common, business, etc.).

Table 128. `endpointSearchResults` Parameters

Param	Description	Required
`endpointList`	Must hold an instance of either of the following: org.systinet.uddi.client.v3.struct.BindingDetail org.systinet.uddi.client.v3.struct.BindingTemplateArrayList The query will not be executed, rather Endpoints which are given in the list will be displayed.	yes, exclusive with query
`defaultView`	The component name of the view component that should be displayed by default. The valid names are: • endpointsCommonResults - common results view • endpointsOperationResults - operations results view • endpointsTechnicalResults - technical results view If the parameter is not present, the serviceCommonResults view will be displayed.	no
`sortedBy`	The name of the column used for the initial sort order. The list of applicable values depends on the selected default view component.	no
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_binding` object. The query will be executed in UDDI using the logged user's credentials.	yes, exclusive with endpointList

Example 66. endpointSearchResults - Usage Example

This example displays all endpoints that are categorized within the Certification taxonomy.

```
<bsc:parseUddiQuery var="endpointQuery" scope="request">
    <find_binding xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
            <findQualifier>
                uddi:uddi.org:findqualifier:approximatematch
            </findQualifier>
        </findQualifiers>
        <categoryBag>
            <keyedReference tModelKey="uddi:65719168-72c6-3f29-8c20-62defb0961c0"
             keyValue="%"/>
            <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
                keyValue="portType"/>
        </categoryBag>
    </find_binding>
</bsc:parseUddiQuery>

<syswf:component prefix="endpoints" name="endpointSearchResults">
    <syswf:param name="query" value="${endpointQuery}"/>
</syswf:component>
```

executeFindEndpoints

This Component executes a find_binding UDDI query that processes the results that match the query. The Component will load more information from the UDDI Registry, for the interfaces and bindings available on the matching Endpoints. These queries will be executed using the logged user's credentials. The processed information will be available as an array of wrapper structures of type `com.systinet.uddi.bui.standard.util.Endpoint`.

The specified result variable is a cache for the result value. If the variable is not empty, the query is not executed. You must clear the variable in order to get a fresh result set. The cache is automatically cleared when a task is selected.

Table 129. `executeFindEndpoints` Parameters

Param	Description	Required
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_binding` object. The query will be executed in UDDI using the logged user's credentials.	yes
`var`	Used to name the result variable in the local session in which the component will store the result of the query. The stored result will be of type `com.systinet.uddi.bui.standard.util.Endpoint[]`.	yes

Example 67. executeFindEndpoints - Usage Example

This example displays all endpoints that are categorized within the Certification taxonomy.

```
<bsc:parseUddiQuery var="endpointQuery" scope="request">
    <find_binding xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
            <findQualifier>
                uddi:uddi.org:findqualifier:approximatematch
            </findQualifier>
        </findQualifiers>
        <categoryBag>
            <keyedReference tModelKey="uddi:65719168-72c6-3f29-8c20-62defb0961c0"
             keyValue="%"/>
            <keyedReference tModelKey="uddi:uddi.org:wsdl:types"
                keyValue="portType"/>
        </categoryBag>
    </find_binding>
</bsc:parseUddiQuery>

<syswf:component prefix="endpoints" name="executeFindEndpoints">
    <syswf:param name="query" value="${endpointQuery}"/>
</syswf:component>
```

interfaceSearchResults

This component executes a find_tModel UDDI query and displays Interfaces that match the query. Alternatively, the component may be given a list of Endpoints to display. This alternative approach is recommended when the result cannot be produced by an UDDI query directly, such as when some post-processing required.

The results are displayed in a standard layout that allows the user to select a view from among the supported ones (common, business, etc.).

Table 130. `interfaceSearchResults` Parameters

Param	Description	Required
`interfaceList`	Must hold an instance of either of the following: org.systinet.uddi.client.v3.struct.TModelList org.systinet.uddi.client.v3.struct.TModelDetail org.systinet.uddi.client.v3.struct.TModelArrayList The query will not be executed, rather Interfaces which are given in the list will be displayed.	yes, exclusive with query
`defaultView`	The component name of the view component that should be displayed by default. The valid names are: • resourcesPortTypeResults - common results view • portTypeTechnicalResults - technical results view If the parameter is not present, the resourcesPortTypeResults view will be displayed.	no
`sortedBy`	The name of the column for the initial sort order. The list of applicable values depends on the selected default view component.	no
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_tModel` object. The query will be executed in UDDI using the logged user's credentials.	yes, exclusive with endpointList

interfaceSearchResults component in action

Example 68. Component Parameters

This example displays all interfaces that are categorized as "Stable" in the interface:status taxonomy.

```
<bsc:parseUddiQuery var="interfaceQuery" scope="request">
    <find_tModel xmlns="urn:uddi-org:api_v3">
        <categoryBag>
            <keyedReference
                tModelKey="uddi:systinet.com:taxonomy:interface:status"
                keyValue="Stable"
            />
        </categoryBag>
    </find_tModel>
</bsc:parseUddiQuery>

<syswf:component prefix="interfaces" name="interfaceSearchResults">
    <syswf:param name="query" value="${interfaceQuery}"/>
</syswf:component>
```

executeFindInterfaces

This component executes a find_tModel UDDI query and provides a list of TModels (Interfaces) that match the query. The result of the query is placed into a result variable into the local session.

The specified result variable is a cache for the result value. If the variable is not empty, the query is not executed. You must clear the variable in order to get a fresh result set. The cache is automatically cleared when a task is selected.

Table 131. `executeFindInterfaces` Parameters

Param	Description	Required
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_tModel` object. The query will be executed in UDDI using the logged user's credentials.	yes
`var`	Used to name the result variable in the local session in which the component will store the result of the query. The stored result will be of type `org.systinet.uddi.client.v3.struct.TModelDetail`.	yes

executeFindInterfaces component in action

Example 69. Component Parameters

This example displays tModel names of all interfaces that are categorized as "Stable" in the interface:status taxonomy.

```
<bsc:parseUddiQuery var="interfaceQuery" scope="request">
    <find_tModel xmlns="urn:uddi-org:api_v3">
        <categoryBag>
            <keyedReference
                tModelKey="uddi:systinet.com:taxonomy:interface:status"
                keyValue="Stable"
            />
        </categoryBag>
    </find_tModel>
</bsc:parseUddiQuery>

<syswf:component prefix="interfaces" name="excuteFindInterfaces">
    <syswf:param name="query" value="${interfaceQuery}"/>
    <syswf:param name="var" value="searchResults"/>
</syswf:component>

Stable interface names:
<ul>
<c:forEach items="${searchResults.TModelArrayList}" var="iface">
    <li><c:out value="${iface.name.value}"/></li>
</c:forEach>
</ul>
```

bindingSearchResults

This component executes a find_tModel UDDI query and displays Bindings that match the query. Alternatively, the component may be given a list of tModels (Bindings) to display. This alternative approach is recommended when the result can not be produced by an UDDI query directly, such as when some post-processing required.

The results are displayed in a standard layout that allows the user to select a view from among the supported ones (common, business, etc.).

Table 132. `bindingSearchResults` Parameters

Param	Description	Required
`bindingList`	Must hold an instance of one of the following: org.systinet.uddi.client.v3.struct.TModelList org.systinet.uddi.client.v3.struct.TModelDetail org.systinet.uddi.client.v3.struct.TModelArrayList The query will not be executed, rather Interfaces which are given in the list will be displayed.	yes, exclusive with query
`defaultView`	The component name of the view component that should be displayed by default. Valid names are: • bindingsCommonResults - common results view • bindingsTechResults - technical results view If the parameter is not present, the serviceCommonResults view will be displayed.	no
`sortedBy`	The name of the column for the initial sort order. The list of applicable values depends on the selected default view Component.	no
`query`	Value must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_tModel` object. The query will be executed in UDDI using the logged user's credentials.	yes, exclusive with bindingList

Example 70. bindingSearchResults - Usage Example

This example displays all Bindings

```
<bsc:parseUddiQuery var="bindingQuery" scope="request">
    <find_tModel xmlns="urn:uddi-org:api_v3">
    </find_tModel>
</bsc:parseUddiQuery>

<syswf:component prefix="bindings" name="bindingSearchResults">
    <syswf:param name="query" value="${bindingQuery}"/>
</syswf:component>
```

executeFindBinding

This component executes a find_tModel UDDI query in order to find Bindings. The result of the query is stored in a local session variable for use with other components and JSP pages.

The specified result variable is considered to be a cache for the result value. If the variable is not empty, the query is not executed. You must clear the variable in order to get a fresh result set. The cache is automatically cleared when a task is selected.

Table 133. `executeFindBinding` Parameters

Param	Description	Required
`query`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.Find_tModel` object. The query will be executed in UDDI using the logged user's credentials.	yes
`var`	Used to name the result variable in the local session in which the component will store the result of the query. The stored result will be of type `org.systinet.uddi.client.v3.struct.TModelDetail`.	yes

executeFindBinding component in action

Example 71. Component Parameters

This example displays names of all Bindings

```
<bsc:parseUddiQuery var="bindingQuery" scope="request">
    <find_tModel xmlns="urn:uddi-org:api_v3">
    </find_tModel>
</bsc:parseUddiQuery>

<syswf:component prefix="bindings" name="executeFindBindings">
    <syswf:param name="query" value="${bindingQuery}"/>
    <syswf:param name="var" value="searchResults"/>
</syswf:component>

Binding names:
<ul>
<c:forEach items="${searchResults.TModelArrayList}" var="binding">
    <li><c:out value="${binding.names.value}"/></li>
</c:forEach>
</ul>
```

getOperations

This component fetches a list of operations for a Binding or a PortType from the WSDL. As the operation list is not published into UDDI, the component will fetch and parse the WSDL from its original location. It operates on a tModel that represents either a WSDL Binding or a WSDL PortType. It extracts the WSDL location from the tModel (according to the WSDL mapping TN).

The result will be available as an array of `com.systinet.uddi.bui.standard.view.OperationView`. The result is also cached in the local session, keyed by the WSDL location (URI) for faster access in subsequent calls. If a Binding is given in the binding parameter, the result will include binding Operations. If a PortType is given in the interface parameter, the result will include PortType operations and the messages used by those operations.

The component will optionally extract all messages from the operations and collect them in the specified result variable. Note that this applies to PortType operations only.

If WSDL parsing or download fails, the component will store the failure (`java.lang.Throwable`) in the specified local session variable. The caller may then display an appropriate message.

Table 134. `getOperations` Parameters

Param	Description	Required
`interface`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.TModel` class. The tModel should represent a WSDL PortType according to the WSDL mapping technical notes.	yes, exclusive with "binding"
`binding`	Must hold an instance of the `org.systinet.uddi.client.v3.struct.TModel` class. The tModel should represent a WSDL Binding according to the WSDL mapping technical notes.	yes, exclusive with "interface"
`error`	Name of the output variable that will receive the `java.lang.Throwable` instance if an error occurs during WSDL processing. The result will be stored in the request scope.	no
`allMessages`	Name of the output variable that will receive the collection of all messages (`java.util.Collection` that contains an instances of `om.systinet.uddi.bui.standard.view.MessageView`).	no
`variable`	Name of the output variable that will receive the list of operations extracted from the PortType or Binding definition. The value will be an instance of `java.util.Collection` that contains an instances of `com.systinet.uddi.bui.standard.view.OperationView`.	

Example 72. getOperations example

The following example displays the list of operations of a WSDL PortType. It prints their names and descriptions into an HTML table.

```
<!-- Get the list of operations for the WSDL PortType represented by the "tModel" -->
<syswf:component prefix="content" name="getOperations">
    <syswf:param name="variable" value="opers"/>
    <syswf:param name="interface" value="${tModel}"/>
</syswf:component>

<!-- Print out tabularized operation info -->
<table width="100%">
<tr>
    <th width="30%">Operation name</th>
    <th width="70%">Description</th>
</tr>
<c:forEach items="opers" var="operation">
    <tr>
        <!-- Print the operation name -->
        <td class="unfoldedRow" class="unfoldedHeader">
            <c:out value="${operation.name}"/>
        </td>

        <!-- Print the operation documentation -->
        <td class="unfoldedRow" class="unfoldedHeader">
            <c:out value="${operation.documentation}"/>
        </td>
    </tr>
</c:forEach>
</table>
```

getDocumentation

This component extracts the documentation of a certain type from an `org.systinet.uddi.client.v3.struct.OverviewDocArrayList` instance. The useType can be given as an exact value, or a regular expression match using the "approximateMatch" findQualifier. All OverviewURLs that match the requested useType are returned in a collection through the result variable declared in the request scope.

Table 135. `getOperations` Parameters

Param	Description	Required
`overviewDocArrayList`	An instance of `org.systinet.uddi.client.v3.struct.OverviewDocArrayList` to search in.	yes
`useType`	The required useType of the OverviewURL. It can contain a literal string or a regular expression.	yes
`variable`	Name of the variable in which the result should be stored. The variable will be declared at the request scope. The stored value will be a `java.util.Collection` that contains instances of `org.systinet.uddi.client.v3.struct.OverviewURL`.	yes
`findQualifier`	Determines whether the value in the "useType" parameter is interpreted as a literal or as a regular expression. The allowed values are: • approximateMatch - useType contains a regular expression that the OverviewURL's useType must satisfy • exactMatch - useType is a literal and the OverviewURL's useType must be equal to this literal (case-insensitive).	no

Example 73. getDocumentation example

The following example displays the WSDL URL for a WSDL service:

```
<syswf:component prefix="doc" name="getDocumentation" >
    <syswf:param name="variable" value="documentation"/>
    <syswf:param name="docArrayList" value="${tModel.overviewDocArrayList}"/>
    <syswf:param name="useType" value="documentation"/>
</syswf:component>

<!-- Print out the URL -->
<a href="<c:out value="${documentation[0].overviewURL.value}"/>">
    <c:out value="${documentation[0].overviewURL.value}"/>
</a>
```

getServiceEndpoints

This component analyzes the Binding Templates of a Business Service, and creates a list of valid WSDL Endpoints mapped to those Binding Templates. The component produces an array of `com.systinet.uddi.bui.standard.util.Endpoints` for endpoints found on the Service.

The Endpoint is identified by a URL (location); some Interfaces or some Bindings (`com.systinet.uddi.bui.standard.util.Binding`) are deployed at that Location, that in turn communicate using Interfaces (`com.systinet.uddi.bui.standard.util.Interface`). These structures will be returned for each of the Endpoints. Each of the

structures contains a reference to the underlying UDDI entity (Binding Template or tModel) so the caller can access the registered information in full.

Table 136. `getServiceEndpoints` Parameters

Param	Description	Required
`service`	The instance of `org.systinet.uddi.client.v3.struct.BusinessService`, whose Endpoints should be returned.	yes
`variable`	Name of the request-scoped variable where the component will store the result. The result will be an array of `com.systinet.uddi.bui.standard.util.Endpoint`, one instance for each Endpoint.	yes

Example 74. getServiceEndpoints Example

The following example lists names and descriptions of all interfaces implemented for a service. It iterates through all available Endpoints and the exposed Interfaces and produces the list into a HTML table.

```
<syswf:component prefix="content" name="getServiceEndpoints" >
    <syswf:param name="variable" value="endpoints"/>
    <syswf:param name="service" value="${service}"/>
</syswf:component>


<table width="100%">
    <tr>
    <th width="30%">Interface name</th>
    <th width="70%">Description</th>
    </tr>
    <c:forEach items="${endpoints}" var="endpoint">
        <c:forEach items="${endpoint.interfaces}" var="iface">
            <bsc:setLocalizedDescriptions var="descriptions"
            value="${iface.portTypeTModel.descriptionArrayList}"
                langCode="${userDefaultLanguage}"/>
            <tr>
            <td >
                <c:out value="${iface.name}"/>
            </td>
            <td><c:out value="${descriptions[0].value}"/></td>
            </tr>
        </c:forEach>
    </c:forEach>
</table>
```

selectCategory

This component helps with UDDI query construction. It takes an existing query and adds a `KeyedReference` into the query's `CategoryBag`. This will either restrict, or broaden the search, depending on the `findQualifiers` present in the query.

The component may act in several ways. It can:

- Select entities that are categorized within the given taxonomy. This is done by using keyValue of `%` and the `approximateMatch` `findQualifier`.

- Select entities that are categorized with exactly the passed value.
- Perform an `approximateMatch` on the keyValue.

The component takes the session variable identified by the passed variable name and modifies the structure to contain the addition search criteria. The following query structures are supported:

- `org.systinet.uddi.client.v3.struct.Find_binding`
- `org.systinet.uddi.client.v3.struct.Find_business`
- `org.systinet.uddi.client.v3.struct.Find_service`
- `org.systinet.uddi.client.v3.struct.Find_tModel`

Table 137. `selectCategory` Parameters

Param	Description	Required
`category`	The `tModelKey` of the category. A `KeyedReference` with this `tModelKey` will be added to the `CategoryBag` of the query.	yes
`variable`	The name of the request-scoped variable where the component will store the result. The result will be an array of `com.systinet.uddi.bui.standard.util.Endpoint`, one instance for each Endpoint.	yes

4. UDDI from Developer Tools

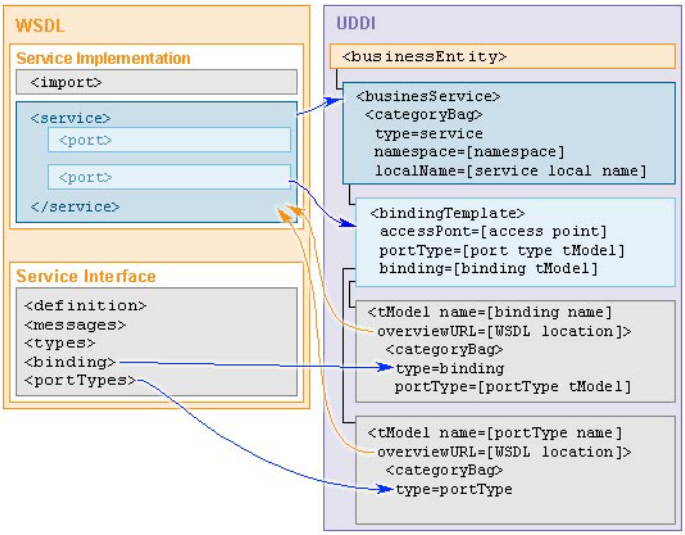
In this section, we will show you how to access UDDI from the following tools:

- Systinet Developer for Eclipse
- Microsoft Visual Studio .NET

Developer tools include wizards for searching a UDDI registry and publishing to a UDDI registry. We can say that UDDI searching and publishing rely on getting and publishing WSDL files.

Figure 16 shows how a WSDL is mapped to UDDI. For more information, see OASIS Technical Note "Using WSDL in a UDDI Registry" [http://www.oasis-open.org/committees/uddi-spec/doc/tns.htm#WSDLTNV2]

Figure 16. WSDL Mapping to UDDI



4.1. UDDI from Systinet Developer for Eclipse

Eclipse is an open source platform for tool integration. Systinet Developer for Eclipse, 5.5 extends the Eclipse IDE to support Web services creation, debugging, and deployment. Systinet Developer provides a simple point-and-click code generation experience that can turn any existing Java application into a Web service. Systinet Developer for Eclipse provides support for:

- Getting data from a UDDI registry for creating Web services and their clients, and for retrieving WSDL files to your project.
- Publishing WSDL definition to a UDDI registry

4.1.1. Getting Data from UDDI

UDDI searching wizards support the following use cases:

- Retrieving a WSDL document from a UDDI registry into your project.
- Creating Web service client applications from the WSDL document retrieved from a UDDI registry.
- Creating Web service implementations from a WSDL document retrieved from a UDDI registry.

As you see, the core is to retrieve the WSDL document from a UDDI registry. Then, the WSDL document can be used for generating a Web service implementation or a Web service client.

You can obtain the WSDL file by the following methods:

- You can get the WSDL file by WSDL service key or binding keys as shown in Figure 17. In this case, you must know exact UDDI keys. You can get these keys by searching a UDDI registry using a web interface. For searching BEA AquaLogic Service Registry, you can use both the Registry Console and the Business Service Console.

  BEA AquaLogic Service Registry is fully compliant with the latest UDDI Specification version 3. One of the benefits of the UDDI Specification version 3 is the option to use human readable UDDI keys. The first step of the UDDI inquiry wizard is selection of the version of UDDI Specification that you wish to use for accessing the UDDI registry. Systinet Developer for Eclipse 5.5 supports version 2 and version 3 of the UDDI Specification.

- You can search by qualified names of the following sections of the WSDL definition:
  - WSDL portType (interface)
  - WSDL binding (transport)
  - WSDL service (endpoint)

  You can specify a target namespace for these qualified names as shown in Figure 18. You can also combine searching the UDDI registry with searching via BEA AquaLogic Service Registry Business Service Console that use names as `interface`, `transport` and `endpoint` for sections of a WSDL file.

Figure 17. UDDI Search by Keys

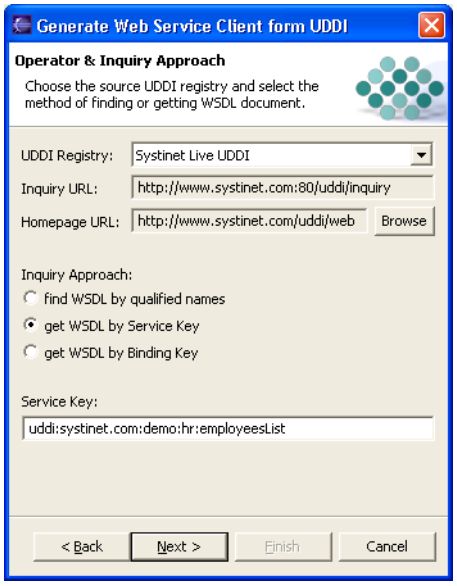
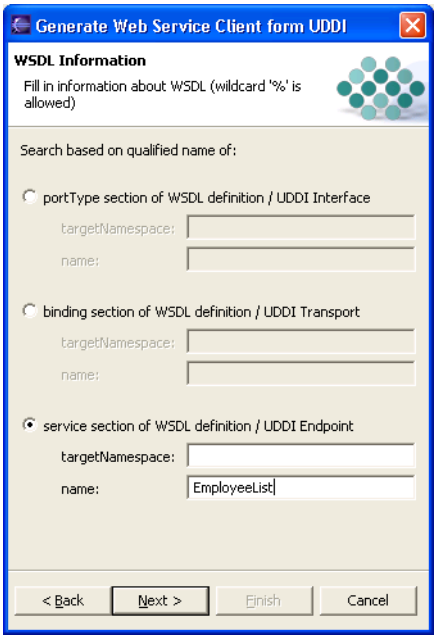


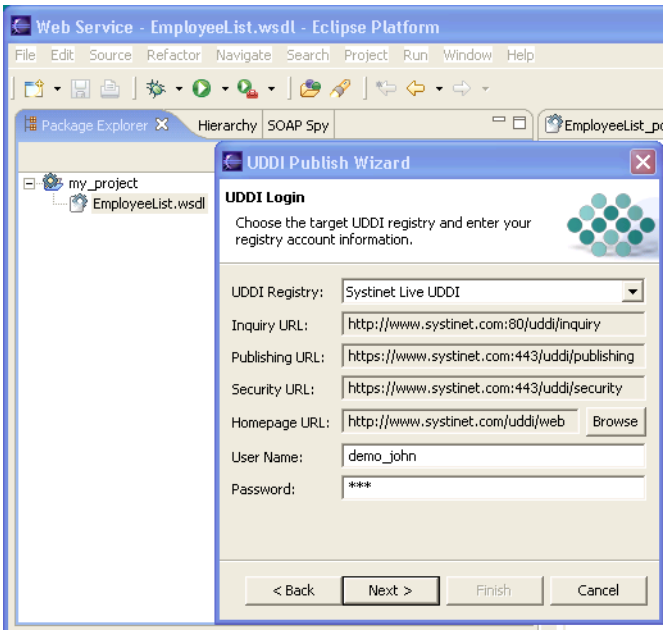
Figure 18. UDDI Search by Qualified Names



4.1.2. Publishing WSDL to UDDI

UDDI publishing wizards allows you to publish the WSDL representing the Web service to a UDDI registry. The publishing wizard supports both version 2 and version 3 of the UDDI Specification. The selected WSDL file from your project will be published to the UDDI registry under the user account you provide in the publishing wizards as shown in Figure 19. Note that before you can publish a WSDL to a UDDI registry, you must create a business entity under which the WSDL definition representing the Web service will be published as shown in Figure 16.

Figure 19. UDDI Publish Wizard



4.2. UDDI from MS Visual Studio

Microsoft Visual Studio .NET 2003 includes a wizard for accessing a UDDI registry that allows you to find a WSDL/ASMX file in the UDDI registry. Once you have found a WSDL, you can add a web reference to the Web service definition file to your project.

To start the Web Reference Wizard:

1. On the **Project** menu in Visual Studio .NET, click **Add Web Reference**.
2. The **Add Web Reference** dialog box shown in Figure 20 appears. Enter the URI of a UDDI registry or the URI of a WSDL document representing the Web service. To browse the Live BEA AquaLogic Service Registry at Systinet's web site, enter `http://systinet.com/uddi/web` or `http://systinet.com/uddi/bsc/web`.

Figure 20. Add Web Reference Default

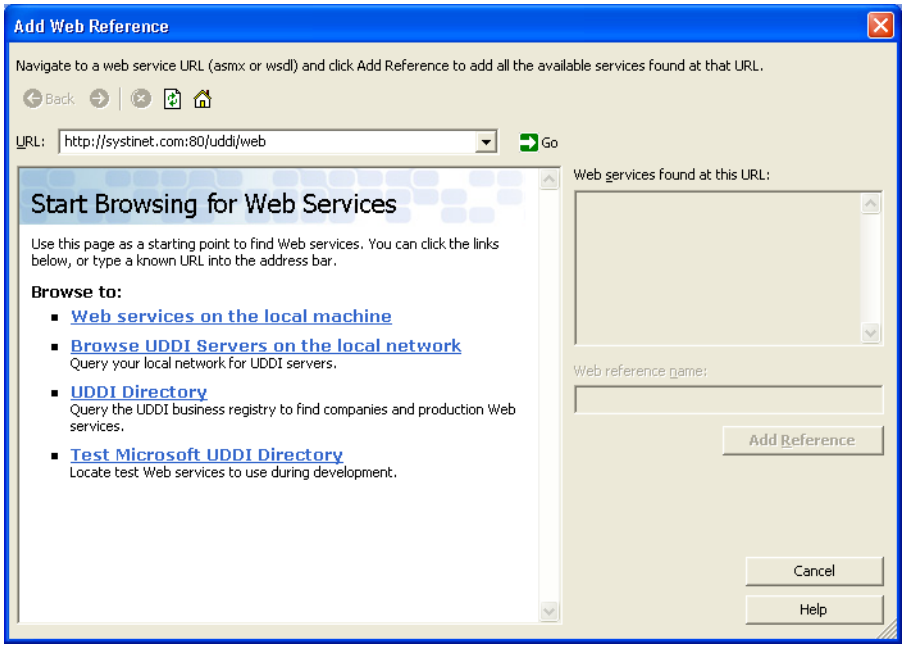


Figure 21 shows how to browse/search BEA AquaLogic Service Registry via the **Add Web Reference Wizard**.

Figure 21. Searching BEA AquaLogic Service Registry via Web Reference Wizard

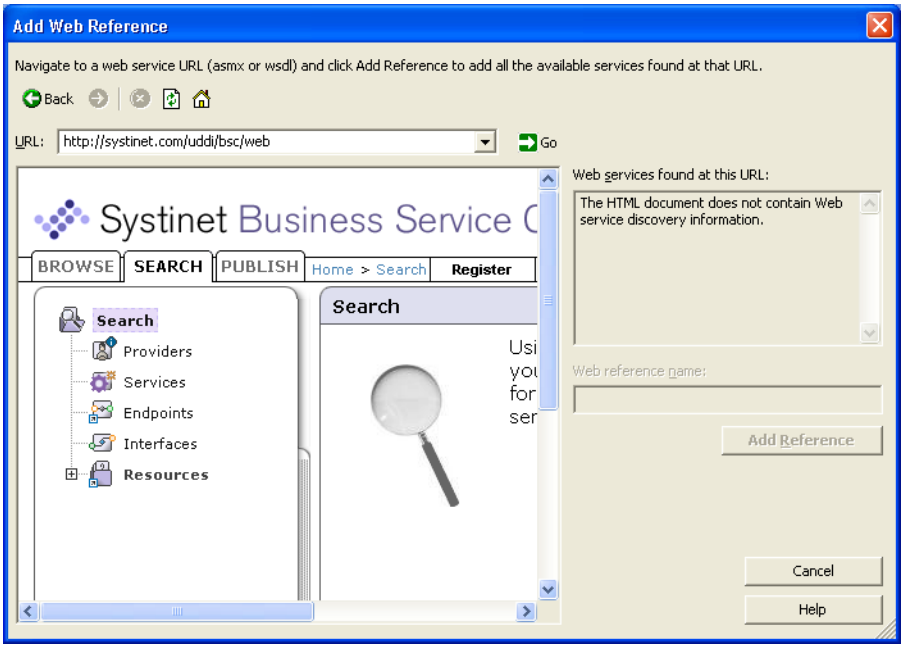
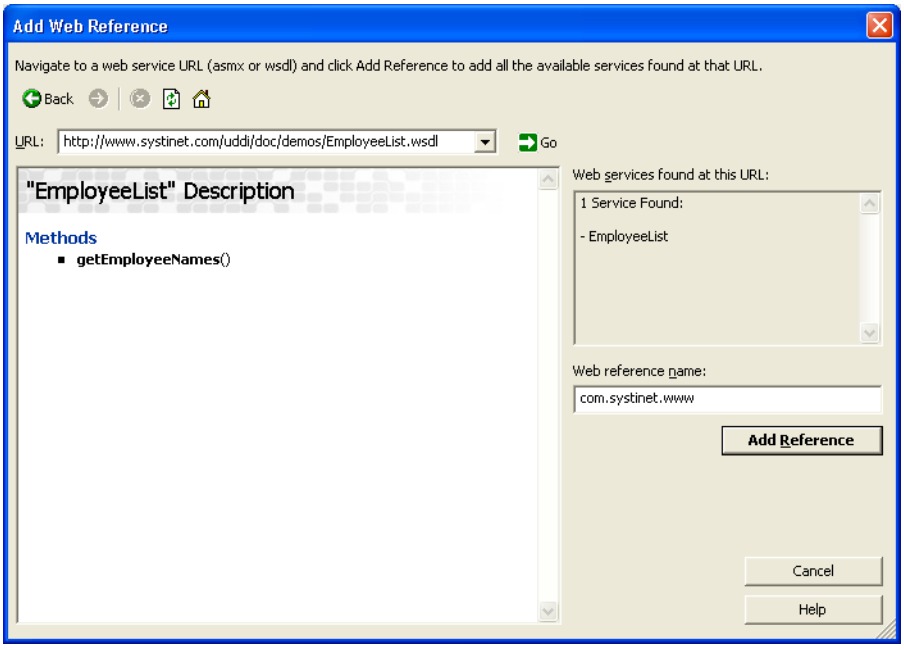


Figure 22. Add Web Reference - Found Web service



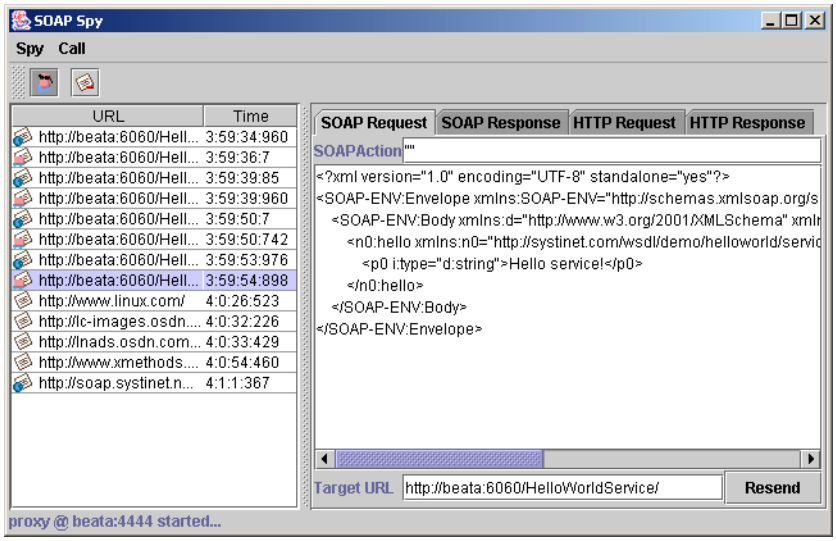
If you find a WSDL file, the wizard shown in Figure 22 parses the WSDL file displaying Web service method. Then, you can click **Add Reference** button to add the reference to your project.

5. How to Debug

5.1. SOAPSpy Tool

When debugging, it can be useful to track communication between the client and server. SOAPSpy allows the inspection of messages that the client and server exchange. Messages, or more precisely, requests and responses, are coupled to calls. Figure 23 shows the SOAPSpy dialog box.

Figure 23. SOAPSpy Tool



SOAPSpy works as an HTTP proxy server. It accepts HTTP requests from clients and resends them to their final destinations, or to another HTTP proxy server. SOAPSpy can track not only SOAP and WSDL messages, but also any other documents (HTML pages, binary data, etc.). However, the binary data is shown only schematically; all invalid text characters are translated into question mark (?) characters. SOAPSpy can also work as an HTTP server client: you can make it contact another proxy server instead of connecting to the final destination.

5.1.1. Running SOAPSpy

This tool is placed in the `bin` subdirectory of your BEA AquaLogic Service Registry server distribution. To start SOAPSpy, enter the command **SoapSpy.bat** on Windows platforms, or **./SoapSpy.sh** on UNIX machines.

Figure 24. Start Spying

Spying must be started first by selecting **Start Spying** from the **Spy** menu or by clicking the spy icon in the main panel, shown in Figure 24.

Figure 25. Status Line

The lower part of the window contains a status bar, shown in Figure 25, with information about the state of the tool. Once started, the status line displays the proxy host and port number.

The following options can be used on the command line when activating SOAPSpy:

- **--port [PORT]**

  Starts SOAPSpy at the given port

- **--help**

  Shows the help screen on the console

- **--version**

  shows the version of SOAPSpy on the console

To make SOAPSpy contact another proxy server instead of making a direct connection to the destination, use the standard Java system properties for HTTP proxies:

- **-Dhttp.proxyHost=PROXY_HOST** - The host name of the proxy server
- **-Dhttp.proxyPort=PROXY_PORT** - The port of the proxy server

There are two possible ways to load the tool:

1. **./SoapSpy**
2. **./SoapSpy --port PROXY_PORT**

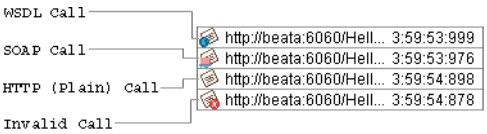
5.1.2. Using SOAPSpy

The program consists of a call list and a message viewer.

Received calls are stored in a list on the left side of the window. Calls can be selected and examined. Unwanted calls can by removed from the list using the **Call** menu or context pop-up.

The message viewer displays the selected call, as shown in Figure 26. Every call contains HTTP Request and HTTP Response tabs, which contain raw data caught by SOAPSpy. SOAP calls contain two specific panels, SOAP Request and SOAP Response, for advanced manipulation of SOAP messages. The same applies for WSDL calls.

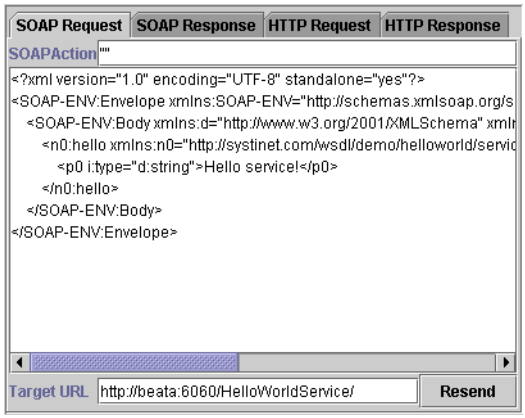
Figure 26. Call Types



5.1.3. SOAP Request Tab

The SOAP Request tab, shown in Figure 27, consists of the SOAP Action, SOAP message and Target URL where the original request was sent. Every file can be edited. Click the **Resend** to produce a new HTTP request. The resent request appears in the call list.

Figure 27. Request Tab



5.1.4. How to Run Clients Using SOAPSpy

Java system properties `http.proxyHost` and `http.proxyPort` need to be set. Use the command **java -Dhttp.proxyHost=CLIENT_COMPUTER_NAME -Dhttp.proxyPort=4444...** before running SoapSpy. E.g.:

```
java -Dhttp.proxyHost=%CLIENT_COMPUTER_NAME% -Dhttp.proxyPort=4444 org.my.FooClient
```

Important

Because SoapSpy works with the `java.net` proxy classes, it will not work with a `localhost` address. This applies to the endpoint URL that your client calls. If you do not see any activity when using SoapSpy, this is a likely cause. If you want to try running a service locally, simply obtain the machine's hostname via the `java.net.InetAddress` class.

5.2. Logging

BEA AquaLogic Service Registry wraps the Log4j [http://logging.apache.org/log4j/docs/index.html] logging service to log errors, warnings, and other information. By default:

- All such events are logged to `REGISTRY_HOME\log\logEvents.log`.
- All errors including stack traces are logged to `REGISTRY_HOME\log\errorEvents.log`.
- Behavior descriptions are configured in `REGISTRY_HOME\conf\log4j.config`.

To use the same logging mechanism in custom server code (such as the Custom Validation Service):

1. Import `com.idoox.debug.Category` to your java class:

   ```
   import com.idoox.debug.Category;
   ```

2. Create static instance with name of the category:

```
private static Category log = Category.getCategory("com.company.MyValidationService");
```

3. It is a good habit to name the category according to its class name. You can use the category

```
...
try{
      ...
} catch(Exception e){
   log.error("Fatal error", e);
    }
...
```

Demos

The BEA AquaLogic Service Registry demos suite is used to teach the capabilities of the BEA AquaLogic Service Registry APIs and how to make use of these to interact with the registry over a SOAP interface.

Note

If you want to run demos on BEA AquaLogic Service Registry ported to an application server, make sure you have properly imported the SSL certificate of the application server to the BEA AquaLogic Service Registry configuration. For more information see the Installation Guide [http://edocs.bea.com/alsr/docs21/install/]. You may also need to modify the BEA AquaLogic Service Registry URLs used in demos as shown in the demos property file, `REGISTRY_HOME/demos/env.properties`.

If you get the `java.lang.reflect.UndeclaredThrowableException`, check whether BEA AquaLogic Service Registry is running

The demos are divided into the following categories:

Basic Demos

The Basic demos cover inquiry and publishing for versions 1, 2, and 3 of the UDDI specification and WSDL2UDDI for versions 2 and 3.

Advanced Demos

The Advanced demos discuss custody, subscriptions, validation, and taxonomies.

Security Demos

In the Security demos, we cover accounts, groups, permissions, and access control lists (ACLs).

Resources Demos

In the resources demos, we cover publishing of WSDL, XML, XSD and XSLT.

1. Basic Demos

Basic Demos section includes the following demos:

- UDDI v1 demos
- UDDI v2 demos
- UDDI v3 demos

1.1. UDDI v1

- UDDI v1 Inquiry demos
- UDDI v1 Publishing demos

1.1.1. Inquiry v1

The BEA AquaLogic Service Registry basic inquiry demo set is used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to teach the reader how to use this API to perform basic inquiry calls to a UDDI registry. This documentation covers the UDDI Version 1 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1].

You will learn how to use the BEA AquaLogic Service Registry client API to contact and get information from a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `find_business` to `get_tModelDetail`.

The BEA AquaLogic Service Registry basic inquiry demo set contains following demos to assist you in learning the BEA AquaLogic Service Registry client API.

FindBinding Demonstrates how to construct and fill the `Find_binding` object, get an Inquiry stub for the UDDI registry, perform a `find_binding` call, and display the results.

FindBusiness Demonstrates how to construct and fill a `Find_business` object, get an Inquiry stub for the UDDI registry, perform a `find_business` call and display the results.

FindService Demonstrates how to construct and fill a `Find_service` object, get an Inquiry stub for the UDDI registry, perform a `find_service` call and display the results.

FindTModel Demonstrates how to construct and fill a `Find_tModel` object, get an Inquiry stub for the UDDI registry, perform a `find_tModel` call and display the results.

GetBindingDetail Demonstrates how to create a `Get_bindingDetail` object, set the bindingKey of the bindingTemplate to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_bindingDetail` call, and display the result.

GetBusinessDetail Demonstrates how to create a `Get_businessDetail` object, set the businessKey of the businessEntity to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_businessDetail` call, and display the result.

GetServiceDetail Demonstrates how to create a `Get_serviceDetail` object, set the serviceKey of the business service to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_serviceDetail` call, and display the result.

GetTModeDetail Demonstrates how to create a `Get_tModelDetail` object, set the tModelKey of the tModel to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_tModelDetail` call, and display the result.

Prerequisites and Preparatory Steps: Code

We expect, that you have already installed the BEA AquaLogic Service Registry and set the REGISTRY_HOME environment variable to its installation location.

To run the BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.bat` (`run.sh`) is located. Local properties for Basic/Inquiry demos are loaded in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v1\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v1/env.properties

Table 1. Properties Used in Demos

Name	Default Value	Description
uddi.demos.result.max_rows	5	limit on data returned from registry
uddi.demos.url.inquiry	`http://localhost:8080/uddi/inquiry`	the inquiry Web service port URL

Presentation and Functional Presentation

This section describes programing pattern used in all demos using the FindTModel demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\src\demo\uddi\v1\inquiry\FindTModel.java
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/src/demo/uddi/v1/inquiry/FindTModel.java

The main method is straightforward. It gathers user's input (tModel name), calls a method to initialize the `Find_tModel` object, executes the `find_tModel` UDDI call, and displays the list of found tModels:

```
String name = UserInput.readString("Enter name", "demo%");
Find_tModel find_tModel = createFindByTModel(name, findQualifier);
TModelList result = findTModel(find_tModel);
printTModelList(result);
```

The `createFindTModel()` method is used to create a new instance of the `Find_tModel` class and initialize it with values from parameters:

```
public static Find_tModel createFindByTModel(String name)
  throws InvalidParameterException {
    System.out.println("name = " + name);
    Find_tModel find = new Find_tModel();
    find.setName(name);
    find.setMaxRows(new Integer(MAX_ROWS));
    find.setGeneric(Constants.GENERIC_1_0);
    return find_tModel;
}
```

The helper method `getInquiryStub()` returns the UDDI Inquiry stub of the web service listening at the URL specified in the `URL_INQUIRY` property.

```
public static InquireSoap getInquiryStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.inquiry
   String url = DemoProperties.getProperty(URL_INQUIRY, "http://localhost:8080/uddi/inquiry");

    System.out.print("Using Inquiry at url " + url + " ..");
    InquireSoap inquiry = UDDIInquiryStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The UDDI API call `find_tModel` is performed in the method `findTModel`:

```
public static TModelList findTModel(Find_tModel find_tModel)
  throws UDDIException, SOAPException {
```

```
    InquireSoap inquiry = getInquiryStub();
    System.out.print("Search in progress ..");
    TModelList tModelList = inquiry.find_tModel(find_tModel);
    System.out.println(" done");
    return tModelList;
}
```

The list of found tModels is printed with the method `printTModelList`. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains the method `toXML()`, which returns a human-readable, formatted listing of its XML representation.

```
public static void printTModelList(TModelList tModelList) {
    System.out.println();

    TModelInfoArrayList tModelInfoArrayList = tModelList.getTModelInfoArrayList();
        if (tModelInfoArrayList==null) {
        System.out.println("Nothing found");
        return;
    }

    int position = 1;
    for (Iterator iterator = tModelInfoArrayList.iterator(); iterator.hasNext();) {
        TModelInfo tModelTemplate = (TModelInfo) iterator.next();
        System.out.println("TModel "+position+" : "+tModelTemplate.getTModelKey());
        System.out.println(tModelTemplate.toXML());
        System.out.println();
        System.out.println("*******************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v1
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v1

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. Run a selected demo by executing the **run** command with the name of the demo as a parameter. For example, to run the FindTModel demo, invoke

Windows:	run.bat FindBinding
UNIX:	./run.sh FindBinding

The output of this demo will resemble the following:

```
Running FindTModel demo...
**********************************************************************
***        BEA AquaLogic Service Registry Demo - FindTModelDemo                    ***
**********************************************************************

Searching for tModel where
Enter name [demo%]:
name = demo%
Using Inquiry at url http://mycomp.com:8080/uddi/inquiry .. done
Search in progress .. done

TModel 1 : uuid:13aee5be-8531-343c-98f8-d2d3a9308329
<tModelInfo tModelKey="uuid:13aee5be-8531-343c-98f8-d2d3a9308329" xmlns="urn:uddi-
org:api_v1">
<name>demo:departmentID</name>
</tModelInfo>

*******************************************************
TModel 2 : uuid:8af5f49e-e793-3719-92f3-6ab8998eb5a9
<tModelInfo tModelKey="uuid:8af5f49e-e793-3719-92f3-6ab8998eb5a9" xmlns="urn:uddi-
org:api_v1">
<name>demo:hierarchy</name>
</tModelInfo>

*******************************************************
TModel 3 : uuid:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd
<tModelInfo tModelKey="uuid:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd" xmlns="urn:uddi-
org:api_v1">
<name>Demo identifier</name>
</tModelInfo>

                    *******************************************************
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

1.1.2. Publishing v1

The BEA AquaLogic Service Registry basic publishing demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The BEA AquaLogic Service Registry basic publishing demos cover the publication aspect of the UDDI Version 1 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/contribs.htm#uddiv1]. You will learn, how to use the BEA AquaLogic Service Registry client API to publish information to a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `delete_binding` to `save_business`.

The BEA AquaLogic Service Registry basic publishing demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API.

DeleteBinding Demonstrates how to construct and fill the `Delete_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_binding` call.

DeleteBusiness Demonstrates how to construct and fill the `Delete_business` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_business` call.

DeleteService Demonstrates how to construct and fill the `Delete_service` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_service` call.

DeleteTModel Demonstrates how to construct and fill the `Delete_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_tModel` call.

GetRegisteredInfo Demonstrates how to construct and fill the `Get_registeredInfo` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_registeredInfo` call.

SaveBinding Demonstrates how to construct and fill the `Save_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_binding` call.

SaveBusiness Demonstrates how to construct and fill the `Save_business` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_business` call.

SaveService Demonstrates how to construct and fill the `Save_service` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_service` call.

SaveTModel Demonstrates how to construct and fill the `Save_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_tModel` call.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to its installation location.

To run the BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	%REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.sh`(`run.bat`) is located. Local level properties for the `Basic/Inquiry` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v1\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v1/env.properties

Table 2. Properties Used in the demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	First user's name
uddi.demos.user.john.password	demo_john	First user's password
uddi.demos.user.jane.name	demo_jane	Second user's name
uddi.demos.user.jane.password	demo_jane	Second user's password
uddi.demos.url.publishing	`http://localhost:8080/uddi/publishing`	The publication Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	The security Web service port URL

Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the SaveBusiness demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\src\demo\uddi\v1\publishing\SaveBusiness.java
UNIX:	$REGISTRY_HOME/demos/basic/publishing/src/demo/uddi/v1/publishing/SaveBusiness.java

The main method is easy to understand:

1. It gathers the user's input: an optional publisher-assigned businessKey, an array of business entity names with their language codes, and the business' description.

2. The next step is to get the security stub and authorize the user. The resulting authInfo string is a secret key passed in all requests.

3. Next, the `Save_business` object is created, filled, and passed to the `saveBusiness` method as a parameter.

   When successful, the `BusinessDetail` object is returned from the UDDI registry and printed.

4. The last step is to discard the authInfo string, so that no malicious user can use it to compromise a user's account.

```
String name = UserInput.readString("Enter business name", "Marketing");
String description = UserInput.readString("Enter description", "Saved by SaveBusiness demo");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
```

```
String authInfo = getAuthInfo(user, password, security);
Save_business save = createSaveBusiness(businessKey, names, languageCodes, description, auth-
Info);
BusinessDetail result = saveBusiness(save);
printBusinessDetail(result);
discardAuthInfo(authInfo, security);
```

The helper method, `getSecurityStub()` returns the UDDI Security stub of the web service listening at the URL specified by the `URL_SECURITY` property.

```
public static UDDI_Security_PortType getSecurityStub()
  throws SOAPException {
  // you can specify your own URL in property - uddi.demos.url.security
  String url = DemoProperties.getProperty(URL_SECURITY, "http://localhost:8080/uddi/security");

  System.out.print("Using Security at url " + url + " ..");
  UDDI_Security_PortType security = UDDISecurityStub.getInstance(url);
  System.out.println(" done");
  return security;
}
```

Similarly, the helper method `getPublishingStub()` returns the UDDI Publication stub of the Web service listening at the URL specified by the `URL_PUBLISHING` property.

```
public static UDDI_Publication_PortType getPublishingStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.publishing
    String url = DemoProperties.getProperty(URL_PUBLISHING,
        "http://localhost:8080/uddi/publishing");
    System.out.print("Using Publishing at url " + url + " ..");
    UDDI_Publication_PortType inquiry = UDDIPublishStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The `getAuthInfo()` method is used to authorize the user against the UDDI registry and to get the secret key authInfo.

```
public static String getAuthInfo(String userName,
     String password, UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging in ..");
    AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
    System.out.println(" done");
    return authToken.getAuthInfo();
}
```

The `discardAuthInfo()` method invalidates the secret key authInfo, so it cannot be reused.

```
public static DispositionReport discardAuthInfo(String authInfo,
       UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging out ..");
    DispositionReport dispositionReport = security.discard_authToken(new Discard_authToken(au-
thInfo));
    System.out.println(" done");
```

```
    return dispositionReport;
}
```

The `createSaveBusiness()` method is used to create a new instance of the `Save_business` class and initialize it with values from parameters:

```
public static Save_business createSaveBusiness(String name,
            String description, String authInfo)
  throws InvalidParameterException {
    System.out.println("name = " + name);
    System.out.println("description = " + description);

    BusinessEntity businessEntity = new BusinessEntity();
    businessEntity.setBusinessKey("");
    businessEntity.setName(name);
    businessEntity.addDescription(new Description(description));

    Save_business save = new Save_business();
    save.addBusinessEntity(businessEntity);
    save.setAuthInfo(authInfo);
    save.setGeneric(Constants.GENERIC_1_0);
    return save;
}
```

The UDDI API call `save_business` is performed in the `saveBusiness()` method:

```
public static BusinessDetail saveBusiness(Save_business save)
  throws UDDIException, SOAPException {
    UDDI_Publication_PortType publishing = getPublishingStub();
    System.out.print("Save in progress ...");
    BusinessDetail businessDetail = publishing.save_business(save);
    System.out.println(" done");
    return businessDetail;
}
```

The saved businessEntity is displayed by the `printBusinessDetail()` method. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains the `toXML()`, which returns a human-readable formatted listing of the XML representation.

```
public static void printBusinessDetail(BusinessDetail businessDetail) {
    System.out.println();
    BusinessEntityArrayList businessEntityArrayList = businessDetail.getBusinessEntityArrayL-
ist();
    int position = 1;
    for (Iterator iterator = businessEntityArrayList.iterator(); iterator.hasNext();) {
        BusinessEntity entity = (BusinessEntity) iterator.next();
        System.out.println("Business " + position + " : " + entity.getBusinessKey());
        System.out.println(entity.toXML());
        System.out.println();
        System.out.println("*********************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Publishing demo set. Let us continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v1
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v1

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of demo as a parameter. For example, to run the SaveBusiness demo, invoke

Windows:	run.bat SaveBusiness
UNIX:	./run.sh SaveBusiness

The output of this demo will resemble the following:

```
Running SaveBusiness demo...
************************************************************************
              BEA AquaLogic Service Registry Demo - SaveBusiness
************************************************************************

Saving business entity where
Enter business name [Marketing]:
Enter description [Saved by SaveBusiness demo]:

Using Publishing at url https://mycomp.com:8443/uddi/publishing .. done
Logging in .. done
name = Marketing
```

```
description = Saved by SaveBusiness demo
Save in progress ... done

Business 1 : 79596f30-a5a9-11d8-91cd-5c1d367091cd
<businessEntity businessKey="79596f30-a5a9-11d8-91cd-5c1d367091cd" operator="Systinet"
  authorizedName="demo_john" xmlns="urn:uddi-org:api">
    <name>Marketing</name>
    <description>Saved by SaveBusiness demo</description>
</businessEntity>

*******************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

1.2. UDDI v2

- UDDI v2 Inquiry demos
- UDDI v2 Publishing demos

1.2.1. Inquiry v2

The BEA AquaLogic Service Registry basic inquiry demo set is used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to teach the reader how to use this API to perform basic inquiry calls to a UDDI registry.

The BEA AquaLogic Service Registry basic inquiry demos cover inquiry aspects of the UDDI Version 2.0.4 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2]. You will learn how to use the BEA AquaLogic Service Registry client API to contact and get information from a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `find_business` to `get_tModelDetail`.

The BEA AquaLogic Service Registry basic inquiry demo set contains following demos to assist you in learning the BEA AquaLogic Service Registry client API.

FindBinding Demonstrates how to construct and fill the `Find_binding` object, get an Inquiry stub for the UDDI registry, perform a `find_binding` call, and display the results.

FindBusiness Demonstrates how to construct and fill a `Find_business` object, get an Inquiry stub for the UDDI registry, perform a `find_business` call and display the results.

FindRelatedBusiness Demonstrates how to construct and fill a `Find_relatedBusiness` object, get an Inquiry stub for the UDDI registry, perform a `find_relatedBusiness` call and display the results.

FindService Demonstrates how to construct and fill a `Find_service` object, get an Inquiry stub for the UDDI registry, perform a `find_service` call and display the results.

FindTModel Demonstrates how to construct and fill a `Find_tModel` object, get an Inquiry stub for the UDDI registry, perform a `find_tModel` call and display the results.

GetBindingDetail Demonstrates how to create a `Get_bindingDetail` object, set the bindingKey of the bindingTemplate to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_bindingDetail` call, and display the result.

GetBusinessDetail Demonstrates how to create a `Get_businessDetail` object, set the businessKey of the businessEntity to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_businessDetail` call, and display the result.

GetServiceDetail Demonstrates how to create a `Get_serviceDetail` object, set the serviceKey of the business service to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_serviceDetail` call, and display the result.

GetTModeDetail Demonstrates how to create a `Get_tModelDetail` object, set the tModelKey of the tModel to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_tModelDetail` call, and display the result.

Prerequisites and Preparatory Steps: Code

We expect, that you have already installed the BEA AquaLogic Service Registry registry and set the REGISTRY_HOME environment variable to its installation location.

To run BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.bat` (`run.sh`) is located. Local level properties for Basic/Inquiry demos are loaded in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v2\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v2/env.properties

Table 3. Properties Used in Demos

Name	Default Value	Description
uddi.demos.result.max_rows	5	limit of data returned from registry
uddi.demos.url.inquiry	`http://localhost:8080/uddi/inquiry`	the inquiry Web service port URL

Presentation and Functional Presentation

This section describes the programing pattern used in all demos using the FindTModel demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\src\demo\uddi\v2\inquiry\FindTModel.java
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/src/demo/uddi/v2/inquiry/FindTModel.java

The main method is straightforward. It gathers user's input (tModel name), calls a method to initialize the `Find_tModel` object, executes the `find_tModel` UDDI call, and displays the list of found tModels:

```
String name = UserInput.readString("Enter name", "demo%");
Find_tModel find_tModel = createFindByTModel(name, findQualifier);
TModelList result = findTModel(find_tModel);
printTModelList(result);
```

The `createFindTModel()` method is used to create new instance of the `Find_tModel` class and initialize it with values from parameters:

```
public static Find_tModel createFindByTModel(String name)
  throws InvalidParameterException {
    System.out.println("name = " + name);
    Find_tModel find = new Find_tModel();
    find.setName(new Name(name));
    find.setMaxRows(new Integer(MAX_ROWS));
    find.setGeneric(Constants.GENERIC_2_0);
    return find_tModel;
}
```

The helper method `getInquiryStub()` returns the UDDI Inquiry stub of the web service listening at the URL specified in the `URL_INQUIRY` property.

```
public static UDDI_Inquiry_PortType getInquiryStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.inquiry
    String url = DemoProperties.getProperty(URL_INQUIRY, "http://localhost:8080/uddi/inquiry");

    System.out.print("Using Inquiry at url " + url + " ..");
    UDDI_Inquiry_PortType inquiry = UDDIInquiryStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The UDDI API call `find_tModel` is performed in the method `findTModel`:

```
public static TModelList findTModel(Find_tModel find_tModel)
  throws UDDIException, SOAPException {
    UDDI_Inquiry_PortType inquiry = getInquiryStub();
    System.out.print("Search in progress ..");
    TModelList tModelList = inquiry.find_tModel(find_tModel);
    System.out.println(" done");
    return tModelList;
}
```

The list of found tModels is printed with the method `printTModelList`. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains method `toXML()`, which returns a human-readable, formatted listing of its XML representation.

```
public static void printTModelList(TModelList tModelList) {
    System.out.println();

    TModelInfoArrayList tModelInfoArrayList = tModelList.getTModelInfoArrayList();
        if (tModelInfoArrayList==null) {
        System.out.println("Nothing found");
        return;
    }
```

```
    int position = 1;
    for (Iterator iterator = tModelInfoArrayList.iterator(); iterator.hasNext();) {
        TModelInfo tModelTemplate = (TModelInfo) iterator.next();
        System.out.println("TModel "+position+" : "+tModelTemplate.getTModelKey());
        System.out.println(tModelTemplate.toXML());
        System.out.println();
        System.out.println("*******************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v2
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v2

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. Run a selected demo by executing the **run** command with the name of the demo as a parameter. For example, to run the FindTModel demo, invoke

Windows:	run.bat FindTModel
UNIX:	./run.sh FindTModel

The output of this demo will resemble the following:

```
Running FindTModel demo...
*********************************************************************
***        BEA AquaLogic Service Registry Demo - FindTModelDemo                    ***
*********************************************************************

Searching for tModel where
Enter name [demo%]:
name = demo%
Using Inquiry at url http://mycomp.com:8080/uddi/inquiry .. done
Search in progress .. done

TModel 1 : uuid:13aee5be-8531-343c-98f8-d2d3a9308329
<tModelInfo tModelKey="uuid:13aee5be-8531-343c-98f8-d2d3a9308329" xmlns="urn:uddi-
org:api_v2">
<name>demo:departmentID</name>
</tModelInfo>

*******************************************************
TModel 2 : uuid:8af5f49e-e793-3719-92f3-6ab8998eb5a9
<tModelInfo tModelKey="uuid:8af5f49e-e793-3719-92f3-6ab8998eb5a9" xmlns="urn:uddi-
org:api_v2">
<name>demo:hierarchy</name>
</tModelInfo>

*******************************************************
TModel 3 : uuid:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd
<tModelInfo tModelKey="uuid:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd" xmlns="urn:uddi-
org:api_v2">
<name>Demo identifier</name>
</tModelInfo>

                    *******************************************************
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

1.2.2. Publishing v2

The BEA AquaLogic Service Registry basic publishing demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The BEA AquaLogic Service Registry basic publishing demos cover the publication aspect of the UDDI Version 2 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2]. You will learn how to use the BEA AquaLogic Service Registry client API to publish information to a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `add_publisherAssertion` through `get_registeredInfo` to `save_business`.

The BEA AquaLogic Service Registry basic publishing demo set contains the following demos. They will assist you in learning the BEA AquaLogic Service Registry client API.

AddAssertion Demonstrates how to construct and fill the `Add_publisherAssertion` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `add_publisherAssertion` call.

DeleteAssertion Demonstrates how to construct and fill the `Delete_publisherAssertion` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_publisherAssertion` call.

DeleteBinding Demonstrates how to construct and fill the `Delete_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_binding` call.

DeleteBusiness Demonstrates how to construct and fill the `Delete_business` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_business` call.

DeleteService Demonstrates how to construct and fill the `Delete_service` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_service` call.

DeleteTModel Demonstrates how to construct and fill the `Delete_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_tModel` call.

GetAssertionStatusReport Demonstrates how to construct and fill the `Get_assertionStatusReport` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_assertionStatusReport` call.

GetPublisherAssertions Demonstrates how to construct and fill the `Get_publisherAssertions` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_publisherAssertions` call.

GetRegisteredInfo Demonstrates how to construct and fill the `Get_registeredInfo` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_registeredInfo` call.

SaveBinding Demonstrates how to construct and fill the `Save_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_binding` call.

SaveBusiness Demonstrates how to construct and fill the `Save_business` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_business` call.

SaveService Demonstrates how to construct and fill the `Save_service` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_service` call.

SaveTModel Demonstrates how to construct and fill the `Save_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_tModel` call.

SetAssertions Demonstrates how to construct and fill the `Set_publisherAssertions` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `set_publisherAssertions` call.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to its installation location.

To run the BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	cd $REGISTRY_HOME/bin/serverstart.sh

It is neccessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.sh`(`run.bat`) is located. Local level properties for the `Basic/Inquiry` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v2\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v2/env.properties

Table 4. Properties Used in the Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	First user's name
uddi.demos.user.john.password	demo_john	First user's password
uddi.demos.user.jane.name	demo_jane	Second user's name
uddi.demos.user.jane.password	demo_jane	Second user's password
uddi.demos.url.publishing	`http://localhost:8080/uddi/publishing`	The publication Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	The security Web service port URL

Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the SaveBusiness demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\src\demo\uddi\v2\publishing\SaveBusiness.java
UNIX:	$REGISTRY_HOME/demos/basic/publishing/src/demo/uddi/v2/publishing/SaveBusiness.java

The main method is easy to understand. First it gathers the user's input. Namely optional publisher assigned businessKey, then an array of business entity names with their language codes and finally a description of the business.

The next step is to get the security stub and authorize the user. The resulting authInfo string is a secret key passed in all requests.

Next, the `Save_business` object is created, filled, and passed to the `saveBusiness` method as a parameter.

When successful, the `BusinessDetail` object is returned from the UDDI registry and printed. The last step is to discard the authInfo string, so it cannot be used to compromise a user's account.

```
int count = UserInput.readInt("Enter count of names", 1);
String[] names = new String[count];
String[] languageCodes = new String[count];
for (int i = 0; i < count; i++) {
    String tmp = UserInput.readString("Enter language code", "");
    languageCodes[i] = (tmp.length() > 0) ? tmp : null;
    names[i] = UserInput.readString("Enter name in language " + tmp, "Marketing");
}
String description = UserInput.readString("Enter description",
                                 "Saved by SaveBusiness demo");
System.out.println();
```

```
UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Save_business save = createSaveBusiness(businessKey, names, languageCodes, description, auth-
Info);
BusinessDetail result = saveBusiness(save);
printBusinessDetail(result);
discardAuthInfo(authInfo, security);
```

The helper method, `getSecurityStub()` returns the UDDI Security stub of the Web service listening at the URL specified by the `URL_SECURITY` property.

```
public static UDDI_Security_PortType getSecurityStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.security
   String url = DemoProperties.getProperty(URL_SECURITY, "http://localhost:8080/uddi/security");

    System.out.print("Using Security at url " + url + " ..");
    UDDI_Security_PortType security = UDDISecurityStub.getInstance(url);
    System.out.println(" done");
    return security;
}
```

The helper method `getPublishingStub()` returns the UDDI Publication stub of the Web service listening at the URL specified by the `URL_PUBLISHING` property.

```
public static UDDI_Publication_PortType getPublishingStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.publishing
    String url = DemoProperties.getProperty(URL_PUBLISHING,
                    "http://localhost:8080/uddi/publishing");
    System.out.print("Using Publishing at url " + url + " ..");
    UDDI_Publication_PortType inquiry = UDDIPublishStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The `getAuthInfo()` method is used to authorize the user against the UDDI registry and to get the secret authInfo key.

```
public static String getAuthInfo(String userName,
                    String password, UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging in ..");
    AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
    System.out.println(" done");
    return authToken.getAuthInfo();
}
```

The `discardAuthInfo()` method invalidates the secret authInfo key, so it cannot be used anymore.

```
public static DispositionReport discardAuthInfo(String authInfo,
               UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging out ..");
    DispositionReport dispositionReport = security.discard_authToken(new Discard_authToken(au-
thInfo));
```

```
    System.out.println(" done");
    return dispositionReport;
}
```

The `createSaveBusiness()` method is used to create a new instance of the `Save_business` class and initialize it with values from parameters:

```
public static Save_business createSaveBusiness(String[] names,
String[] nameLangCodes, String description, String authInfo)
  throws InvalidParameterException {
    for (int i = 0; i < names.length; i++) {
        System.out.println("lang = " + nameLangCodes[i] + ", name = " + names[i]);
    }
    System.out.println("description = " + description);

    BusinessEntity businessEntity = new BusinessEntity();
    businessEntity.setBusinessKey("");
    for (int i = 0; i < names.length; i++) {
        if (nameLangCodes[i] == null) {
            businessEntity.addName(new Name(names[i]));
        } else {
            businessEntity.addName(new Name(names[i], nameLangCodes[i]));
        }
    }
    businessEntity.addDescription(new Description(description));

    Save_business save = new Save_business();
    save.addBusinessEntity(businessEntity);
    save.setAuthInfo(authInfo);
    save.setGeneric(Constants.GENERIC_2_0);
    return save;
}
```

The UDDI API call `save_business` is performed in the method `saveBusiness()`:

```
public static BusinessDetail saveBusiness(Save_business save)
  throws UDDIException, SOAPException {
    UDDI_Publication_PortType publishing = getPublishingStub();
    System.out.print("Save in progress ...");
    BusinessDetail businessDetail = publishing.save_business(save);
    System.out.println(" done");
    return businessDetail;
}
```

The saved businessEntity is displayed by the `printBusinessDetail()` method. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains the `toXML()`, which returns a human-readable formatted listing of the XML representation.

```
public static void printBusinessDetail(BusinessDetail businessDetail) {
    System.out.println();
    BusinessEntityArrayList businessEntityArrayList = businessDetail.getBusinessEntityArrayL-
ist();
    int position = 1;
    for (Iterator iterator = businessEntityArrayList.iterator(); iterator.hasNext();) {
        BusinessEntity entity = (BusinessEntity) iterator.next();
```

```
        System.out.println("Business " + position + " : " + entity.getBusinessKey());
        System.out.println(entity.toXML());
        System.out.println();
        System.out.println("*******************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Publishing demo set. Let us continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v2
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v2

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example to run the SaveBusiness demo, invoke

Windows:	run.bat SaveBusiness
UNIX:	./run.sh SaveBusiness

The output of this demo will resemble the following:

```
Running SaveBusiness demo...
***********************************************************************
***    BEA AquaLogic Service Registry Demo - SaveBusiness    ***
***********************************************************************
```

```
Saving business entity where
Enter count of names [1]:
Enter language code []:
Enter name in language  [Marketing]:
Enter description [Saved by SaveBusiness demo]:

Using Publishing at url https://mycomp.com:8443/uddi/publishing .. done
Logging in .. done
lang = null, name = Marketing
description = Saved by SaveBusiness demo
Save in progress ... done

Business 1 : c9e8be50-a5a5-11d8-91cd-5c1d367091cd
<businessEntity businessKey="c9e8be50-a5a5-11d8-91cd-5c1d367091cd" operator="Systinet"
authorizedName="demo_john" xmlns="urn:uddi-org:api_v2">
    <name>Marketing</name>
    <description>Saved by SaveBusiness demo</description>
</businessEntity>

*******************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

1.3. UDDI v3

- UDDI v3 Inquiry demos
- UDDI v3 Publishing demos

1.3.1. Inquiry v3

The BEA AquaLogic Service Registry basic inquiry demo set is used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to teach the reader how to use this API to perform basic inquiry calls to a UDDI registry.

The BEA AquaLogic Service Registry basic inquiry demos cover the inquiry aspect of the UDDI Version 3.0.1 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]. You will learn how to use the BEA AquaLogic Service Registry client API to contact and get information from a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `find_business` to `get_tModel`.

The BEA AquaLogic Service Registry basic inquiry demo set contains following demos. They will assist you in learning the BEA AquaLogic Service Registry client API.

FindBinding Demonstrates how to construct and fill the `Find_binding` object, get an Inquiry stub for the UDDI registry, perform a `find_binding` call, and display the results.

FindBusiness Demonstrates how to construct and fill a `Find_business` object, get an Inquiry stub for the UDDI registry, perform a `find_business` call and display the results.

FindRelatedBusiness Demonstrates how to construct and fill a `Find_relatedBusiness` object, get an Inquiry stub for the UDDI registry, perform a `find_relatedBusiness` call and display the results.

FindService Demonstrates how to construct and fill a `Find_service` object, get an Inquiry stub for the UDDI registry, perform a `find_service` call and display the results.

FindTModel Demonstrates how to construct and fill a `Find_tModel` object, get an Inquiry stub for the UDDI registry, perform a `find_tModel` call and display the results.

GetBindingDetail Demonstrates how to create a `Get_bindingDetail` object, set the bindingKey of the bindingTemplate to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_bindingDetail` call, and display the result.

GetBusinessDetail Demonstrates how to create a `Get_businessDetail` object, set the businessKey of the businessEntity to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_businessDetail` call, and display the result.

GetOperationalInfo Demonstrates how to create a `Get_operationalInfo` object, set a UDDI key, get an Inquiry stub for the UDDI registry, perform a `get_operationalInfo` call, and display the operational info of the selected UDDI structure.

GetServiceDetail Demonstrates how to create a `Get_serviceDetail` object, set the serviceKey of the business service to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_serviceDetail` call, and display the result.

GetTModeDetail Demonstrates how to create a `Get_tModelDetail` object, set the tModelKey of the tModel to be fetched, get an Inquiry stub for the UDDI registry, perform a `get_tModelDetail` call, and display the result.

Prerequisites and Preparatory Steps: Code

We expect, that you have already installed the BEA AquaLogic Service Registry and set the REGISTRY_HOME environment variable to its installation location.

To run BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the UDDI registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.bat` (`run.sh`) is located. Local level properties for Basic/Inquiry demos are loaded in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v3\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v3/env.properties

Table 5. Properties Used in Demos

Name	Default value	Description
uddi.demos.result.max_rows	5	limit of data returned from registry
uddi.demos.url.inquiry	`http://localhost:8080/uddi/inquiry`	the inquiry Web service port URL

Presentation and Functional Presentation

This section describes programing pattern used in all demos using the FindTModel demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\src\demo\uddi\v3\inquiry\FindTModel.java
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/src/demo/uddi/v3/inquiry/FindTModel.java

The main method is straightforward. It gathers user's input (tModel name and findQualifier name), calls a method to initialize the `Find_tModel` object, executes the `find_tModel` UDDI call, and displays the list of found tModels:

```
String name = UserInput.readString("Enter name", "demo%");
String findQualifier = UserInput.readString("Enter findQualifier", "approximateMatch");
Find_tModel find_tModel = createFindByTModel(name, findQualifier);
TModelList result = findTModel(find_tModel);
printTModelList(result);
```

The `createFindTModel()` method is used to create new instance of Find_tModel class and initialize it with values from parameters:

```
public static Find_tModel createFindByTModel(String name, String findQualifier)
        throws InvalidParameterException {
    System.out.println("findQualifier = " + findQualifier);
    System.out.println("name = " + name);
    Find_tModel find_tModel = new Find_tModel();
    find_tModel.setName(new Name(name));
    find_tModel.setMaxRows(new Integer(MAX_ROWS));
    find_tModel.addFindQualifier(findQualifier);
    return find_tModel;
}
```

The helper method `getInquiryStub()` returns the UDDI Inquiry stub of the web service listening at the URL specified in the `URL_INQUIRY` property.

```
public static UDDI_Inquiry_PortType getInquiryStub()
        throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.inquiry
    String url = DemoProperties.getProperty(URL_INQUIRY, "http://localhost:8080/uddi/inquiry");

    System.out.print("Using Inquiry at url " + url + " ..");
    UDDI_Inquiry_PortType inquiry = UDDIInquiryStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The UDDI API call `find_tModel` is performed in the method `findTModel`:

```
public static TModelList findTModel(Find_tModel find_tModel)
        throws UDDIException, SOAPException {
    UDDI_Inquiry_PortType inquiry = getInquiryStub();
    System.out.print("Search in progress ..");
    TModelList tModelList = inquiry.find_tModel(find_tModel);
    System.out.println(" done");
    return tModelList;
}
```

The list of found tModels are printed with the method `printTModelList`. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains method `toXML()`, which returns a human-readable, formatted, listing of its XML representation.

```
public static void printTModelList(TModelList tModelList) {
    System.out.println();
    ListDescription listDescription = tModelList.getListDescription();
    if (listDescription!=null) {
        // list description is mandatory part of result,
 // if the resultant list is subset of available data
        int includeCount = listDescription.getIncludeCount();
        int actualCount = listDescription.getActualCount();
        int listHead = listDescription.getListHead();
        System.out.println("Displaying "+includeCount+" of "+
                           actualCount+", starting at position " + listHead);
    }

    TModelInfoArrayList tModelInfoArrayList = tModelList.getTModelInfoArrayList();
    if (tModelInfoArrayList==null) {
        System.out.println("Nothing found");
        return;
    }

    int position = 1;
    for (Iterator iterator = tModelInfoArrayList.iterator(); iterator.hasNext();) {
        TModelInfo tModelTemplate = (TModelInfo) iterator.next();
        System.out.println("TModel "+position+" : "+tModelTemplate.getTModelKey());
        System.out.println(tModelTemplate.toXML());
        System.out.println();
        System.out.println("*******************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Inquiry demo set. Our example continues with the FindTModel demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\basic\inquiry\v3
UNIX:	$REGISTRY_HOME/demos/basic/inquiry/v3

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. Run a selected demo by executing the **run** command with the name of the demo as a parameter. For example, to run the FindTModel demo, invoke

Windows:	run.bat FindBinding
UNIX:	./run.sh FindBinding

The output of this demo will resemble the following:

```
*****************************************
***    BEA AquaLogic Service Registry Demo - FindTModelDemo     ***
            ****************************************

Searching for tModel where
Enter name [demo%]:
Enter findQualifier [approximateMatch]:
findQualifier = approximateMatch
name = demo%
Using Inquiry at url http://localhost:8080/uddi/inquiry .. done
Search in progress .. done

Displaying 3 of 3, starting at position 1
TModel 1 : uddi:systinet.com:demo:departmentID

<tModelInfo tModelKey="uddi:systinet.com:demo:departmentID"
               xmlns="urn:uddi-org:api_v3">
  <name>demo:departmentID</name>
  <description>Identifier of the department</description>
</tModelInfo>

*******************************************************
TModel 2 : uddi:systinet.com:demo:hierarchy

<tModelInfo tModelKey="uddi:systinet.com:demo:hierarchy"
```

```
                xmlns="urn:uddi-org:api_v3">
  <name>demo:hierarchy</name>
  <description>Business hierarchy taxonomy</description>
</tModelInfo>

*******************************************************
TModel 3 : uddi:systinet.com:demo:location:floor

<tModelInfo tModelKey="uddi:systinet.com:demo:location:floor" xmlns="
            urn:uddi-org:api_v3">
  <name>demo:location:floor</name>
  <description>Specifies floor, on which the department is located</description>
</tModelInfo>

*******************************************************
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

1.3.2. Publishing v3

The BEA AquaLogic Service Registry basic publishing demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and teaches how to use this API to perform basic publishing calls to a UDDI registry.

The BEA AquaLogic Service Registry basic publishing demos cover the publication aspect of the UDDI Version 3 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]. You will learn, how to use the BEA AquaLogic Service Registry client API to publish information to a UDDI registry over a SOAP interface. There is one demo for each UDDI call, from `add_publisherAssertion` through `get_registeredInfo` to `save_business`.

The BEA AquaLogic Service Registry basic publishing demo set contains the following demos. They will assist you in learning the BEA AquaLogic Service Registry client API.

AddAssertion Demonstrates how to construct and fill the `Add_publisherAssertion` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `add_publisherAssertion` call.

DeleteAssertion Demonstrates how to construct and fill the `Delete_publisherAssertion` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_publisherAssertion` call.

DeleteBinding Demonstrates how to construct and fill the `Delete_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_binding` call.

DeleteBusiness Demonstrates how to construct and fill the `Delete_business` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_business` call.

DeleteService Demonstrates how to construct and fill the `Delete_service` object, get Publishing stub for the UDDI registry, get an authToken, and perform the `delete_service` call.

DeleteTModel Demonstrates how to construct and fill the `Delete_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `delete_tModel` call.

GetAssertionStatusReport Demonstrates how to construct and fill the `Get_assertionStatusReport` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_assertionStatusReport` call.

GetPublisherAssertions Demonstrates how to construct and fill the `Get_publisherAssertions` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_publisherAssertions` call.

GetRegisteredInfo Demonstrates how to construct and fill the `Get_registeredInfo` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `get_registeredInfo` call.

SaveBinding Demonstrates how to construct and fill the `Save_binding` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_binding` call.

SaveBusiness Demonstrates how to construct and fill the `Save_business` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_business` call.

SaveService Demonstrates how to construct and fill the `Save_service` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_service` call.

SaveTModel Demonstrates how to construct and fill the `Save_tModel` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `save_tModel` call.

SetAssertions Demonstrates how to construct and fill the `Set_publisherAssertions` object, get a Publishing stub for the UDDI registry, get an authToken, and perform the `set_publisherAssertions` call.

Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to its installation location.

To run the BEA AquaLogic Service Registry's demos, your UDDI registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is neccessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit the file `env.properties` in the directory where `run.sh`(`run.bat`) is located. Local level properties for the `Basic/Inquiry` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v3\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v3/env.properties

Table 6. Properties Used in the Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	First user's name
uddi.demos.user.john.password	demo_john	First user's password
uddi.demos.user.jane.name	demo_jane	Second user's name
uddi.demos.user.jane.password	demo_jane	Second user's password
uddi.demos.url.publishing	`http://localhost:8080/uddi/publishing`	The publication Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	The security web service port URL

Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the SaveBusiness demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\src\demo\uddi\v3\publishing\SaveBusiness.java
UNIX:	$REGISTRY_HOME/demos/basic/publishing/src/demo/uddi/v3/publishing/SaveBusiness.java

The main method is easy to understand. First it gathers the user's input: an optional publisher-assigned businessKey, then variable long array of business entity names with their language codes, and a description of the business.

The next step is to get the security stub and authorize the user. The resulting authInfo string is a secret key passed in all requests.

Next, the `Save_business` object is created, filled, and passed to the `saveBusiness` method as a parameter.

When successful, the `BusinessDetail` object is returned from the UDDI registry and printed. The last step is to discard the authInfo string, so no malicious user can use it to compromise a user's account.

```
String businessKey = UserInput.readString("Enter (optional) businessKey", "");
int count = UserInput.readInt("Enter count of names", 1);
String[] names = new String[count];
String[] languageCodes = new String[count];
for (int i = 0; i < count; i++) {
    String tmp = UserInput.readString("Enter language code", "");
    languageCodes[i] = (tmp.length() > 0) ? tmp : null;
    names[i] = UserInput.readString("Enter name in language " + tmp, "Marketing");
}
String description = UserInput.readString("Enter description", "Saved by SaveBusiness demo");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Save_business save = createSaveBusiness(businessKey, names, languageCodes, description, auth-
Info);
BusinessDetail result = saveBusiness(save);
printBusinessDetail(result);
discardAuthInfo(authInfo, security);
```

The helper method, `getSecurityStub()` returns the UDDI Security stub of the web service listening at the URL specified by the `URL_SECURITY` property.

```
public static UDDI_Security_PortType getSecurityStub()
 throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.security
   String url = DemoProperties.getProperty(URL_SECURITY, "http://localhost:8080/uddi/security");

    System.out.print("Using Security at url " + url + " ..");
    UDDI_Security_PortType security = UDDISecurityStub.getInstance(url);
    System.out.println(" done");
    return security;
}
```

Similarly, the helper method `getPublishingStub()` returns the UDDI Publication stub of the web service listening at the URL specified by the `URL_PUBLISHING` property.

```
public static UDDI_Publication_PortType getPublishingStub()
 throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.publishing
   String url = DemoProperties.getProperty(URL_PUBLISHING, "http://localhost:8080/uddi/publish-
ing");
    System.out.print("Using Publishing at url " + url + " ..");
    UDDI_Publication_PortType inquiry = UDDIPublishStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The `getAuthInfo()` method is used to authorize the user against the UDDI registry and to get the secret authInfo key.

```
public static String getAuthInfo(String userName, String password, UDDI_Security_PortType se-
curity)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging in ..");
    AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
    System.out.println(" done");
    return authToken.getAuthInfo();
}
```

The `discardAuthInfo()` method invalidates the secret authInfo key, so it cannot be used anymore.

```
public static void discardAuthInfo(String authInfo, UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging out ..");
    security.discard_authToken(new Discard_authToken(authInfo));
    System.out.println(" done");
}
```

The `createSaveBusiness()` method is used to create a new instance of the `Save_business` class and initialize it with values from parameters:

```
public static Save_business createSaveBusiness(String businessKey, String[] names,
  String[] nameLangCodes, String description, String authInfo)
  throws InvalidParameterException {
    System.out.println("businessKey = " + businessKey);
    for (int i = 0; i < names.length; i++) {
```

```
        System.out.println("lang = " + nameLangCodes[i] + ", name = " + names[i]);
    }
    System.out.println("description = " + description);

    BusinessEntity businessEntity = new BusinessEntity();
    if (businessKey!=null && businessKey.length()>0)
        businessEntity.setBusinessKey(businessKey);
    for (int i = 0; i < names.length; i++) {
        if (nameLangCodes[i] == null) {
            businessEntity.addName(new Name(names[i]));
        } else {
            businessEntity.addName(new Name(names[i], nameLangCodes[i]));
        }
    }
    businessEntity.addDescription(new Description(description));

    Save_business save = new Save_business();
    save.addBusinessEntity(businessEntity);
    save.setAuthInfo(authInfo);
    return save;
}
```

The UDDI API call `save_business` is performed in the method `saveBusiness()`:

```
public static BusinessDetail saveBusiness(Save_business save)
  throws UDDIException, SOAPException {
    UDDI_Publication_PortType publishing = getPublishingStub();
    System.out.print("Save in progress ...");
    BusinessDetail businessDetail = publishing.save_business(save);
    System.out.println(" done");
    return businessDetail;
}
```

The saved businessEntity is displayed by the `printBusinessDetail()` method. One interesting aspect of the BEA AquaLogic Service Registry client API is that each UDDIObject contains the `toXML()`, which returns a human-readable formatted listing of the XML representation.

```
public static void printBusinessDetail(BusinessDetail businessDetail) {
    System.out.println();
    BusinessEntityArrayList businessEntityArrayList = businessDetail.getBusinessEntityArrayL-
ist();
    int position = 1;
    for (Iterator iterator = businessEntityArrayList.iterator(); iterator.hasNext();) {
        BusinessEntity entity = (BusinessEntity) iterator.next();
        System.out.println("Business " + position + " : " + entity.getBusinessKey());
        System.out.println(entity.toXML());
        System.out.println();
        System.out.println("*********************************************************");
        position++;
    }
}
```

Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\basic\publishing\v3
UNIX:	$REGISTRY_HOME/demos/basic/publishing/v3

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example to run the SaveBusiness demo, invoke

Windows:	run.bat SaveBusiness
UNIX:	./run.sh SaveBusiness

The output of this demo will resemble the following:

```
***************************************************************************
***    BEA AquaLogic Service Registry Demo - SaveBusiness   ***
***************************************************************************

Saving business entity where
Enter (optional) businessKey []: uddi:systinet.com:demo:marketing
Enter count of names [1]: 1
Enter language code []:
Enter name in language  [Marketing]:
Enter description [Saved by SaveBusiness demo]: Marketing department

Using Security at url http://localhost:8080/uddi/security .. done
Logging in .. done
```

```
businessKey = uddi:systinet.com:demo:marketing
lang = null, name = Marketing
description = Marketing department
Using Publishing at url http://localhost:8080/uddi/publishing .. done
Save in progress ... done

Business 1 : uddi:systinet.com:demo:marketing

<businessEntity businessKey="uddi:systinet.com:demo:marketing" xmlns="urn:uddi-org:api_v3">

  <name>Marketing</name>
  <description>Marketing department</description>
</businessEntity>

*******************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

2. Advanced Demos

Advanced demos section includes the following demos:

- Inquiry Range Queries demo - The BEA AquaLogic Service Registry Range queries demos set demonstrates, how to use BEA AquaLogic Service Registry inquiry enhancement - Range Queries. BEA AquaLogic Service Registry range queries functionality allows you to search UDDI entities with the ability to use comparative operators (>, <) for matching keyValues in keyedReferences.
- Custody demos - The BEA AquaLogic Service Registry Custody demo covers the custody transfer aspects of the UDDI API specification. You will learn how to generate a custody transfer token and transfer the ownership of selected structures to another user.
- Subscription demos - The BEA AquaLogic Service Registry advanced subscription demos cover the subscription aspects of the UDDI Version 3 Specification. They teach how to use the BEA AquaLogic Service Registry client API to create new subscriptions, get lists of subscriptions, get subscription results, and delete subscriptions.
- Validation demos - The valueset validation API provides methods to validate values used in the keyedReferences of checked taxonomies. The checks might range from very simple (check value against list of available values as in the InternalValidation service), to complex, such as performing contextual checks.
- Taxonomy demos - The Taxonomy API is used to manage and query taxonomies in the BEA AquaLogic Service Registry. These demos cover all API methods, so you can learn how to download, upload, save, delete, get and find taxonomies. In addition, you can manage individual values in internally checked taxonomies using the Category API.

2.1. Advanced Inquiry - Range Queries

The BEA AquaLogic Service Registry Range queries demos set demonstrates, how to use BEA AquaLogic Service Registry inquiry enhancement - Range Queries. BEA AquaLogic Service Registry range queries functionality allows you to search UDDI entities with the ability to use comparative operators (>, <) for matching keyValues in keyedReferences.

The demos set includes the following demo:

- `FindBusiness`

2.1.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Advanced Inquiry` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\advanced\inquiry\env.properties
UNIX:	$REGISTRY_HOME/demos/advanced/inquiry/env.properties

Table 7. Properties Used in Demos

Name	Default Value	Description
uddi.demos.result.max_rows	5	limit of data returned from registry
uddi.demos.url.inquiryExt	`http://localhost:8080/uddi/inquiryExt`	the extended inquiry web service port URL

2.1.2. Presentation and Functional Presentation

This section describes the programming pattern used in demos using the FindBusiness demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\advanced\inquiry\src\demo\uddi\rq\FindBusiness.java
UNIX:	$REGISTRY_HOME/demos/advanced/inquiry/src/demo/uddi/rq/FindBusiness.java

The helper method `createFindBusiness` creates a FindBusiness structure:

```
public Find_business createFindBusiness(String tModelKey, String keyValue,
                                        String operator, String quantifier)
        throws InvalidParameterException {
    System.out.println("tModelKey = " + tModelKey);
    System.out.println("keyValue = " + keyValue);
    System.out.println("operator = " + operator);
    System.out.println("quantifier = " + quantifier);
```

```
    Find_business find_business = new Find_business();
    QualifiedKeyedReference qualifiedKeyedReference = new QualifiedKeyedReference();
    qualifiedKeyedReference.setTModelKey(tModelKey);
    qualifiedKeyedReference.setKeyValue(keyValue);
   qualifiedKeyedReference.setFindQualifierArrayList(parseFindQualifiers(operator, quantifier));

   find_business.setCategoryBag(new CategoryBag(new KeyedReferenceArrayList(qualifiedKeyedRefer-
ence)));
    find_business.setMaxRows(new Integer(MAX_ROWS));

    return find_business;
}
```

The `findBusiness` method performs the searching operation:

```
public BusinessList findBusiness(Find_business find_business) throws UDDIException, SOAPException
 {
    System.out.print("Check structure validity .. ");
    try {
        find_business.check();
    } catch (InvalidParameterException e) {
        System.out.println("Failed!");
        throw new UDDIException(e);
    }
    System.out.println("OK");

    UDDI_Inquiry_PortType inquiry = getInquiryStub();
    System.out.print("Search in progress ..");
    BusinessList businessList = inquiry.find_business(find_business);
    System.out.println(" done");
    return businessList;
}
```

2.1.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry Advanced Inquiry demo set. Let us continue with our FindBusiness demo.

1. Be sure that the demo are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\advanced\inquiry
UNIX	$REGISTRY_HOME/demos/advanced/inquiry

3. Build demo using:

Windows:	UNIX:
`run.bat make`	`./run.sh make`

Note

When compiling demo on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

. This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the FindBusiness demo, invoke

Windows:	run.bat FindBusiness
UNIX:	./run.sh FindBusiness

The output of this demo will resemble the following:

```
**********************************************************************
***             BEA AquaLogic Service Registry Demo - FindBusiness             ***
**********************************************************************

Searching for businesses by category where keyedReference
Enter tModelKey [uddi:systinet.com:demo:location:floor]:
Enter keyValue [1]: 3
Enter operator (=,<,>,<=,>=,<>) [=]:>
Enter quantifier (exists,notExists) [exists]:
tModelKey = uddi:systinet.com:demo:location:floor
keyValue = 3
operator = >
quantifier = exists
Check structure validity .. OK
Using Inquiry at url http://van.in.idoox.com:8080/uddi/inquiryExt .. done
Search in progress .. done

Displaying 1 of 1, starting at position 1
Business 1 : uddi:systinet.com:demo:it
<businessInfoExt businessKey="uddi:systinet.com:demo:it" xmlns="http://systin-
et.com/uddi/api/v3/ext/5.0">
  <name xmlns="urn:uddi-org:api_v3">IT</name>
  <description xmlns="urn:uddi-org:api_v3">IT department</description>
  <serviceInfos xmlns="urn:uddi-org:api_v3">
    <serviceInfoExt serviceKey="uddi:systinet.com:demo:it:support" businessKey="uddi:sys-
tinet.com:demo:it" xmlns="http://systinet.com/uddi/api/v3/ext/5.0">
      <name xmlns="urn:uddi-org:api_v3">Support</name>
      <description xmlns="urn:uddi-org:api_v3">Telephone support</description>
      <bindingTemplates xmlns="urn:uddi-org:api_v3">
```

```
        <bindingTemplate bindingKey="uddi:b77eb8f0-86ce-11d8-ba05-123456789012" ser-
viceKey="uddi:systinet.com:demo:it:support">
          <description>IT related issues shall be reported there</description>
          <accessPoint useType="endPoint">tel:+1-123-456-7890</accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="uddi:uddi.org:transport:telephone"/>
          </tModelInstanceDetails>
        </bindingTemplate>
      </bindingTemplates>
    </serviceInfoExt>
    <serviceInfoExt serviceKey="uddi:systinet.com:demo:hr:employeesList" business-
Key="uddi:systinet.com:demo:hr" xmlns="http://systinet.com/uddi/api/v3/ext/5.0">
      <name xmlns="urn:uddi-org:api_v3">EmployeeList</name>
     <description xmlns="urn:uddi-org:api_v3">wsdl:type representing service</description>

      <bindingTemplates xmlns="urn:uddi-org:api_v3">
        <bindingTemplate bindingKey="uddi:5c546520-78b8-11d8-bec4-123456789012" ser-
viceKey="uddi:systinet.com:demo:hr:employeesList">
          <description>wsdl:type representing port</description>
         <accessPoint useType="http://schemas.xmlsoap.org/soap/http">urn:unknown-location-
uri</accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo tModelKey="uddi:systinet.com:demo:employeeList:binding">
              <instanceDetails>
                <instanceParms>EmployeeList</instanceParms>
              </instanceDetails>
            </tModelInstanceInfo>
            <tModelInstanceInfo tModelKey="uddi:systinet.com:demo:employeeList:portType">
              <instanceDetails>
                <instanceParms>EmployeeList</instanceParms>
              </instanceDetails>
            </tModelInstanceInfo>
          </tModelInstanceDetails>
          <categoryBag>
            <keyedReference tModelKey="uddi:uddi.org:xml:namespace" key-
Name="uddi.org:xml:namespace" keyValue="http://systinet.com/wsdl/demo/uddi/services/"/>
            <keyedReference tModelKey="uddi:uddi.org:wsdl:types" key-
Name="uddi.org:wsdl:types" keyValue="port"/>
           <keyedReference tModelKey="uddi:uddi.org:xml:localName" keyName="uddi.org:xml:loc-
alName" keyValue="EmployeeList"/>
            <keyedReference tModelKey="uddi:systinet.com:taxonomy:endpoint:availability"
keyName="Available" keyValue="Available"/>
            <keyedReference tModelKey="uddi:systinet.com:taxonomy:endpoint:status" key-
Name="Operational" keyValue="Operational"/>
          </categoryBag>
        </bindingTemplate>
      </bindingTemplates>
    </serviceInfoExt>
  </serviceInfos>
  <contactInfos>
    <contactInfo useType="Technical support">
      <personName xmlns="urn:uddi-org:api_v3">John Demo</personName>
    </contactInfo>
```

```
  </contactInfos>
</businessInfoExt>

*****************************************************
```

2.2. Custody

The BEA AquaLogic Service Registry demo is used to demonstrate the registry's application programming interface's capabilities and to demonstrate how to use this API.

The BEA AquaLogic Service Registry Custody demo covers the custody transfer aspects of the UDDI Version 3.01 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3].. You will learn how to generate a custody transfer token and transfer the ownership of selected structure to another user.

There is a single demo within this package - CustodyDemo. It demonstrates how to generate a transfer token for a selected UDDI key and how to use it to transfer the custody of the structure identified by the UDDI key to another user.

2.2.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is,, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Custody` demo are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\advanced\custody\env.properties
UNIX:	$REGISTRY_HOME/demos/advanced/custody/env.properties

Table 8. Properties used in demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.user.jane.name	demo_jane	second user's name
uddi.demos.user.jane.password	demo_jane	second user's password
uddi.demos.url.custody	`http://localhost:8080/uddi/custody`	the custody Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

2.2.2. Presentation and Functional Presentation

This section describes programming pattern of the Custody demo. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\advanced\custody\src\demo\uddi\custody\CustodyDemo.java
UNIX:	$REGISTRY_HOME/demos/advanced/custody/src/demo/uddi/custody/CustodyDemo.java

To make the demo easier to use, it contains two use cases. The first use case shows the owner of a UDDI structure who wants to transfer it to another user. The second use case is the second user transferring the same structure to his own custody. Let us start with first use case.

We must gather user input first. It is necessary to read user credentials and the key of the structure owned by the user. If you use default values, this means that the user demo_john is transferring custody of the `systinet.com:departmentID` tModel to user demo_jane. The user logs in and generates a transfer token for the given UDDI key. The transfer token contains information about the registry, expiration time, and secret opaqueToken. Any user who knows these data, can transfer the structure(s) covered by the transferToken.

```
String user = UserInput.readString("Enter first user name",
            DemoProperties.getProperty(USER_JOHN_NAME));
String password = UserInput.readString("Enter password",
                   DemoProperties.getProperty(USER_JOHN_PASSWORD));
String uddiKey = UserInput.readString("Enter UDDI key",
                                      "uddi:systinet.com:demo:departmentID");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Get_transferToken get = createGetTransferToken(uddiKey, authInfo);
TransferToken token = getTransferToken(get);
printTransferToken(token);
discardAuthInfo(authInfo, security);
```

The helper method `getCustodyStub()` returns the UDDI Custody stub of the Web service listening at the URL specified by the `URL_CUSTODY` property.

```
public static UDDI_CustodyTransfer_PortType getCustodyStub() throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.custody
    String url = DemoProperties.getProperty(URL_CUSTODY, "http://localhost:8080/uddi/custody");

    System.out.print("Using Custody at url " + url + " ..");
    UDDI_CustodyTransfer_PortType custody = UDDICustodyStub.getInstance(url);
    System.out.println(" done");
```

```
    return custody;
}
```

The `createGetTransferToken()` method is used to create the Get_transferToken object, which encapsulates the parameters of this UDDI call. In this example we set authInfo and a single key for the UDDI structure to be transferred int the custody of the second user.

```
public static Get_transferToken createGetTransferToken(String uddiKey, String authInfo)
  throws InvalidParameterException {
    System.out.println("uddiKey = " + uddiKey);
    Get_transferToken get = new Get_transferToken();
    get.addKey(uddiKey);
    get.setAuthInfo(authInfo);
    return get;
}
```

The next step is to invoke the `get_transferToken` UDDI call and get the result, which is a TransferToken.

```
public static TransferToken getTransferToken(Get_transferToken get)
  throws UDDIException, SOAPException {
    UDDI_CustodyTransfer_PortType custody = getCustodyStub();
    System.out.print("Get in progress ...");
    TransferToken token = custody.get_transferToken(get);
    System.out.println(" done");
    return token;
}
```

At this point the first user, John Demo, has generated a transfer token. He can discard it or send it to the second user Jane Demo, so she can transfer the entities to her custody. The transfer token must be kept secret, so plain text transports such as unencrypted emails are not suitable for this purpose. Let us suppose that Jane Demo has received the transfer token already. She logs in, creates a Transfer_entities object and invokes the UDDI call `transfer_entities`.

```
user = UserInput.readString("Enter second user name",
                                DemoProperties.getProperty(USER_JANE_NAME));
password = UserInput.readString("Enter password", DemoProperties.getProperty(USER_JANE_PASS-
WORD));
System.out.println();

authInfo = getAuthInfo(user, password, security);
Transfer_entities transfer = createTransferEntities(uddiKey, token, authInfo);
transferEntities(transfer);
discardAuthInfo(authInfo, security);
```

The `createTransferEntities()` method is used to create Transfer_entities object, which encapsulates parameters of same name UDDI call. In this example we set Jane's authInfo, UDDI key to be transferred, and the TransferToken generated by John.

```
public static Transfer_entities createTransferEntities(String uddiKey,
                                                        TransferToken token, String authInfo)

  throws InvalidParameterException {
    Transfer_entities transfer = new Transfer_entities();
    transfer.addKey(uddiKey);
    transfer.setTransferToken(token);
```

```
    transfer.setAuthInfo(authInfo);
    return transfer;
}
```

The final step is to make the `transfer_entities` UDDI call. When it successfully returns, the second user (Jane) is the happy owner of the UDDI structure `systinet.com:demo:departmentID`.

```
public static void transferEntities(Transfer_entities transfer)
  throws UDDIException, SOAPException {
    UDDI_CustodyTransfer_PortType custody = getCustodyStub();
    System.out.print("Transfer in progress ...");
    custody.transfer_entities(transfer);
    System.out.println(" done");
}
```

2.2.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Custody demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\advanced\custody
UNIX:	$REGISTRY_HOME/demos/advanced/custody

3. Build demo using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available commands, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The demo can be executed via the **run** command, using the name of the demo as a parameter. To run the Custody demo, invoke

Windows:	run.bat CustodyDemo
UNIX:	./run.sh CustodyDemo

The output of this demo will resemble the following:

```
Running CustodyDemo demo...
***********************************************************************
***                BEA AquaLogic Service Registry Demo - CustodyDemo                ***
***********************************************************************

Getting transfer token where
Enter first user name [demo_john]:
Enter password [demo_john]:
Enter UDDI key [uddi:systinet.org:demo:departmentID]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
uddiKey = uddi:systinet.org:demo:departmentID
Using Custody at url https://mycomp.com:8443/uddi/custody .. done
Get in progress ... done

TransferToken
<transferToken xmlns="urn:uddi-org:custody_v3">
<nodeID xmlns="urn:uddi-org:api_v3">Systinet</nodeID>
<expirationTime>2004-05-17T12:32:51.236+02:00</expirationTime>
<opaqueToken>ZmZmZmZmZmZlMDVmZGEzNg==</opaqueToken>
</transferToken>

Logging out .. done

Transfering custody where
Enter second user name [demo_jane]:
Enter password [demo_jane]:

Logging in .. done
Using Custody at url https://mycomp.com:8443/uddi/custody .. done
Transfer in progress ... done
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

2.3. Subscription

The BEA AquaLogic Service Registry advanced subscription demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and shows how to use the Subscription API to perform subscription calls to the registry.

The BEA AquaLogic Service Registry advanced subscription demos cover the subscription aspects of the UDDI Version 3 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]. They teach how to use the BEA AquaLogic Service Registry client API to create new subscriptions, get lists of subscriptions, get subscription results, and delete subscriptions.

The BEA AquaLogic Service Registry basic publishing demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API:

SaveSubscription Demonstrates how to construct and fill the `Save_subscription` object, get a Subscription stub for the UDDI registry, and perform the `save_subscription` call.

GetSubscriptions Demonstrates how to construct and fill the `Get_subscriptions` object, get a Subscription stub for the UDDI registry, and perform the `get_subscriptions` call.

GetSubscriptionResults Demonstrates how to construct and fill the `Get_subscriptionResults` object, get a Subscription stub for the UDDI registry, and perform the `get_subscriptionResults` call.

DeleteSubscription Demonstrates how to construct and fill the `Delete_subscription` object, get a Subscription stub for the UDDI registry, and perform the `delete_subscription` call.

2.3.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Subscription` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\advanced\subscription\env.properties
UNIX:	$REGISTRY_HOME/demos/advanced/subscription/env.properties

Table 9. Properties used in demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.subscription	`http://localhost:8080/uddi/subscription`	the subscription web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security web service port URL

2.3.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the GetSubscriptionResults demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\subscription\src\demo\uddi\subscription\GetSubscriptionResults.java

UNIX:	$REGISTRY_HOME/demos/basic/subscription/src/demo/uddi/subscription/GetSubscriptionResults.java

Let us start with a description of `main` method. The first part is used to configure the demo by the user. Then it logs the user into the UDDI registry, creates a `Get_subscriptionResults` object holding the parameters of the request. This object is transformed in the next step into the SOAP UDDI call `get_subscriptionResults`. Its results are then displayed and the user is logged off from the UDDI registry.

```
String user = UserInput.readString("Enter user name",
                                    DemoProperties.getProperty(USER_JOHN_NAME));
String password = UserInput.readString("Enter password",
                                       DemoProperties.getProperty(USER_JOHN_PASSWORD));
String key = UserInput.readString("Enter subscription key", "");
int shift = UserInput.readInt("Enter start of coverage period in minutes", 60);
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Get_subscriptionResults get = createGetSubscriptionResults(key, shift, authInfo);
SubscriptionResultsList result = getSubscriptionResults(get);
printSubscriptionResults(result);
discardAuthInfo(authInfo, security);
```

The method `createGetSubscriptionResults` takes subscriptionKey as a parameter that identifies the subscription in the UDDI registry, coveragePeriod, and authInfo of the user. The CoveragePeriod is used to identify the time period for which the user is interested in changes matched by the selected Subscription.

```
public static Get_subscriptionResults createGetSubscriptionResults(String subscriptionKey,
  int coveragePeriod, String authInfo) throws InvalidParameterException {
    Get_subscriptionResults getSubscriptionResults = new Get_subscriptionResults();
    getSubscriptionResults.setSubscriptionKey(subscriptionKey);

    // calculate coverage period
    long coveragePeriodShiftInMs = coveragePeriod * 60 * 1000;
    long endPoint = System.currentTimeMillis();
    long startPoint = endPoint - coveragePeriodShiftInMs;
    getSubscriptionResults.setCoveragePeriod(new CoveragePeriod(new Date(startPoint),
                                                                 new Date(endPoint)));

    getSubscriptionResults.setAuthInfo(authInfo);

    return getSubscriptionResults;
}
```

The helper method, `getSubscriptionStub()`, returns the UDDI Subscription stub of the web service listening at the URL specified by the `URL_SUBSCRIPTION` property.

```
public static UDDI_Subscription_PortType getSubscriptionStub() throws SOAPException {
     String url = DemoProperties.getProperty(URL_SUBSCRIPTION,
                                             "http://localhost:8080/uddi/subscription");
     System.out.print("Using Subscription at url " + url + " ..");
     UDDI_Subscription_PortType subscriptionStub = UDDISubscriptionStub.getInstance(url);
     System.out.println(" done");
```

```
    return subscriptionStub;
}
```

The UDDI API call `get_subscriptionResults` is performed in the method `getSubscriptionResults()`:

```
public static SubscriptionResultsList getSubscriptionResults(Get_subscriptionResults save)
  throws UDDIException, SOAPException {
    UDDI_Subscription_PortType subscriptionStub = getSubscriptionStub();
    System.out.print("Get in progress ...");
    SubscriptionResultsList result = subscriptionStub.get_subscriptionResults(save);
    System.out.println(" done");
    return result;
}
```

2.3.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Advanced Subscription demo set. Let us continue with our GetSubscriptionResults demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\advanced\subscription
UNIX:	$REGISTRY_HOME/demos/advanced/subscription

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get a list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** with the name of the demo as parameter. For example, to run the GetSubscriptionResults demo, invoke

Windows:	run.bat GetSubscriptionResults
UNIX:	./run.sh GetSubscriptionResults

6. The BEA AquaLogic Service Registry Subscription demos show a complete use case for the Subscription API. The SaveSubscription demo creates a new subscription for the user John Demo. This subscription monitors changes to the business entity named `Marketing`.

```
Running SaveSubscription demo...
************************************************************************
***             BEA AquaLogic Service Registry Demo - SaveSubscriptionDemo           ***
************************************************************************

Saving subscription where
Enter user name [demo_john]:
Enter password [demo_john]:
Enter business name to watch [Marketing]:
Enter subscription validity in days [2]:
Enter limit of subscription results [5]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
businessName = Marketing
limit = 5
valid = 2
Using Subscription at url https://mycomp.com:8443/uddi/subscription .. done
Save in progress ... done

Subscription 1 : uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
<subscription brief="false" xmlns="urn:uddi-org:sub_v3">
    <subscriptionKey>uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd</subscriptionKey>
    <subscriptionFilter>
        <find_business xmlns="urn:uddi-org:api_v3">
            <name>Marketing</name>
        </find_business>
    </subscriptionFilter>
    <maxEntities>5</maxEntities>
    <expiresAfter>2004-05-14T11:28:30.721+02:00</expiresAfter>
</subscription>

*******************************************************
Logging out .. done
```

If you want to list your available subscriptions, run the `GetSubscriptions` demo:

```
Finding subscriptions where
Enter user name [demo_john]:
Enter password [demo_john]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
Using Subscription at url https://mycomp.com:8443/uddi/subscription .. done
Get in progress ... done

Subscription 1 : uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
<subscription brief="false" xmlns="urn:uddi-org:sub_v3">
    <subscriptionKey>uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd</subscriptionKey>
    <subscriptionFilter>
```

```
        <find_business xmlns="urn:uddi-org:api_v3">
            <name>Marketing</name>
        </find_business>
    </subscriptionFilter>
    <maxEntities>5</maxEntities>
    <expiresAfter>2004-05-14T11:28:30.721+02:00</expiresAfter>
</subscription>

*******************************************************
Logging out .. done
```

Now we need to generate some traffic on UDDI registry, that matches the subscription filter, that we have defined. You can use SaveBusiness demo from BEA AquaLogic Service Registry Basic Publishing demos to save business entity named Marketing.

```
Running SaveBusiness demo...
***********************************************************************
***             BEA AquaLogic Service Registry Demo - SaveBusinessDemo              ***
***********************************************************************


Saving business entity where
Enter (optional) businessKey []:
Enter count of names [1]:
Enter language code []:
Enter name in language  [Marketing]:
Enter description [Saved by SaveBusiness demo]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
businessKey =
lang = null, name = Marketing
description = Saved by SaveBusiness demo
Using Publishing at url https://mycomp.com:8443/uddi/publishing .. done
Save in progress ... done

Business 1 : uddi:8097cc00-a578-11d8-91cd-5c1d367091cd
<businessEntity businessKey="uddi:8097cc00-a578-11d8-91cd-5c1d367091cd" xmlns="urn:uddi-
org:api_v3">
    <name> Marketing</name>
    <description> Saved by SaveBusiness demo</description>
</businessEntity>
```

Then we want to get the results of the subscription. It is necessary to specify correct subscription key and sufficient coverage period.

```
Running GetSubscriptionResults demo...
***********************************************************************
***            BEA AquaLogic Service Registry Demo - GetSubscriptionResultsDemo      ***
***********************************************************************


Finding subscription results where
Enter user name [demo_john]:
Enter password [demo_john]:
Enter subscription key []: uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
```

```
Enter start of coverage period in minutes [60]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
Using Subscription at url https://mycomp.com:8443/uddi/subscription .. done
Get in progress ... done
Subscription uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
Coverage period=Fri May 14 08:30:28 CEST 2004 - Fri May 14 09:30:28 CEST 2004

Subscription results:
<subscriptionResultsList xmlns="urn:uddi-org:sub_v3">
    <chunkToken>0</chunkToken>
    <coveragePeriod>
        < startPoint>2004-05-14T08:30:28.565+02:00</startPoint>
        < endPoint>2004-05-14T09:30:28.824+02:00</endPoint>
    </coveragePeriod>
    < subscription brief="false">
        < subscriptionKey> uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd</subscriptionKey>
        < subscriptionFilter>
            < find_business xmlns="urn:uddi-org:api_v3">
                 < name> Marketing</name>
            </find_business>
        </subscriptionFilter>
        < maxEntities>5</maxEntities>
        < expiresAfter>2004-05-14T11:28:30.721+02:00</expiresAfter>
    </subscription>
    < businessList>
        < businessInfos>
            < businessInfo businessKey="uddi:8097cc00-a578-11d8-91cd-5c1d367091cd">
                 < name> Marketing</name>
                 < description> Saved by SaveBusiness demo</description>
            </businessInfo>
        </businessInfos>
    </businessList>
</subscriptionResultsList>

*******************************************************
```

If we do not need the subscription anymore, we can delete it with DeleteSubscription demo.

```
*************************************************************************
***             BEA AquaLogic Service Registry Demo - DeleteSubscriptionDemo         ***
*************************************************************************

Deleting subscription where
Enter subscription key []: uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
subscriptionKey = uddi:4f0d7450-a578-11d8-91cd-5c1d367091cd
Using Subscription at url https://mycomp.com:8443/uddi/subscription .. done
Delete in progress ... done
Logging out .. done
```

2.4. Validation

The BEA AquaLogic Service Registry Validation demo shows how to implement, deploy, and use a custom valueset validation service.

The valueset validation API provides methods to validate values used in keyedReferences of checked taxonomies. The checks might range from very simple (check value against list of available values like in InternalValidation service) to complex, which performs contextual checks.

There are two classes and one xml file to import taxonomy, that are used by the Validation demo.

ISBNValidation Valueset validation interface implementation. It checks keyValues from keyedReferences in all structures. The keyValue must be in ISBN format, otherwise E_invalidValue UDDI exception is thrown to deny the save operation.

isbn.xml Taxonomy description used to import checked categorization demo:ISBN into the BEA AquaLogic Service Registry.

ValidationDemo Demonstrates how to save a tModel with the keyedReference, that uses demo:ISBN categorization checked by ISBNValidation.

2.4.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Validation` demo is loaded from the file:

Windows:	%REGISTRY_HOME%\demos\advanced\validation\env.properties
UNIX:	$REGISTRY_HOME/demos/advanced/validation/env.properties

Table 10. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.publishing	`http://localhost:8080/uddi/publishing`	the publishing Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

2.4.2. Presentation and Functional Presentation

This section describes programming pattern used in ISBNValidation class. You can find its source code in the file

Windows:	%REGISTRY_HOME%\demos\advanced\validation\src\demo\uddi\validation\ISBNValidation.java
UNIX:	$REGISTRY_HOME/demos/advanced/validation/src/demo/uddi/validation/ISBNValidation.java

The BEA AquaLogic Service Registry simplifies the development of Valueset validation services. It intelligently performs some checks automatically based on the properties of the taxonomy (content of categoryBag), so you as developer may concentrate on logic of your validation service. For example it ensures, that categorization tModelKey is not used in identifierBag or that it is used only in UDDI structures, for which its compatibility was declared.

Let's start with description of `validate_values` method. It serves as starting point to the validation service. The Validate_values object contains at least one tModel, businessEntity, businessService, bindingTemplate or publisherAsertion, which contains reference to the taxonomy validated by this web service. If the validation service is shared between several taxonomies, UDDI structures, which use them, are grouped in single validate_values call.

When the method `validate_values` finds the structure type to be validated, it calls `validate_values` on the list of UDDI structures, which iterates over each element in the list and call `validate` method on single structure. If there is at least one error in dispositionReport, UDDI exception is thrown to deny the save operation.

```
public DispositionReport validate_values(Validate_values body) throws UDDIException {
    DispositionReport report = new DispositionReport();

    if (body.getBusinessEntityArrayList() != null)
        validate_values(body.getBusinessEntityArrayList(), report);

    else if (body.getBusinessServiceArrayList() != null)
        validate_values(body.getBusinessServiceArrayList(), report);

    else if (body.getTModelArrayList() != null)
        validate_values(body.getTModelArrayList(), report);

    else if (body.getPublisherAssertionArrayList() != null)
        validate_values(body.getPublisherAssertionArrayList(), report);

    else if (body.getBindingTemplateArrayList() != null)
        validate_values(body.getBindingTemplateArrayList(), report);

    ResultArrayList results = report.getResultArrayList();
    if (results == null || results.size() == 0)
        return DispositionReport.DISPOSITION_REPORT_SUCCESS;
```

```
    throw new UDDIException(report);
}
```

This method than validates all keyedReferences and if the structure contains children (for example businessServices in businessEntity), it recursively validates the too. For demo:ISBN categorization the check of identifierBag is useless, because the BEA AquaLogic Service Registry would already detect it as error and stop the execution of save operation.

```
private void validate(TModel tModel, DispositionReport report) throws UDDIException {
    CategoryBag categoryBag = tModel.getCategoryBag();
    IdentifierBag identifierBag = tModel.getIdentifierBag();
    KeyedReferenceArrayList keyedReferences;

    if (categoryBag != null) {
        keyedReferences = categoryBag.getKeyedReferenceArrayList();
        if (keyedReferences != null) {
            validate(keyedReferences, report);
        }

        validateKeyedReferenceGroups(categoryBag.getKeyedReferenceGroupArrayList(), report);
    }

    if (identifierBag != null) {
        keyedReferences = identifierBag.getKeyedReferenceArrayList();
        if (keyedReferences != null) {
            validate(keyedReferences, report);
        }
    }
}
```

The method `validate` iterates over all keyedReferences and if they reference demo:ISBN taxonomy, than it checks the keyValue, if it is in valid ISBN format. If not, it adds error report to dispositionReport.

```
private void validate(KeyedReferenceArrayList keyedReferenceArrayList, DispositionReport report)

  throws UDDIException {
    for (Iterator iter = keyedReferenceArrayList.iterator(); iter.hasNext();) {
        KeyedReference keyedReference = (KeyedReference) iter.next();
        if (TMODEL_KEY.equalsIgnoreCase(keyedReference.getTModelKey())) {
            if (!checkISBN(keyedReference.getKeyValue())) {
                String message = "KeyValue is not valid ISBN number in " + keyedRefer-
ence.toXML();
                report.addResult(createResult(UDDIErrorCodes.E_INVALID_VALUE, message));
            }
        }
    }
}
```

The implementation of ISBNValidation web service is not optimal. It scans all UDDI structures and containers of keyedReferences, even if the BEA AquaLogic Service Registry was configured to deny such usage. The optimal code would check only categoryBag in tModels.

2.4.3. Building and Running Demos

This section shows, how to build, deploy and run the BEA AquaLogic Service Registry Advanced Validation demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\advanced\validation
UNIX:	$REGISTRY_HOME/demos/advanced/validation

3. Build all classes using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. Copy the file ISBNValidation.class to REGISTRY_HOME/app/uddi/services/Wasp-inf/classes

Windows:	cd %REGISTRY_HOME%\demos\advanced\validation\build
	xcopy classes %REGISTRY_HOME%\app\uddi\services\Wasp-inf\classes /S
UNIX:	cd $REGISTRY_HOME/demos/advanced/validation/build
	cp -r classes $REGISTRY_HOME/app/uddi/services/Wasp-inf

5. Now use Advanced Taxonomy demo UploadTaxonomy to upload the file isbn.xml located in data subdirectory of Validation demo directory. For more information, how to do it, read Taxonomy demo documentation.

6. When the demo:ISBN taxonomy has been uploaded and `ISBNValidation.class` copied, you must shutdown the BEA AquaLogic Service Registry, delete the REGISTRY_HOME/work directory, and restart the BEA AquaLogic Service Registry.

7. The ValidationDemo can be executed via command run with

Windows:	run.bat ValidationDemo
UNIX:	./run.sh ValidationDemo

The output of this demo will resemble the following:

8. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

2.5. Taxonomy

The BEA AquaLogic Service Registry Taxonomy demos demonstrates the BEA AquaLogic Service Registry's Taxonomy capabilities and show how to use this API.

The Taxonomy is used to manage and query taxonomies in the BEA AquaLogic Service Registry. These demos cover all API methods, so you can learn how to download, upload, save, delete, get and find taxonomies. In addition, you can manage individual values in internally checked taxonomies using the Category API.

The BEA AquaLogic Service Registry contains the following demos to assist you in learning the BEA AquaLogic Service Registry Taxonomy and Category APIs.

SaveTaxonomy Demonstrates how to save unchecked taxonomy, which can be used in businessEntities and tModels.

DeleteTaxonomy Demonstrates how to deletes selected taxonomy. If the taxonomy was checked, associated binding template is automatically removed too.

UploadTaxonomy Demonstrates how to upload the file containg taxonomy. This API call is usefull, when you need to process really large taxonomies, because it operates on stream of data.

DownloadTaxonomy Demonstrates how to download selected taxonomy. Again this method is stream oriented.

GetTaxonomy Demonstrates how to get details of selected taxonomy.

FindTaxonomy Demonstrates how to search for taxonomies based on given criteria.

AddCategory Demonstrates how to add new category (keyedReference value) to existing internal taxonomy.

DeleteCategory Demonstrates how to delete the category in existing internal taxonomy.

SetCategory Demonstrates how to update the category in existing internal taxonomy.

MoveCategory Demonstrates how to change the parent of the category in existing internal taxonomy.

GetCategory Demonstrates how to get the category of the internal taxonomy.

GetRootCategory Demonstrates how to get list of the top-level categories of the internal taxonomy.

GetRootPath Demonstrates how to get list of parents of selected category, from the top-level category to the selected one.

FindCategory Demonstrates how to get list of categories, that match some criterias.

2.5.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Taxonomy` demo is loaded from the file:

Windows:	%REGISTRY_HOME%\demos\advanced\taxonomy\env.properties
UNIX:	$REGISTRY_HOME/demos/advanced/taxonomy/env.properties

Table 11. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.taxonomy	`http://localhost:8080/uddi/taxonomy`	the taxonomy Web service port URL
uddi.demos.url.category	`http://localhost:8080/uddi/category`	the category Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

2.5.2. Presentation and Functional Presentation

This section describes programming pattern used in all demos using the SaveTaxonomy demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\advanced\taxonomy\src\demo\uddi\taxonomy\SaveTaxonomy.java
UNIX:	$REGISTRY_HOME/demos/advanced/taxonomy/src/demo/uddi/taxonomy/SaveTaxonomy.java

The main method of this demo is straightforward. It gathers user's input, logs the user in the BEA AquaLogic Service Registry, creates an object of Save_taxonomy, sends it to UDDI registry over SOAP and displays the result.

```
String user = UserInput.readString("Enter user name", "admin");
String password = UserInput.readString("Enter password", "changeit");
String name = UserInput.readString("Enter name", "Demo identifier");
String description = UserInput.readString("Enter description", "Saved by SaveTaxonomy demo");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Save_taxonomy save = createSaveTaxonomy(name, description, authInfo);
TaxonomyDetail result = saveTaxonomy(save);
printTaxonomyDetail(result);
discardAuthInfo(authInfo, security);
```

When saving taxonomy, you must first create a tModel, that will represent it. You can set your publisher assigned tModelKey and other properties. The only mandatory property is name. You don't need to specify taxonomy related keyedReferences in categoryBag, they shall be set in Taxonomy.

The Categorization is used to define usage of the taxonomy. Valid values are identifier, categorization, categorizationGroup and relationship. The compatibility marks tModel with information, in which UDDI structures it can be used.

This example creates an unchecked identifier, that can be used only in categoryBags of business entities and tModels.

```
public static Save_taxonomy createSaveTaxonomy(String name, String description, String authInfo)

  throws InvalidParameterException {
    System.out.println("name = " + name);
    System.out.println("description = " + description);

    TModel tModel = new TModel();
    tModel.setName(new Name(name));
    tModel.addDescription(new Description(description));

    Taxonomy taxonomy = new Taxonomy(tModel);
    taxonomy.setCheck(Boolean.FALSE);
    taxonomy.addCategorization(Categorization.identifier);
    taxonomy.addCompatibility(Compatibility.businessEntity);
    taxonomy.addCompatibility(Compatibility.tModel);

    Save_taxonomy save = new Save_taxonomy();
    save.addTaxonomy(taxonomy);
    save.setAuthInfo(authInfo);

    return save;
}
```

The helper method `getTaxonomyStub()` returns the Taxonomy stub of the Web service listening at the URL specified by the `URL_TAXONOMY` property.

```
public static TaxonomyApi getTaxonomyStub() throws SOAPException {
    String url = DemoProperties.getProperty(URL_TAXONOMY, "http://localhost:8080/uddi/taxonomy");

    System.out.print("Using Taxonomy at url " + url + " ..");
    TaxonomyApi taxonomy = TaxonomyStub.getInstance(url);
    System.out.println(" done");
    return taxonomy;
}
```

The Taxonomy API call `save_taxonomy` is performed in the method `saveTaxonomy()`.

```
public static TaxonomyDetail saveTaxonomy(Save_taxonomy save)
  throws UDDIException, SOAPException {
    TaxonomyApi taxonomy = getTaxonomyStub();
    System.out.print("Save in progress ...");
    TaxonomyDetail taxonomyDetail = taxonomy.save_taxonomy(save);
    System.out.println(" done");
    return taxonomyDetail;
}
```

The returned TaxonomyDetail object is displayed in `printTaxonomyDetail` method.

```
public static void printTaxonomyDetail(TaxonomyDetail taxonomyDetail) {
    System.out.println();

    TaxonomyArrayList taxonomyArrayList = taxonomyDetail.getTaxonomyArrayList();
```

```
    int position = 1;
    for (Iterator iterator = taxonomyArrayList.iterator(); iterator.hasNext();) {
        Taxonomy taxonomy = (Taxonomy) iterator.next();
       System.out.println("Taxonomy " + position + " : " + taxonomy.getTModel().getTModelKey());

        System.out.println(taxonomy.toXML());
        System.out.println();
        System.out.println("*******************************************************");
        position++;
    }
}
```

2.5.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry Advanced Taxonomy demo set. Let's continue with our SaveTaxonomy demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.
2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\advanced\taxonomy
UNIX:	$REGISTRY_HOME/demos/advanced/taxonomy

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via command run with name of demo as parameter. For example to run the SaveTaxonomy demo, invoke

Windows:	run.bat SaveTaxonomy
UNIX:	./run.sh SaveTaxonomy

The output of this demo will resemble the following:

```
Running SaveTaxonomy demo...
************************************************************************
***             BEA AquaLogic Service Registry Demo - SaveTaxonomyDemo              ***
************************************************************************

Saving taxonomy where
Enter user name [admin]:
Enter password [changeit]:
Enter name [Demo identifier]:
Enter description [Saved by SaveTaxonomy demo]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
name = Demo identifier
description = Saved by SaveTaxonomy demo
Using Taxonomy at url https://mycomp.com:8443/uddi/taxonomy .. done
Save in progress ... done

Taxonomy 1 : uddi:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd
<taxonomy check="false" xmlns="http://systinet.com/uddi/taxonomy/v3/5.0">
  <tModel tModelKey="uddi:5c1d5d80-a4d4-11d8-91cd-5c1d367091cd"
   xmlns="urn:uddi-org:api_v3">
    <name>Demo identifier</name>
    <description>Saved by SaveTaxonomy demo</description>
    <categoryBag>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types"
                      keyName="Identifier system" keyValue="identifier"/>
      <keyedReference tModelKey="uddi:systinet.com:taxonomy:compatibility"
                      keyName="Compatibility" keyValue="businessEntity"/>
      <keyedReference tModelKey="uddi:systinet.com:taxonomy:compatibility"
                      keyName="Compatibility" keyValue="tModel"/>
      <keyedReference tModelKey="uddi:uddi.org:categorization:types"
                      keyName="Unchecked value set" keyValue="unchecked"/>
    </categoryBag>
  </tModel>
  <compatibilityBag>
    <compatibility>businessEntity</compatibility>
    <compatibility>tModel</compatibility>
  </compatibilityBag>
  <categorizationBag>
    <categorization>identifier</categorization>
  </categorizationBag>
</taxonomy>

*******************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

3. Security Demos

Security Demos section includes the following demos:

- Account Demos - You will learn how to register new accounts (or update existing accounts), enable, get, find, and delete accounts.
- Group Demos - You will learn how to create or update, get, find and delete groups.
- Permission Demos - You will learn how to set and search permissions.
- ACL Demos - The Systinet ACL extension is used to grant or revoke rights to selected users or groups. You will learn how to create, save, delete, get and find ACLs.

3.1. Account

The BEA AquaLogic Service Registry Account Demos are used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to demonstrate how to use this API.

You will learn how to register new accounts (or update existing accounts), enable, get, find, and delete accounts.

The BEA AquaLogic Service Registry security account demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API:

SaveAccount Demonstrates how to construct and fill the `Save_account` object, get an Account stub for the UDDI registry, and perform the `save_account` call.

DeleteAccount Demonstrates how to construct and fill the `Delete_account` object, get an Account stub for the UDDI registry, and perform the `delete_account` call.

3.1.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is,, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Account` demo are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\security\account\env.properties
UNIX:	$REGISTRY_HOME/demos/security/account/env.properties

Table 12. Properties Used in Demos

Name	Default Value	Description
uddi.demos.url.account	`http://localhost:8080/uddi/account`	the account Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

3.1.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the SaveAccount demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\security\account\src\demo\uddi\account\SaveAccount.java
UNIX:	$REGISTRY_HOME/demos/security/account/src/demo/uddi/account/SaveAccount.java

The main method is divided into two parts. The first part serves to configure the demo by the user. It reads the credentials of the user who will run the demo. If you wish to save new user on a registry that supports public registration, then the demo may be modified to skip authentication. It then reads information about the new user to be saved (or about the user to be updated) including login name, password, name, and email address.

The second part contains the execution of the demo. It looks up the security stub and authenticates the user. It then creates a `Save_userAccount` object and sends it over SOAP to the UDDI registry as a `save_userAccount` operation. The returned `UserAccount` object is printed to the console and the authInfo is discarded.

```
String admin = UserInput.readString("Enter admin login","admin");
String admin_password = UserInput.readString("Enter admin password","changeit");
String login = UserInput.readString("Enter new user's login","demo_eric");
String password = UserInput.readString("Enter password","demo_eric");
String name = UserInput.readString("Enter full name","Eric Demo");
String email = UserInput.readString("Enter email","demo_eric@localhost");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(admin, admin_password, security);
Save_userAccount save = createSaveUserAccount(login, password, name, email, authInfo);
UserAccount userAccount = saveUserAccount(save);
printUserAccount(userAccount);
discardAuthInfo(authInfo, security);
```

The method `createSaveUserAccount` is used to create an object representing the `save_userAccount` operation. The authInfo is required under two circumstances: if the BEA AquaLogic Service Registry is configured not to allow public registration or if the account already exists.

```
public static Save_userAccount createSaveUserAccount(String login, String password,
  String name, String email, String authInfo) throws InvalidParameterException {
    System.out.println("login = " + login);
    System.out.println("password = " + password);
    System.out.println("name = " + name);
    System.out.println("email = " + email);

    UserAccount account = new UserAccount();
    account.setLoginName(login);
    account.setPassword(password);
    account.setFullName(name);
```

```
    account.setEmail(email);
    account.setLanguageCode("EN");

    Save_userAccount save = new Save_userAccount(account, authInfo);
    return save;
}
```

The helper method, `getAccountStub()`, returns the UDDI Account stub of the web service listening at the URL specified by the `URL_ACCOUNT` property.

```
public static AccountApi getAccountStub() throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.account
    String url = DemoProperties.getProperty(URL_ACCOUNT, "http://localhost:8080/uddi/account");

    System.out.print("Using Account at url " + url + " ..");
    AccountApi account = AccountStub.getInstance(url);
    System.out.println(" done");
    return account;
}
```

The BEA AquaLogic Service Registry API call `save_userAccount` is performed in the method `saveUserAccount`.

```
public static UserAccount saveUserAccount(Save_userAccount save) throws SOAPException, Accoun-
tException {
    AccountApi accountApi = getAccountStub();
    System.out.print("Save in progress ...");
    UserAccount userAccount = accountApi.save_userAccount(save);
    System.out.println(" done");
    return userAccount;
}
```

3.1.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Account demos.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.
2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\security\account
UNIX:	$REGISTRY_HOME/demos/security/account

3. Build demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available commands, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the SaveAccount demo, invoke

Windows:	run.bat SaveAccount
UNIX:	./run.sh SaveAccount

The output of this demo will resemble the following:

```
Running SaveAccount demo...
***************************************************************************
***    BEA AquaLogic Service Registry Demo - SaveAccount   ***
***************************************************************************

Saving user account where
Enter admin login [admin]:
Enter admin password [changeit]:
Enter new user's login [demo_eric]:
Enter password [demo_eric]:
Enter full name [Eric Demo]:
Enter email [demo_eric@localhost]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
login = demo_eric
password = demo_eric
name = Eric Demo
email = demo_eric@localhost
Using Account at url https://mycomp.com:8443/uddi/account .. done
Save in progress ... done

User account
<userAccount xmlns="http://systinet.com/uddi/account/5.0">
<loginName>demo_eric</loginName>
<password>GD70gCeNfkwBph1m2bgGxQ==</password>
<email>demo_eric@localhost</email>
<fullName>Eric Demo</fullName>
<languageCode>EN</languageCode>
<expiration>1970-01-01T02:00:00.000+02:00</expiration>
<external>false</external>
<blocked>false</blocked>
<businessesLimit>1</businessesLimit>
<servicesLimit>4</servicesLimit>
<bindingsLimit>2</bindingsLimit>
<tModelsLimit>100</tModelsLimit>
<assertionsLimit>10</assertionsLimit>
<subscriptionsLimit>0</subscriptionsLimit>
```

```
<lastLoginTime>2004-05-18T16:20:09.084+02:00</lastLoginTime>
</userAccount>

*****************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

3.2. Group

The BEA AquaLogic Service Registry Group demos are used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to demonstrate how to use this API.

You will learn how to create or update, get, find and delete groups.

The BEA AquaLogic Service Registry security group demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API:

Save Demonstrates how to construct and fill the `Save_group` object, get a Group stub for the UDDI registry, and perform the `save_group` call.

Delete Demonstrates how to construct and fill the `Delete_group` object, get a Group stub for the UDDI registry, and perform the `delete_group` call.

Get Demonstrates how to construct and fill the `Get_group` object, get a Group stub for the UDDI registry, and perform the `get_group` call.

Find Demonstrates how to construct and fill the `Find_group` object, get a Group stub for the UDDI registry, and perform the `find_group` call.

WhereIAm Demonstrates how to construct and fill the `Where_amI` object, get a Group stub for the UDDI registry, and perform the `where_amI` call.

3.2.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is,, at the local level), edit `env.prop-`

erties. This file is located in the same directory as the file run.sh (run.bat). Local level properties for the Group demo are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\security\group\env.properties
UNIX:	$REGISTRY_HOME/demos/security/group/env.properties

Table 13. Properties Used in Demos

Name	Default Value	Description
uddi.demos.url.group	http://localhost:8080/uddi/group	the group Web service port URL
uddi.demos.url.security	http://localhost:8080/uddi/security	the security Web service port URL

3.2.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the WhereIAm demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\security\group\src\demo\uddi\group\WhereIAm.java
UNIX:	$REGISTRY_HOME/demos/security/group/src/demo/uddi/group/WhereIAm.java

The main method starts by gathering configuration information from the user. The first, login name, is used to run the command; the second is argument of the where_amI operation. It then logs the user to the registry, creates the Where_amI object, sends it over SOAP and prints a list of groups to which the login belongs.

```
String user = UserInput.readString("Enter login to authenticate",
                DemoProperties.getProperty(USER_JOHN_NAME));
String password = UserInput.readString("Enter password",
                      DemoProperties.getProperty(USER_JOHN_PASSWORD));
String login = UserInput.readString("Enter login to search", user);
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Where_amI save = createWhereAmI(login, authInfo);
GroupList groups = whereAmI(save);
printGroupList(groups);
discardAuthInfo(authInfo, security);
```

The method createWhereAmI is used to create an object representation of the where_amI operation.

```
public static Where_amI createWhereAmI(String login, String authInfo)
  throws InvalidParameterException {
    System.out.println("login = " + login);

    Where_amI find = new Where_amI();
    find.setLoginName(login);
    find.setAuthInfo(authInfo);

    return find;
}
```

The helper method, getGroupStub(), returns the UDDI Group stub of the Web service listening at the URL specified by the URL_GROUP property.

```
public static GroupApi getGroupStub() throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.group
    String url = DemoProperties.getProperty(URL_GROUP, "http://localhost:8080/uddi/group");
    System.out.print("Using Group API at url " + url + " ..");
    GroupApi account = GroupStub.getInstance(url);
    System.out.println(" done");
    return account;
}
```

The BEA AquaLogic Service Registry API call `where_amI` is performed in the method `whereAmI`.

```
public static GroupList whereAmI(Where_amI find)
  throws SOAPException, GroupException {
    GroupApi groupApi = getGroupStub();
    System.out.print("Search in progress ...");
    GroupList groups = groupApi.where_amI(find);
    System.out.println(" done");
    return groups;
}
```

Finally the method `printGroupList` is used to print the found groups to the console.

```
public static void printGroupList(GroupList groups) {
    System.out.println();
    ListDescription listDescription = groups.getListDescription();
    if (listDescription != null) {
        // list description is mandatory part of result, if the resultant list is subset of
available data
        int includeCount = listDescription.getIncludeCount();
        int actualCount = listDescription.getActualCount();
        int listHead = listDescription.getListHead();
        System.out.println("Displaying " + includeCount + " of " + actualCount + ",
                                                    starting at position " + listHead);
    }

    GroupInfoArrayList groupInfoArrayList = groups.getGroupInfoArrayList();
    if (groupInfoArrayList == null) {
        System.out.println("Nothing found");
        return;
    }

    int position = 1;
    for (Iterator iterator = groupInfoArrayList.iterator(); iterator.hasNext();) {
        GroupInfo group = (GroupInfo) iterator.next();
        System.out.println("Group " + position);
        System.out.println(group.toXML());
        System.out.println();
        System.out.println("*********************************************************");
        position++;
    }
}
```

3.2.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Group demos.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\security\group
UNIX:	$REGISTRY_HOME/demos/security/group

3. Build demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available commands, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command with the name of the demo as parameter. For example, to run the WhereIAm demo, invoke

Windows:	run.bat WhereIAm
UNIX:	./run.sh WhereIAm

The output of this demo will resemble the following:

```
Running WhereIAm demo...
************************************************************************
***    BEA AquaLogic Service Registry Demo - WhereIAm   ***
************************************************************************

Find groups of user where
Enter login to authenticate [demo_john]:
Enter password [demo_john]:
Enter login to search [demo_john]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
login = demo_john
Using Group API at url https://mycomp.com:8443/uddi/group .. done
Search in progress ... done

Group 1
```

```
<groupInfo xmlns="http://systinet.com/uddi/group/5.0">
<name>system#everyone</name>
<description>The special group that contains all users.</description>
<privateGroup>false</privateGroup>
<external>false</external>
</groupInfo>

*******************************************************
Group 2
<groupInfo xmlns="http://systinet.com/uddi/group/5.0">
<name>system#registered</name>
<description>The special group that contains all users who are logged
onto the UDDI registry.</description>
<privateGroup>false</privateGroup>
<external>false</external>
</groupInfo>

*******************************************************
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

3.3. Permission

The BEA AquaLogic Service Registry Permission Demos are used to demonstrate the BEA AquaLogic Service Registry application programming interface's capabilities and to demonstrate how to use this API.

You will learn how to set and search permissions.

The BEA AquaLogic Service Registry security permission demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API:

SetPermission Demonstrates how to construct and fill the `Set_permission` object, get a Permission stub for the UDDI registry, and perform the `set_permission` call.

WhoHasPermission Demonstrates how to construct and fill the `Who_hasPermission` object, get a Permission stub for the UDDI registry, and perform the `who_hasPermission` call.

GetPermission Demonstrates how to construct and fill the `Get_permission` object, get a Permission stub for the UDDI registry, and perform the `get_permission` call.

3.3.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is,, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `Permission` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\security\permission\env.properties
UNIX:	$REGISTRY_HOME/demos/security/permission/env.properties

Table 14. Properties Used in Demos

Name	Default Value	Description
uddi.demos.url.permission	`http://localhost:8080/uddi/permission`	the permission Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

3.3.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the SetPermission demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\security\permission\src\demo\uddi\permission\SetPermission.java
UNIX:	$REGISTRY_HOME/demos/security/permission/src/demo/uddi/permission/SetPermission.java

The main method is divided into two parts. The first part serves to configure the demo by the user. It reads the credentials of the user who will run the demo and is allowed to set permissions. Then it reads permission type, name, and action.

The second part contains the execution of the demo. It looks up the security stub and authenticates the user. It then creates a `Set_permission` object and sends it over SOAP to the UDDI registry as a `set_permission` operation. If the user has explicitly declared permissions that are not present in this operation, these will be removed.

```
String user = UserInput.readString("Enter login","admin");
String password = UserInput.readString("Enter password","changeit");
String principal = UserInput.readString("Enter principal type", PrincipalType.user.getValue());
String login = UserInput.readString("Enter login/group name",
                                                        DemoProperties.getProp-
erty(USER_JOHN_NAME));
String type = UserInput.readString("Enter permission type",
                                    "org.systinet.uddi.security.permission.ApiManagerPermis-
sion");
String name = UserInput.readString("Enter permission name",
                                                "org.systinet.uddi.client.taxonomy.v3.Tax-
onomyApi");
String action = UserInput.readString("Enter action", "download_taxonomy");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
```

```
String authInfo = getAuthInfo(user, password, security);
Set_permission set = createSetPermission(login, principal, name, type, action, authInfo);
setPermission(set);
discardAuthInfo(authInfo, security);
```

The method `createSetPermission` creates an object representing the `set_permission` operation.

```
public static Set_permission createSetPermission(String login, String principal,
  String name, String type, String action, String authInfo) throws InvalidParameterException
{
    System.out.println(principal+", login/name = " + login);
    System.out.println("type = " + type);
    System.out.println("name = " + name);
    System.out.println("action = " + action);

    PermissionDescriptors permissionDescriptors = new PermissionDescriptors();
   permissionDescriptors.setPrincipal(new Principal(login, PrincipalType.getPrincipalType(prin-
cipal)));
    PermissionDescriptor descriptor = new PermissionDescriptor();
    descriptor.setName(name);
    descriptor.setType(type);
    descriptor.addAction(action);
    permissionDescriptors.addPermissionDescriptor(descriptor);

    Set_permission set = new Set_permission();
    set.setPermissionDescriptors(permissionDescriptors);
    set.setAuthInfo(authInfo);

    return set;
}
```

The helper method, `getPermissionStub()`, returns the UDDI Permission stub of the Web service listening at the URL specified by the `URL_PERMISSION` property.

```
public static PermissionApi getPermissionStub() throws SOAPException {
// you can specify your own URL in property - uddi.demos.url.permission
String url = DemoProperties.getProperty(URL_PERMISSION, "http://localhost:8080/uddi/permission");
System.out.print("Using Permission API at url " + url + " ..");
PermissionApi permission = PermissionStub.getInstance(url);
System.out.println(" done");
return permission;
}
```

The BEA AquaLogic Service Registry API call `set_permission` is performed in the method `setPermission`.

```
public static void setPermission(Set_permission set) throws
  SOAPException, PermissionException {
    PermissionApi permissionApi = getPermissionStub();
    System.out.print("Save in progress ...");
    permissionApi.set_permission(set);
    System.out.println(" done");
}
```

3.3.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry Permission demos.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.
2. Change your working directory to

Windows:	%REGISTRY_HOME%\demos\security\permission
UNIX:	$REGISTRY_HOME/demos/security/permission

3. Build demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available commands, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the SetPermission demo, invoke

Windows:	run.bat SetPermission
UNIX:	./run.sh SetPermission

The output of this demo will resemble the following:

```
Running SetPermission demo...
************************************************************************
***   BEA AquaLogic Service Registry Demo: SetPermission   ***
************************************************************************

Setting permission where
Enter login [admin]:
Enter password [changeit]:
Enter principal type [user]:
Enter login/group name [demo_john]:
Enter permission type [org.systinet.uddi.security.permission.ApiManagerPermission]:
Enter permission name [org.systinet.uddi.client.taxonomy.v3.TaxonomyApi]:
Enter action [download_taxonomy]:
```

```
Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
user, login/name = demo_john
type = org.systinet.uddi.security.permission.ApiManagerPermission
name = org.systinet.uddi.client.taxonomy.v3.TaxonomyApi
action = download_taxonomy

Using Permission API at url https://mycomp.com:8443/uddi/permission .. done
Save in progress ... done
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

3.4. ACL

The BEA AquaLogic Service Registry ACL Demos demonstrate the BEA AquaLogic Service Registry ACL application programming interface's capabilities and how to use this API.

The Systinet ACL extension is used to grant or revoke rights to selected users or groups. You will learn how to create, save, delete, get and find ACLs.

The BEA AquaLogic Service Registry Security ACL demo set contains the following demos to assist you in learning the BEA AquaLogic Service Registry client API:

Create Demonstrates how to use Create ACL to give one user rights to create a service in the business entity of another user.

Save Demonstrates how to use Save ACL to give one user rights to update the business entity of another user.

Delete Demonstrates how to use Delete ACL to give one user rights to delete a business entity of another user.

Get Demonstrates how to use Get ACL to revoke from a selected user the right to get the business detail of a business entity.

Find Demonstrates how to use Find ACL to hide the business entity in a find_business operation from a selected user.

3.4.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine a property's value for a single demo (that is,, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `ACL` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\security\acl\env.properties
UNIX:	$REGISTRY_HOME/demos/security/acl/env.properties

Table 15. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.user.jane.name	demo_jane	second user's name
uddi.demos.user.jane.password	demo_jane	second user's password
uddi.demos.url.publishing	`http://localhost:8080/uddi/publishing`	The publication Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

3.4.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the Find demo as an example. You can find this demo's source code in the file:

Windows:	%REGISTRY_HOME%\demos\security\acl\src\demo\uddi\acl\Find.java
UNIX:	$REGISTRY_HOME/demos/security/acl/src/demo/uddi/acl/Find.java

The main method is divided into several logical parts. The first part is used to configure the demo for the user. The "good" user represents the user who will receive a positive ACL; the "bad" user represents the user who will receive a negative ACL.

The second part contains the `save_business` operation with extra information. The ACLs are set in the categoryBag. In the next section, the bad user unsuccessfully tries to find the business entity by name, and finally the good user finds the business entity.

```
String name = UserInput.readString("Enter business name", "ACL find demo");
String description = UserInput.readString("Enter description",
                                              "Demonstration of find-allowed, find-denied
ACLs");
String searchName = UserInput.readString("Enter search string", "ACL%");
String owner = UserInput.readString("Enter entity owner", "admin");
String password = UserInput.readString("Enter owner's password", "changeit");
String loginGood = UserInput.readString("Enter good user's login",
                                                  DemoProperties.getProp-
erty(USER_JOHN_NAME));
String passwordGood = UserInput.readString("Enter good user's password",
                                                DemoProperties.getProperty(USER_JOHN_PASS-
WORD));
String loginBad = UserInput.readString("Enter bad user's login",
                                                  DemoProperties.getProp-
erty(USER_JANE_NAME));
```

```
String passwordBad = UserInput.readString("Enter bad user's password",
                                                    DemoProperties.getProperty(USER_JANE_PASS-
WORD));
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfoOwner = getAuthInfo(owner, password, security);
Save_business saveBusiness = createSaveBusiness(name, description, loginGood, loginBad, authIn-
foOwner);
BusinessDetail result = saveBusiness(saveBusiness);
printBusinessDetail(result);
discardAuthInfo(authInfoOwner, security);

System.out.println(" ");
System.out.println("Finding business entity where");
String authInfoGood = getAuthInfo(loginGood, passwordGood, security);
Find_business findBusiness = createFindByName(searchName, authInfoGood);
BusinessList businessList = findBusiness(findBusiness);
printBusinessList(businessList);
discardAuthInfo(authInfoGood, security);

System.out.println(" ");
System.out.println("Finding business entity where");
String authInfoBad = getAuthInfo(loginBad, passwordBad, security);
findBusiness = createFindByName(searchName, authInfoBad);
businessList = findBusiness(findBusiness);
printBusinessList(businessList);
discardAuthInfo(authInfoGood, security);
```

The `createSaveBusiness` operation is used to create the `Save_business` object. The ACLs are stored in the keyedReferenceGroup with the `uddi:systinet.com:acl` tModelKey as keyedReference, where the tModelKey specifies the tModelKey of the ACL, keyValue holds the login name of the user or group, and finally keyName is used to distinguish between users and groups in the keyValue.

```
public static Save_business createSaveBusiness(String name,
                                                        String description, String goodUser,

  String badUser, String authInfo) throws InvalidParameterException {
    System.out.println("name = " + name);
    System.out.println("description = " + description);
    System.out.println("goodUser = " + goodUser);
    System.out.println("badUser = " + badUser);

    BusinessEntity businessEntity = new BusinessEntity();
    businessEntity.addName(new Name(name));
    businessEntity.addDescription(new Description(description));

    CategoryBag categoryBag = new CategoryBag();
    businessEntity.setCategoryBag(categoryBag);
    KeyedReferenceGroup aclGroup = new KeyedReferenceGroup("uddi:systinet.com:acl");
    aclGroup.addKeyedReference(new KeyedReference("uddi:systinet.com:acl:find-allowed",
                                                                                goodUser,
"user"));
    aclGroup.addKeyedReference(new KeyedReference("uddi:systinet.com:acl:find-denied",
```

```
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                badUser,
"user"));
    categoryBag.addKeyedReferenceGroup(aclGroup);

    Save_business save = new Save_business();
    save.addBusinessEntity(businessEntity);
    save.setAuthInfo(authInfo);

    return save;
}
```

The `find_business` operation takes the *`authInfo`* parameter used to identify the user who runs the query.

```
public static Find_business createFindByName(String name, String authInfo)
  throws InvalidParameterException {
System.out.println("name = " + name);
Find_business find_business = new Find_business();
find_business.addName(new Name(name));
find_business.setMaxRows(new Integer(MAX_ROWS));
find_business.setAuthInfo(authInfo);
find_business.addFindQualifier("approximateMatch");
return find_business;
}
```

3.4.3. Building and Running Demos

This section shows how to build and run the BEA AquaLogic Service Registry ACL demos.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to:

Windows:	%REGISTRY_HOME%\demos\security\acl
UNIX:	$REGISTRY_HOME/demos/security/acl

3. Build demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available commands, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command with the name of the demo as parameter. For example, to run the Find demo, invoke

Windows:	run.bat Find
UNIX:	./run.sh Find

The output of this demo will resemble the following:

```
Running Find demo...
***************************************************************************
***    BEA AquaLogic Service Registry Demo - ACLFind   ***
***************************************************************************

Saving business entity where
Enter business name [ACL find demo]:
Enter description [Demonstration of find-allowed, find-denied ACLs]:
Enter search string [ACL%]:
Enter entity owner [admin]:
Enter owner's password [changeit]:
Enter good user's login [demo_john]:
Enter good user's password [demo_john]:
Enter bad user's login [demo_jane]:
Enter bad user's password [demo_jane]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Authenticating the user admin .. done
name = ACL find demo
description = Demonstration of find-allowed, find-denied ACLs
goodUser = demo_john
badUser = demo_jane
Using Publishing at url https://mycomp.com:8443/uddi/publishing .. done
Save business in progress ... done

Business 1 : uddi:91ba8390-a8e0-11d8-b2ad-779f83c0b2ad
<businessEntity businessKey="uddi:91ba8390-a8e0-11d8-b2ad-779f83c0b2ad"
xmlns="urn:uddi-org:api_v3">
<name>ACL find demo</name>
<description>Demonstration of find-allowed, find-denied ACLs</description>
<categoryBag>
<keyedReferenceGroup tModelKey="uddi:systinet.com:acl">
<keyedReference tModelKey="uddi:systinet.com:acl:find-allowed"
keyName="user" keyValue="demo_john"/>
<keyedReference tModelKey="uddi:systinet.com:acl:find-denied"
keyName="user" keyValue="demo_jane"/>
</keyedReferenceGroup>
</categoryBag>
</businessEntity>

Logging out .. done

Finding business entity where
Authenticating the user demo_john .. done
name = ACL%
```

```
Using Inquiry at url http://mycomp.com:8080/uddi/inquiry .. done
Search in progress .. done

Displaying 1 of 1, starting at position 1
Business 1 : uddi:91ba8390-a8e0-11d8-b2ad-779f83c0b2ad
<businessInfo businessKey="uddi:91ba8390-a8e0-11d8-b2ad-779f83c0b2ad"
xmlns="urn:uddi-org:api_v3">
<name>ACL find demo</name>
<description>Demonstration of find-allowed, find-denied ACLs</description>
</businessInfo>

Logging out .. done

Finding business entity where
Authenticating the user demo_jane .. done
name = ACL%
Using Inquiry at url http://mycomp.com:8080/uddi/inquiry .. done
Search in progress .. done

Displaying 0 of 0, starting at position 1
Nothing found
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

4. Resources Demos

The Resources Demos section includes the following demos:

- WSDL - Teaches how to publish, unpublish and find a WSDL document in UDDI version 2 and UDDI version 3.
- XML - Teaches how to publish, unpublish and find an XML document.
- XSD - Teaches how to publish, unpublish and find an XML Schema.
- XSLT - Teaches how to publish, unpublish and find a XSL Transformation.

4.1. WSDL2UDDI v2

The BEA AquaLogic Service Registry WSDL2UDDI demo set is used to demonstrate the BEA AquaLogic Service Registry WSDL2UDDI application programming interface's capabilities and to demonstrate how to use this API. The BEA AquaLogic Service Registry WSDL2UDDI demos cover the UDDI Version 2.0.4 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv2]. You will learn how to query and publish a WSDL to a UDDI registry over a SOAP interface. The BEA AquaLogic Service Registry WSDL2UDDI demo set contains following demos to assist you in learning the WSDL2UDDI client API.

PublishWSDL Demonstrates how to construct and fill the `Publish_wsdl` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `publish_wsdl` call.

UnPublishWSDL Demonstrates how to construct and fill the `Unpublish_wsdl` object, get WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `unpublish_wsdl` call.

FindWSDL Demonstrates how to construct and fill the `Find_wsdlServiceInfo` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `find_wsdlServiceInfo` call.

GetWSDL Demonstrates how to construct and fill the `Get_wsdlServiceInfo` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `get_wsdlServiceInfo` call.

4.1.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `WSDL2UDDI` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\basic\wsdl\v2\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/wsdl/v2/env.properties

Table 16. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.wsdl2uddi	`http://local-host:8080/uddi/wsdl2uddi`	the wsdl2uddi Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/secur-ity`	the security Web service port URL

4.1.2. Presentation and Functional Presentation

This section describes programming pattern used in all demos using the PublishWSDL demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\basic\wsdl2uddi\src\demo\uddi\v2\wsdl2uddi\PublishWSDL.java
UNIX:	$REGISTRY_HOME/demos/basic/wsdl2uddi/src/demo/uddi/v2/wsdl2uddi/PublishWSDL.java

The main method is very short. After gathering the user's input, it gets the security stub and authorizes the user. The resulting authInfo string is a secret key passed to the Publish request, which is created and initialized in the `createPublish()` method.

The user's choice of WSDL is published to the selected businessEntity within the `publishWSDL()` method.

When successful, the WsdlDetail object is returned from the UDDI registry and printed.

The last step is to discard the authInfo string, so that no malicious user can use it to compromise another user's account.

```
String businessKey = UserInput.readString("Enter businessKey",
       "d7222f66-08aa-3a6e-a299-2ed4ac785682");
String url = UserInput.readString("Enter WSDL URL",
              "http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Publish_wsdl publish = createPublish(businessKey, url, authInfo);
WsdlDetail result = publishWSDL(publish);
printWsdlDetail(result);
discardAuthInfo(authInfo, security);
```

The helper method `getSecurityStub()` returns the UDDI Security stub of the Web service listening at the URL specified by the `URL_SECURITY` property.

```
public static UDDI_Security_PortType getSecurityStub()
  throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.security
    String url = DemoProperties.getProperty(URL_SECURITY,
                            "http://localhost:8080/uddi/security");
    System.out.print("Using Security at url " + url + " ..");
    UDDI_Security_PortType security = UDDISecurityStub.getInstance(url);
    System.out.println(" done");
    return security;
}
```

Similarly, the helper method `getWsdl2uddiStub()` returns the WSDL2UDDI stub of the Web service listening at URL specified by the `URL_WSDL2UDDI` property.

```
public static Wsdl2uddiApi getWsdl2uddiStub() throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.wsdl2uddi
    String url = DemoProperties.getProperty(URL_WSDL2UDDI,
                                            "http://localhost:8080/uddi/wsdl2uddi");
    System.out.print("Using WSDL2UDDI at url " + url + " ..");
    Wsdl2uddiApi inquiry = Wsdl2uddiStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The `getAuthInfo()` method is used to authorize the user against the UDDI registry and to get the secret authInfo key.

```
public static String getAuthInfo(String userName,
                        String password, UDDI_Security_PortType security)
    throws InvalidParameterException, UDDIException {
    System.out.print("Logging in ..");
```

```
    AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
    System.out.println(" done");
    return authToken.getAuthInfo();
}
```

The `discardAuthInfo()` method invalidates the secret authInfo key, so that it cannot be reused.

```
public static DispositionReport discardAuthInfo(String authInfo,
                                                    UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging out ..");
    DispositionReport dispositionReport = security.discard_authToken(new Discard_authToken(au-
thInfo));
    System.out.println(" done");
    return dispositionReport;
}
```

The `createPublish()` method is used to create a new instance of the `Publish` class and initialize it with values from parameters:

```
public static Publish_wsdl createPublish(String businessKey,
                                                 String url, String authInfo)
  throws InvalidParameterException {
    System.out.println("businessKey = " + businessKey);
    System.out.println("url = " + url);

    WsdlMapping wsdlMapping = new WsdlMapping();
    wsdlMapping.setBusinessKey(businessKey);
    Wsdl wsdl = new Wsdl(url);
    WsdlDetail wsdlDetail = new WsdlDetail(wsdl, wsdlMapping);
    Publish_wsdl publish = new Publish_wsdl(wsdlDetail, authInfo);
    return publish;
}
```

The WSDL2UDDI API call `Publish_wsdl` is performed in the method `publishWSDL()`.

```
public static WsdlDetail publishWSDL(Publish_wsdl save)
  throws UDDIException, SOAPException {
    Wsdl2uddiApi publishing = getWsdl2uddiStub();
    System.out.print("Save in progress ...");
    WsdlDetail wsdlDetail = publishing.publish_wsdl(save);
    System.out.println(" done");
    return wsdlDetail;
}
```

The returned WsdlDetail is displayed by the `printWsdlDetail()` method.

One interesting aspect of BEA AquaLogic Service Registry client API is that each UDDIObject contains the `toXML()` method, which returns a human-readable formatted listing of its XML representation.

```
public static void printWsdlDetail(WsdlDetail wsdlDetail) {
    System.out.println();
    System.out.println(wsdlDetail.toXML());
}
```

4.1.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\basic\wsdl\v2
UNIX	$REGISTRY_HOME/demos/basic/wsdl/v2

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of demo as parameter. For example, to run the PublishWSDL demo, invoke

Windows:	run.bat PublishWSDL
UNIX:	./run.sh PublishWSDL

The output of this demo will resemble the following:

```
Running PublishWSDL demo...
*******************************************************************
***              BEA AquaLogic Service Registry Demo - PublishWSDL              ***
*******************************************************************

Publishing WSDL where
Enter businessKey [d7222f66-08aa-3a6e-a299-2ed4ac785682]:
Enter WSDL URL [http://localhost:8080/uddi/inquiry/wsdl]:
       http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl

Using Publishing at url https://mycomp.com:8443/uddi/publishing .. done
Logging in .. done
```

```
businessKey = d7222f66-08aa-3a6e-a299-2ed4ac785682
url = http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl
Using WSDL2UDDI at url https://mycomp.com:8443/uddi/wsdl2uddi .. done
Save in progress ... done

<wsdlDetail xmlns="http://systinet.com/uddi/wsdl2uddi/v2/5.0">
    <wsdl>
       <wsdlLocation>http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl</wsdlLocation>

    </wsdl>
    <wsdlMapping>
        <businessKey xmlns="urn:uddi-org:api_v2">d7222f66-08aa-3a6e-a299-2ed4ac785682<
                      /businessKey>
        <services>
            <service name="EmployeeList" namespace="
                                  http://systinet.com/wsdl/demo/uddi/services/"
              publishingMethod="rewrite">
                <serviceKey xmlns="urn:uddi-org:api_v2">
                       d0a50390-af1c-11d8-b9bf-eb2d7e20b9bf</serviceKey>
                    <ports>
                        <port name="EmployeeList" publishingMethod="rewrite">
                        <bindingKey xmlns="urn:uddi-org:api_v2">
                     d0aca4b0-af1c-11d8-b9bf-eb2d7e20b9bf</bindingKey>
                    </port>
                </ports>
            </service>
        </services>
        <bindings>
            <binding name="EmployeeList_binding"
                           namespace="http://systinet.com/wsdl/demo/uddi/services/"
              publishingMethod="rewrite">
                <tModelKey xmlns="urn:uddi-org:api_v2">
                    uuid:d07da570-af1c-11d8-b9bf-eb2d7e20b9bf</tModelKey>
            </binding>
        </bindings>
        <portTypes>
            <portType name="EmployeeList_portType"
                           namespace="http://systinet.com/wsdl/demo/uddi/services/"
              publishingMethod="rewrite">
                <tModelKey xmlns="urn:uddi-org:api_v2">
                           uuid:d0658990-af1c-11d8-b9bf-eb2d7e20b9bf</tModelKey>
            </portType>
        </portTypes>
    </wsdlMapping>
</wsdlDetail>
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

4.2. WSDL2UDDI v3

The BEA AquaLogic Service Registry WSDL2UDDI demo set is used to demonstrate the BEA AquaLogic Service Registry WSDL2UDDI application programming interface's capabilities and to show how to use this API. The BEA AquaLogic Service Registry WSDL2UDDI demos cover the UDDI Version 3.01 Specification [http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3]. You will learn how to query and publish a WSDL to a UDDI registry over a SOAP interface.

The BEA AquaLogic Service Registry WSDL2UDDI demo set contains following demos to assist you in learning the WSDL2UDDI client API.

PublishWSDL Demonstrates how to construct and fill the `Publish_wsdl` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `publish_wsdl` call.

UnPublishWSDL Demonstrates how to construct and fill the `Unpublish_wsdl` object, get WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `unpublish_wsdl` call.

FindWSDL Demonstrates how to construct and fill the `Find_wsdlServiceInfo` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `find_wsdlServiceInfo` call.

GetWSDL Demonstrates how to construct and fill the `Get_wsdlServiceInfo` object, get the WSDL2UDDI stub for the UDDI registry, get an authToken, and perform the `get_wsdlServiceInfo` call.

4.2.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your BEA AquaLogic Service Registry must be running. To start the registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during installation of the BEA AquaLogic Service Registry work out of the box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `WSDL2UDDI` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\basic\wsdl\v3\env.properties
UNIX:	$REGISTRY_HOME/demos/basic/wsdl/v3/env.properties

Table 17. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.wsdl2uddi	`http://local-host:8080/uddi/wsdl2uddi`	the wsdl2uddi Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

4.2.2. Presentation and Functional Presentation

This section describes programming pattern used in all demos using the PublishWSDL demo as an example. You can find its source code in file

Windows:	%REGISTRY_HOME%\demos\basic\wsdl2uddi\src\demo\uddi\v3\wsdl2uddi\PublishWSDL.java
UNIX:	$REGISTRY_HOME/demos/basic/wsdl2uddi/src/demo/uddi/v3/wsdl2uddi/PublishWSDL.java

The main method is very short. After gathering the user's input, it gets the security stub and authorizes the user. The resulting authInfo string is a secret key passed to the Publish request, which is created and initialized in the `createPublish()` method.

The user's choice of WSDL is published to the selected businessEntity within the `publishWSDL()` method.

When successful, the WsdlDetail object is returned from the UDDI registry and printed.

The last step is to discard the authInfo string, so that no malicious user can use it to compromise another user's account.

```
String businessKey = UserInput.readString("Enter businessKey", "uddi:systinet.com:demo:hq");
String url = UserInput.readString("Enter WSDL URL", "http://localhost:8080/uddi/doc/demos/Em-
ployeeList.wsdl");
System.out.println();

UDDI_Security_PortType security = getSecurityStub();
String authInfo = getAuthInfo(user, password, security);
Publish_wsdl publish = createPublish(businessKey, url, authInfo);
WsdlDetail result = publishWSDL(publish);
printWsdlDetail(result);
discardAuthInfo(authInfo, security);
```

The helper method `getSecurityStub()` returns the UDDI Security stub of the Web service listening at the URL specified by the `URL_SECURITY` property.

```
public static UDDI_Security_PortType getSecurityStub()
 throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.security
   String url = DemoProperties.getProperty(URL_SECURITY, "http://localhost:8080/uddi/security");

    System.out.print("Using Security at url " + url + " ..");
    UDDI_Security_PortType security = UDDISecurityStub.getInstance(url);
    System.out.println(" done");
    return security;
}
```

Similarly, the helper method `getWsdl2uddiStub()` returns the WSDL2UDDI stub of the Web service listening at URL specified by the `URL_WSDL2UDDI` property.

```
public static Wsdl2uddiApi getWsdl2uddiStub() throws SOAPException {
    // you can specify your own URL in property - uddi.demos.url.wsdl2uddi
    String url = DemoProperties.getProperty(URL_WSDL2UDDI, "http://local-
host:8080/uddi/wsdl2uddi");
    System.out.print("Using WSDL2UDDI at url " + url + " ..");
    Wsdl2uddiApi inquiry = Wsdl2uddiStub.getInstance(url);
    System.out.println(" done");
    return inquiry;
}
```

The `getAuthInfo()` method is used to authorize the user against the UDDI registry and to get the secret authInfo key.

```
public static String getAuthInfo(String userName, String password, UDDI_Security_PortType se-
curity)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging in ..");
    AuthToken authToken = security.get_authToken(new Get_authToken(userName, password));
    System.out.println(" done");
    return authToken.getAuthInfo();
}
```

The `discardAuthInfo()` method invalidates the secret authInfo key, so that it cannot be reused.

```
public static void discardAuthInfo(String authInfo, UDDI_Security_PortType security)
  throws InvalidParameterException, UDDIException {
    System.out.print("Logging out ..");
    security.discard_authToken(new Discard_authToken(authInfo));
    System.out.println(" done");
}
```

The `createPublish()` method is used to create a new instance of the `Publish` class and initialize it with values from parameters:

```
public static Publish_wsdl createPublish(String businessKey, String url, String authInfo)
  throws InvalidParameterException {
    System.out.println("businessKey = " + businessKey);
    System.out.println("url = " + url);

    WsdlMapping wsdlMapping = new WsdlMapping();
    wsdlMapping.setBusinessKey(businessKey);
    Wsdl wsdl = new Wsdl(url);
    WsdlDetail wsdlDetail = new WsdlDetail(wsdl, wsdlMapping);
    Publish_wsdl publish = new Publish_wsdl(wsdlDetail, authInfo);
    return publish;
}
```

The WSDL2UDDI API call `Publish_wsdl` is performed in the method `publishWSDL()`.

```
public static WsdlDetail publishWSDL(Publish_wsdl save)
  throws UDDIException, SOAPException {
    Wsdl2uddiApi publishing = getWsdl2uddiStub();
    System.out.print("Save in progress ...");
    WsdlDetail wsdlDetail = publishing.publish_wsdl(save);
```

```
    System.out.println(" done");
    return wsdlDetail;
}
```

The returned WsdlDetail is displayed by the `printWsdlDetail()` method.

One interesting aspect of BEA AquaLogic Service Registry client API is that each UDDIObject contains the `toXML()` method, which returns a human-readable formatted listing of its XML representation.

```
public static void printWsdlDetail(WsdlDetail wsdlDetail) {
    System.out.println();
    System.out.println(wsdlDetail.toXML());
}
```

4.2.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry Basic Publishing demo set. Let's continue with our SaveBusiness demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\basic\wsdl\v3
UNIX	$REGISTRY_HOME/demos/basic/wsdl/v3

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the PublishWSDL demo, invoke

Windows:	run.bat PublishWSDL
UNIX:	./run.sh PublishWSDL

The output of this demo will resemble the following:

```
Running PublishWSDL demo...
*******************************************************************
***              BEA AquaLogic Service Registry Demo - PublishWSDL              ***
*******************************************************************

Publishing WSDL where
Enter businessKey [uddi:systinet.com:demo:hq]:
Enter WSDL URL [http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl]:

Using Security at url https://mycomp.com:8443/uddi/security .. done
Logging in .. done
businessKey = uddi:systinet.com:demo:hq
url = http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl
Using WSDL2UDDI at url https://mycomp.com:8443/uddi/wsdl2uddi .. done
Save in progress ... done

<wsdlDetail xmlns="http://systinet.com/uddi/wsdl2uddi/v3/5.0">
    <wsdl>
       <wsdlLocation>http://localhost:8080/uddi/doc/demos/EmployeeList.wsdl</wsdlLocation>

    </wsdl>
    <wsdlMapping>
    <businessKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:hq</businessKey>
    <services>
       <service name="EmployeeList" namespace="http://systinet.com/wsdl/demo/uddi/services/"

        publishingMethod="rewrite">
        <serviceKey xmlns="urn:uddi-org:api_v3">uddi:dde19a70-af1a-11d8-b9bf-
eb2d7e20b9bf</serviceKey>
            <ports>
                <port name="EmployeeList" publishingMethod="rewrite">
                    <bindingKey xmlns="urn:uddi-org:api_v3">uddi:dde85130-af1a-11d8-b9bf-
eb2d7e20b9bf</bindingKey>
                </port>
            </ports>
        </service>
    </services>
    <bindings>
        <binding name="EmployeeList_binding" namespace="http://systin-
et.com/wsdl/demo/uddi/services/"
        publishingMethod="rewrite">
            <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ddc84610-af1a-11d8-b9bf-
eb2d7e20b9bf</tModelKey>
        </binding>
    </bindings>
    <portTypes>
        <portType name="EmployeeList_portType" namespace="http://systin-
et.com/wsdl/demo/uddi/services/"
        publishingMethod="rewrite">
            <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ddbc3820-af1a-11d8-b9bf-
eb2d7e20b9bf</tModelKey>
        </portType>
    </portTypes>
```

```
    </wsdlMapping>
</wsdlDetail>
Logging out .. done
```

6. To rebuild demos, execute `run.bat clean` (`./run.sh clean`) to delete the classes directory and `run.bat make` (`./run.sh make`) to rebuild the demo classes.

4.3. XML2UDDI

The BEA AquaLogic Service Registry XML2UDDI demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and shows how to use the XML2UDDI API to manipulate XML documents.

The demos set include the following demos:

- `FindXml`
- `FindXmlMapping`
- `GetXmlDetail`
- `PublishXml`
- `UnpublishXml`

4.3.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	cd %REGISTRY_HOME%\bin serverstart
UNIX:	cd $REGISTRY_HOME/bin ./serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `XML2UDDI` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\resources\xml\env.properties
UNIX:	$REGISTRY_HOME/demos/resources/xml/env.properties

Table 18. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.xml2uddi	`http://localhost:8080/uddi/xml2uddi`	the xml2uddi web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security web service port URL

4.3.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the PublishXml demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\resources\xml\src\demo\uddi\xml\PublishXml.java
UNIX:	$REGISTRY_HOME/demos/resources/xml/src/demo/uddi/xml/PublishXml.java

The helper method `createPublishXml` creates a Publish_xml structure:

```
public Publish_xml createPublishXml(String location, String publishingMethod, String nsPublish-
Method, String nsPublishPolicy,
                                                String authInfo) throws InvalidParameterException
 {
        System.out.println("location = " + location);

        Publish_xml publish = new Publish_xml();
        publish.setLocation(location);
       publish.setPublishingMethod(XmlPublishingMethod.getXmlPublishingMethod(publishingMeth-
od));
        publish.setPolicy(PublishPolicy.getPublishPolicy(nsPublishMethod));
       publish.setNamespacePublishingMethod(NsPublishingMethod.getNsPublishingMethod(nsPublish-
Policy));
        publish.setAuthInfo(authInfo);

        return publish;
}
```

The `publishXmlResource` method performs the publishing operation:

```
public XmlResourceDetail publishXmlResource(Publish_xml publish) throws UDDIException, SOAPEx-
ception {
        System.out.print("Check structure validity .. ");
        try {
            publish.check();
        } catch (InvalidParameterException e) {
            System.out.println("Failed!");
            throw new UDDIException(e);
```

```
        }
        System.out.println("OK");

        Xml2uddiApi xmlApi = getXml2UddiStub();
        System.out.print("Publishing in progress ...");
        XmlResourceDetail xmlDetail = xmlApi.publish_xml(publish);
        System.out.println(" done");
        return xmlDetail;
}
```

4.3.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry XML2UDDI demo set. Let us continue with our PublishXml demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\resources\xml
UNIX	$REGISTRY_HOME/demos/resources/xml

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

. This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the PublishWSDL demo, invoke

Windows:	run.bat PublishXml
UNIX:	./run.sh PublishXml

The output of this demo will resemble the following:

```
*********************************************************************
***              BEA AquaLogic Service Registry Demo - PublishXml              ***
*********************************************************************

Publishing an XML document with the following parameters:
Enter XML location (URI) [http://localhost:8080/uddi/doc/demos/employees.xml]:
Enter publishing method (update,create) [update]:
Enter import publishing policy (automatic,explicit,locations) [automatic]:
Enter import publishing method (reuse,create,ignore) [reuse]:

Using Security at url https://localhost:8443/uddi/security ..

 done
Logging in .. done
location = http://localhost:8080/uddi/doc/demos/employees.xml
Check structure validity .. OK
Using XML2UDDI at url https://localhost:8443/uddi/xml2uddi .. done
Publishing in progress ... done

XML http://localhost:8080/uddi/doc/demos/employees.xml
<xmlResourceDetail xmlns="http://systinet.com/uddi/xml2uddi/v3/5.5">
  <xmlResourceInfo>
    <location>http://localhost:8080/uddi/doc/demos/employees.xml</location>
    <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xml:employees</tModelKey>

    <name xmlns="urn:uddi-org:api_v3">employees.xml</name>
  </xmlResourceInfo>
  <namespace>
    <uri>http://systinet.com/uddi/demo/employeeList</uri>
    <namespaceModel>
      <name xmlns="urn:uddi-org:api_v3">employees.xsd</name>
     <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xsd:employees</tModelKey>

    </namespaceModel>
  </namespace>
</xmlResourceDetail>
Logging out .. done
```

4.4. XSD2UDDI

The BEA AquaLogic Service Registry XSD2UDDI demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and shows how to use the XSD2UDDI API to manipulate XSD documents.

The demos set includes the following demos:

- `FindXsd`
- `FindXsdMapping`
- `GetXsdDetail`
- `PublishXsd`

- `UnpublishXsd`

4.4.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `XSD2UDDI` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\resources\xsd\env.properties
UNIX:	$REGISTRY_HOME/demos/resources/xsd/env.properties

Table 19. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.xsd2uddi	`http://localhost:8080/uddi/xsd2uddi`	the xsd2uddi web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security web service port URL

4.4.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the PublishXsd demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\resources\xsd\src\demo\uddi\xsd\PublishXsd.java
UNIX:	$REGISTRY_HOME/demos/resources/xsd/src/demo/uddi/xsd/PublishXsd.java

The helper method `createPublishXsd` creates a Publish_xsd structure:

```
public Publish_xsd createPublishXsd(String location, String publishingMethod, String importMeth-
od, String importPolicy,
                                                String contentMethod, String contentPolicy,
```

```
String authInfo) throws InvalidParameterException {
        System.out.println("location = " + location);

        Publish_xsd publish = new Publish_xsd();
        publish.setLocation(location);
       publish.setPublishingMethod(XsdPublishingMethod.getXsdPublishingMethod(publishingMeth-
od));
        publish.setImportPolicy(ImportPublishPolicy.getImportPublishPolicy(importMethod));
       publish.setImportPublishingMethod(ImportPublishingMethod.getImportPublishingMethod(im-
portPolicy));
       publish.setContentPolicy(ContentPublishPolicy.getContentPublishPolicy(contentPolicy));

        publish.setContentPublishingMethod(ContentPublishingMethod.getContentPublishingMeth-
od(contentMethod));
        publish.setAuthInfo(authInfo);

        return publish;
}
```

The `publishXsdResource` method performs the publishing operation:

```
public XsdDetail publishXsdResource(Publish_xsd publish) throws UDDIException, SOAPException
{
        System.out.print("Check structure validity .. ");
        try {
            publish.check();
        } catch (InvalidParameterException e) {
            System.out.println("Failed!");
            throw new UDDIException(e);
        }
        System.out.println("OK");

        Xsd2uddiApi xsdApi = getXsd2UddiStub();
        System.out.print("Publishing in progress ...");
        XsdDetail xsdDetail = xsdApi.publish_xsd(publish);
        System.out.println(" done");
        return xsdDetail;
}
```

4.4.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry XSD2UDDI demo set. Let us continue with our PublishXsd demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\resources\xsd
UNIX	$REGISTRY_HOME/demos/resources/xsd

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

. This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the PublishWSDL demo, invoke

Windows:	run.bat PublishXsd
UNIX:	./run.sh PublishXsd

The output of this demo will resemble the following:

```
Running PublishXsd demo...
*********************************************************************
***              BEA AquaLogic Service Registry Demo - PublishXsd              ***
*********************************************************************

Publishing XML schema with the following parameters:
Enter XSD location (URI) [http://localhost:8080/uddi/doc/demos/employees.xsd]:
Enter publishing method (update,create) [update]:
Enter import publishing policy (all,explicit) [all]:
Enter import publishing method (reuse,create,ignore) [reuse]:
Enter content publishing policy (all,explicit) [all]:
Enter content publishing method (reuse,create,ignore) [reuse]:

Using Security at url https://localhost:8443/uddi/security .. done
Logging in .. done
location = http://localhost:8080/uddi/doc/demos/employees.xsd
Check structure validity .. OK
Using XSD2UDDI at url https://localhost:8443/uddi/xsd2uddi .. done
Publishing in progress ... done

XML Schema http://localhost:8080/uddi/doc/demos/employees.xsd
<xsdDetail xmlns="http://systinet.com/uddi/xsd2uddi/v3/5.5">
  <xsdInfo>
    <location>http://localhost:8080/uddi/doc/demos/employees.xsd</location>
```

```
    <namespace>http://systinet.com/uddi/demo/employeeList</namespace>
    <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xsd:employees</tModelKey>

    <name xmlns="urn:uddi-org:api_v3">employees.xsd</name>
  </xsdInfo>
  <elements>
    <element>
      <localName>persons</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">persons</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca43cec0-20f8-11d9-9c6a-
1d0743509c6a</tModelKey>
      </symbolModel>
    </element>
    <element>
      <localName>person</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">person</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca5e82b0-20f8-11d9-9c6a-
1d0743509c6a</tModelKey>
      </symbolModel>
    </element>
    <element>
      <localName>department</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">department</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca6a90a0-20f8-11d9-9c6a-
1d0743509c6a</tModelKey>
      </symbolModel>
    </element>
  </elements>
  <types>
    <type>
      <localName>persons</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">persons</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca742d90-20f8-11d9-9c6a-
1d0743509c6a</tModelKey>
      </symbolModel>
    </type>
    <type>
      <localName>person</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">person</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca856ba0-20f8-11d9-9c6a-
1d0743509c6a</tModelKey>
      </symbolModel>
    </type>
    <type>
      <localName>department</localName>
      <symbolModel>
        <name xmlns="urn:uddi-org:api_v3">department</name>
        <tModelKey xmlns="urn:uddi-org:api_v3">uddi:ca908f30-20f8-11d9-9c6a-
```

```
1d0743509c6a</tModelKey>
      </symbolModel>
    </type>
  </types>
</xsdDetail>
Logging out .. donee
```

4.5. XSLT2UDDI

The BEA AquaLogic Service Registry XSLT2UDDI demo set demonstrates the BEA AquaLogic Service Registry application programming interface's capabilities and shows how to use the XSLT2UDDI API to manipulate XSLT documents.

The demos set includes the following demos:

- `FindXslt`
- `FindXsltMapping`
- `GetXsltDetail`
- `PublishXslt`
- `UnpublishXslt`

4.5.1. Prerequisites and Preparatory Steps: Code

We expect that you have already installed the BEA AquaLogic Service Registry and set the `REGISTRY_HOME` environment variable to the registry's installation location.

To run the BEA AquaLogic Service Registry's demos, your registry must be running. To start the BEA AquaLogic Service Registry, execute the **serverstart** script:

Windows:	%REGISTRY_HOME%\bin\serverstart.bat
UNIX:	$REGISTRY_HOME/bin/serverstart.sh

It is necessary to configure the demos. The configuration system has two levels: global and local. The properties defined at the global level may be overwritten at the local level. The global properties are located in the file:

Windows:	%REGISTRY_HOME%\demos\env.properties
UNIX:	$REGISTRY_HOME/demos/env.properties

The values set during the installation of the BEA AquaLogic Service Registry work out of box, and their modification affects all demos. If you need to redefine the value of some property for a single demo (that is, at the local level), edit `env.properties`. This file is located in the same directory as the file `run.sh` (`run.bat`). Local level properties for the `XSLT2UDDI` demos are loaded from the file:

Windows:	%REGISTRY_HOME%\demos\resources\xslt\env.properties
UNIX:	$REGISTRY_HOME/demos/resources/xslt/env.properties

Table 20. Properties Used in Demos

Name	Default Value	Description
uddi.demos.user.john.name	demo_john	first user's name
uddi.demos.user.john.password	demo_john	first user's password
uddi.demos.url.xslt2uddi	`http://localhost:8080/uddi/xslt2uddi`	the xslt2uddi Web service port URL
uddi.demos.url.security	`http://localhost:8080/uddi/security`	the security Web service port URL

4.5.2. Presentation and Functional Presentation

This section describes the programming pattern used in all demos using the PublishXslt demo as an example. You can find its source code in the file:

Windows:	%REGISTRY_HOME%\demos\resources\xslt\src\demo\uddi\xslt\PublishXslt.java
UNIX:	$REGISTRY_HOME/demos/resources/xslt/src/demo/uddi/xslt/PublishXslt.java

The helper method `createPublishXslt` creates a Publish_xslt structure:

```
public Publish_xslt createPublishXslt(String location, String publishingMethod, String import-
Method, String schemaMethod, String authInfo) throws InvalidParameterException {
        System.out.println("location = " + location);

        Publish_xslt publish = new Publish_xslt();
        publish.setLocation(location);
        publish.setPublishingMethod(PublishingMethod.getPublishingMethod(publishingMethod));
        publish.setImportMethod(RefPublishingMethod.getRefPublishingMethod(importMethod));
        publish.setSchemaMethod(RefPublishingMethod.getRefPublishingMethod(schemaMethod));
        publish.setAuthInfo(authInfo);

        return publish;
}
```

The `publishXsltResource` method performs the publishing operation:

```
public XsltMapping publishXsltResource(Publish_xslt publish) throws UDDIException, SOAPException
 {
        System.out.print("Check structure validity .. ");
        try {
            publish.check();
        } catch (InvalidParameterException e) {
            System.out.println("Failed!");
            throw new UDDIException(e);
        }
        System.out.println("OK");

        Xslt2uddiApi xsltApi = getXslt2UddiStub();
```

```
        System.out.print("Publishing in progress ...");
        XsltMapping xsltMapping = xsltApi.publish_xslt(publish);
        System.out.println(" done");
        return xsltMapping;
}
```

4.5.3. Building and Running Demos

This section shows, how to build and run the BEA AquaLogic Service Registry XSLT2UDDI demo set. Let us continue with our PublishXslt demo.

1. Be sure that the demos are properly configured and the BEA AquaLogic Service Registry is up and running.

2. Change your working directory to

Windows	%REGISTRY_HOME%\demos\resources\xslt
UNIX	$REGISTRY_HOME/demos/resources/xslt

3. Build all demos using:

Windows:	run.bat make
UNIX:	./run.sh make

Note

When compiling demos on Windows platforms, you may see the following text:

```
A subdirectory or file ..\..\common\.\build\classes already exists.
```

This is expected and does not indicate a problem.

4. To get list of all available demos, run

Windows:	run.bat help
UNIX:	./run.sh help

5. The selected demo can be executed via the **run** command using the name of the demo as a parameter. For example, to run the PublishWSDL demo, invoke

Windows:	run.bat PublishXslt
UNIX:	./run.sh PublishXslt

The output of this demo will resemble the following:

```
*********************************************************************
***              BEA AquaLogic Service Registry Demo - PublishXslt              ***
*********************************************************************
```

```
Publishing XSLT with the following parameters:
Enter XSLT location (URI) [http://localhost:8080/uddi/doc/demos/employeesToDepartments.xsl]:

Enter publishing method (update,create) [update]:
Enter import publishing method (reuse,create,ignore) [reuse]:
Enter schema publishing method (reuse,create,ignore) [reuse]:

Using Security at url https://localhost:8443/uddi/security ..
 done
Logging in .. done
location = http://localhost:8080/uddi/doc/demos/employeesToDepartments.xsl
Check structure validity .. OK
Using XSLT2UDDI at url https://localhost:8443/uddi/xslt2uddi .. done
Publishing in progress ... done

XSL transformation http://localhost:8080/uddi/doc/demos/employeesToDepartments.xsl
<xsltMapping xmlns="http://systinet.com/uddi/xslt2uddi/v3/5.5">
  <name xmlns="urn:uddi-org:api_v3">employeesToDepartments.xsl</name>
  <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xslt:employeesToDepart-
ments</tModelKey>
  <location>http://localhost:8080/uddi/doc/demos/employeesToDepartments.xsl</location>
  <contentMapping>
    <inputSchemaList>
      <xmlSchema>
        <namespace>http://systinet.com/uddi/demo/employeeList</namespace>
        <location>http://localhost:8080/uddi/doc/demos/employees.xsd</location>
        <tModelRef>
          <name xmlns="urn:uddi-org:api_v3">employees.xsd</name>
         <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xsd:employees</tMod-
elKey>
        </tModelRef>
      </xmlSchema>
    </inputSchemaList>
    <outputTypeList>
      <outputType>
        <xmlSchema>
          <namespace>http://systinet.com/uddi/demo/companyDepartments</namespace>
          <location>http://localhost:8080/uddi/doc/demos/departments.xsd</location>
          <tModelRef>
            <name xmlns="urn:uddi-org:api_v3">departments.xsd</name>
            <tModelKey xmlns="urn:uddi-org:api_v3">uddi:systinet.com:demo:xsd:depart-
ments</tModelKey>
          </tModelRef>
        </xmlSchema>
      </outputType>
    </outputTypeList>
    <outputMethod>xml</outputMethod>
  </contentMapping>
</xsltMapping>
Logging out .. done
```

Glossary

.NET	A software platform designed by the Microsoft Corporation. It is an environment for writing C#, Visual Basic, and C++ programs that can easily and securely interoperate.
.NET Framework	An environment for building, deploying, and running Web services and other applications. It consists of three main parts: the Common Language Runtime, the Framework classes, and ASP.NET.
.NET Framework Software Development Kit (SDK)	A set of documentation, samples, command-line tools, compilers, and the .NET Framework; that is, everything you need to write, build, test, and deploy .NET Framework applications.
Accepting Security Provider	A security provider that is responsible for accepting secure requests and usually also for determining the invoker identity. See Also Identity.
Access Control	Restrictions of a subject's access to a resource. See Also Access Controller, Subject.
Access Controller	An application component that is responsible for access control decisions. See Also Access Control.
accessPoint	A binding template element that indicates where you can find the endpoint of the Web service that is described by this entity. This may be a URL, an electronic mail address, or even a telephone number. See Also Universal Description, Discovery and Integration.
ACL	Access Control List — A list of entities, together with their access rights, the members of which have authorized access to a resource. See Also Subject.
Admin Service	The core System Web service, allowing you to manage advanced settings for each deployed service on a Systinet Server. Using this Web service it is possible to manage settings like security mechanisms, transport interceptors, polymorphism, automatic Web service authentication, and automatic authorization checks per Web service method.
Alias	A name that an entity uses in place of its real name.
Apache Containers	A schema for transferring containers proposed by Apache group. This schema is not compatible with Microsoft .NET.
Application Server-Dependent Deployment Descriptor	When an enterprise application is deployed on the server, it contains a set of deployment descriptors. They contain application metadata. Format and meaning of Application Server Dependent Deployment Descriptor is closely related to the application server and cannot be used in the context of any other application server.
Application Web services	Web services can be categorized into the three groups: System, Application, and Utility Web services. Application Services are created for specific tasks by the developer. To accomplish the task they typically use Utility Web services.
ASP .NET	ASP .NET is a unified Web development platform that provides the services necessary for developers to build enterprise-class Web applications.

Asynchronous Client Invocation	Client invocation of any Web service in an asynchronous way.
Asynchronous Return Mechanism	A service implementation returning the results of a call to Systinet Server in an asynchronous way.
Asynchronous Transport Coupling	Sending the response from a Web service invocation over a different transport channel than the one on which the request came.
Authentication	The process of establishing the validity of a claimed identity, it usually consists of two steps: 1/ identification - presenting identity credentials to the security system, 2/ verification - generating identity that corroborates the binding between the identity principals and credentials.
Authorization	The process of determining what types of activities are permitted. Usually, authorization is in the context of authentication. Once you have authenticated principals, they may be authorized different types of access or activity. See Also Authentication.
BEA WebLogic Application Server	An application server provided by BEA Systems, Inc.
Binding Template	For a businessService entry, a list of binding templates that point to specifications and other technical information about the service is associated. For example, a binding template might point to a URL that supplies information on how to invoke the service. The binding template also associates the service with a service type. See Also Universal Description, Discovery and Integration.
Borland Application Server	An application server provided by the Borland Software Corporation.
Borland Enterprise Server	An application server provided by Borland Software Corporation.
Business Entity	A representation of information about a business. Each business entity contains a unique identifier, the business name, a short description of the business, some basic contact information, a list of categories and identifiers that describe the business, and a URL pointing to more information about the business. See Also Universal Description, Discovery and Integration.
Business Service	A structure associated with a businessEntity that consists of a list of businessService structures offered by the businessEntity. Each businessService entry contains a business description of the service, a list of categories that describe the service, and a list of pointers to references and information related to the service. See Also Universal Description, Discovery and Integration.
C#	A modern, object-oriented language that enables programmers to build a applications for the Microsoft .NET platform.
Certificate	An electronic identifier from a certification authority that includes the certification authority signature made with its private key. The authenticity of the signature is validated by other users who trust the certification authority public key. See Also Certification Authority.
Certificate Chain	A list of Certificates (usually X.509 Certificates), starting with a certificate for a given subject that is signed by the authority represented by the next certificate in the list. This list usually ends with the root certification authority certificate. See Also X.509.

Certificate Revocation List	A data structure that enumerates digital certificates that have been invalidated by their issuer prior to when they were scheduled to expire. See Also Certificate.
Certification Authority	An entity that issues digital certificates (especially X.509 certificates) and vouches for the binding between the data items in a certificate. See Also X.509.
Client Package	Client side-specific information needed to invoke a specific Web service. This usually consists of a deployment descriptor and custom code, such as header processors, interceptors, serializers.
Client Profile (in Systinet Developer)	A set of client packages and additional configuration, such as security settings. See Also Client Package.
Clustering	The act of connecting multiple computers and making them act like a single machine. Corporations often cluster servers to distribute computing-intensive tasks and risks. If one server in a cluster fails, some operating systems can move its processes to another server, allowing end users to continue working while the first server is revived.
Content Based Routing (CBR)	An advanced and easy to use technique for message routing based on message content. See Also XPath.
Credentials	Data that is transferred to establish the claimed identity of an entity. According to RFC2828, a credential is the information one entity presents to another to authenticate the other's identity.
CRL	See Certificate Revocation List.
CTS (Common Types System)	A definition of how types work within runtime (their declaration and usage), which enables types in one language to interoperate with types in another language, including cross-language exception handling. See Also .NET.
DMZ (Demilitarized Zone)	An unprotected server on which all parties have access to everything. A web server may be put in the DMZ while the assets it accesses, such as databases, remain behind a firewall. It works in conjunction with transport layer security. See Also TLS.
Deploy Service	A System Web service that is used to deploy packages to a Systinet Server.
Deploy Tool	A part of Systinet Server that deploys and undeploys deployment packages to Systinet Servers.
Deployed Web service (in Systinet Developer)	A Web service that is assigned to a particular Deployment Package in the Project. See Also Deployment Package.
Deployment	The process of installing a deployment package to particular Systinet Server. See Also Deployment Package, Deployment Descriptor.
Deployment Descriptor	An XML document describing a package. See Also Deployment.
Deployment Package	A definition of Web services plus deployment information. See Also Deployment.

Deserialization	The process of creating Java objects out of a SOAP message.
Deserializer	A class that creates a Java object and fills it with the data from a SOAP message.
Distinguished Name	A distinguished name (DN) is a set of attribute values that identify the path leading from the base of the directory information tree to the object that is named. An X.509 public-key certificate or CRL contains a DN that identifies its issuer, and an X.509 attribute certificate contains a DN or other form of a name that identifies its subject. See Also Certificate, X.509.
Document/Literal	One possible encoding for a SOAP message, indicating that the message must strictly follow a schema written in the WSDL Document.
DOM	Document Object Model - a tree of objects with interfaces for traversing the tree and writing an XML version of it, as defined by the W3C specification.
DOM element	A structure representing an XML element as defined by DOM.
Dynamic Call	Constructing and issuing a request whose signature is possibly not known until runtime.
Dynamic Invocation	Constructing and issuing a request whose signature is possibly not known until runtime.
EAR File	Applications deployed on an application server are usually delivered as one compressed file with .ear extension. The file may contain software components, web applications, and resources.
EJB	Enterprise JavaBean.
Embedded Server	The Systinet Server in Systinet Developer that is used for testing purposes. It is tightly bound with the IDE.
Encoded Serialization	Serialization that uses an encoding layer to read/write data.
Endpoint	A referenceable entity (using, for example, a URL or URI).
Entity JavaBean	The kind of EJB that provides an object view of data in the database. See Also EJB.
Exception (Unhandled Java Exception)	An event during program execution that prevents the program from continuing normally.
Forte For Java	Sun Microsystems Forte For Java. An IDE for development of Java applications. It was a branded and commercial version of NetBeans; now it is named Sun ONE Studio (SOS). Systinet Developer for Sun ONE Studio is a plug-in that can be plugged into SOS and lets developers develop Web service based applications right in the IDE. See Also Sun ONE Studio.
Gateway	A connection between two MOMs. The gateway passes messages from one environment into the other. See Also MOM.
GSS-API	Generic Security Services API (GSS-API) is a programming interface that allows two applications to establish a security context independent of the underlying security mechanisms. Specified in RFC-2743. See Also Security Mechanism.
Header	A part of a SOAP message usually carrying some metadata.

Header Processor	A Java class for parsing/creating headers.
HTTP	HyperText Transfer Protocol. The Internet protocol, based on TCP/IP.
HTTPS	HyperText Transfer Protocol layered over the SSL protocol. See Also HTTP, Security Mechanism.
Identity	Information that is unique within a security domain and that is recognized as denoting a particular entity within that domain.
IETF	Internet Engineering Task Force (www.ietf.org).
IIS (Internet Information Services)	A secure platform for building and deploying business applications, hosting and managing Web sites, and publishing and sharing information across a company intranet or the Internet.
In Parameter	A parameter that is passed from client to server.
In/Out Parameter	A parameter that is passed in both directions. For example, it may contain an input value for the server and the processed result for the client.
Incoming Message	A message that is sent to the Systinet Server runtime. On the client side, this is a response message. On the server side, a request message.
Initiating Security Provider	A security provider that is responsible for initiating and maintaining secure communication from the client to the server side. See Also Security Provider.
Interceptor	A class for intercepting (that is, inspecting or modifying) the content of a message.
J2EE Application Server	An application server that is compliant with the J2EE specification published by Sun Microsystems Incorporated.
J2EE Specification	The Java 2 Platform, Enterprise Edition specification published by Sun Microsystems Incorporated.
JAAS	The Java Authentication and Authorization Service (JAAS) is a set of Java packages that enable services to authenticate and enforce access controls upon users. See Also Authentication, Authorization, Access Control.
JAR File	A file compressed using the Java Archive (JAR) file format.
Java Collections	A set of collections defined by the Java Platform specification (java.util.Map, java.util.Set, java.util.List).
Java Security	A set of Java security concepts based on the security framework provided by Java itself. See Also JSSE, JCE, JAAS.
Java2WSDL tool	A tool for converting Java classes and/or interfaces into their WSDL description.
JavaBeans Activation Framework	Standard services used to determine the type of an arbitrary piece of data, encapsulate access to it, discover the operations available on it, and to instantiate the appropriate bean to perform said operation(s).
JAX-RPC	A standard created by Sun's Java Community Process (#101) intended as a high-level API for calling Web services.

JAXM	A standard created by Sun's Java Community Process (#67) intended as a low-level API for calling Web services.
JCE	The Java Cryptography Extension - a set of packages that provide a framework and implementations for encryption, key generation and key agreement, and Message Authentication Code (MAC) algorithms. Support for encryption includes symmetric, asymmetric, block, and stream ciphers. The software also supports secure streams and sealed objects.
JDBC	Java DataBase Connectivity (JDBC) Data Access API.
JMS	The Java Message Service API.
JMS Destination	For sending and receiving messages, JMS uses a destination, which may be either JMS Topic or JMS Queue. See Also JMS.
JMS Message	A message sent by the Java Message Service. See Also JMS.
JMS Provider	A provider of JMS administered objects, such as JMS Queue or JMS Queue Connection Factory. See Also JMS.
JMS Queue	Used by the Java Message Service in Point-to-Point communications. See Also JMS.
JMS Queue Connection Factory	Used by the Java Message Service in Point-to-Point communications for creating JMS Connections. See Also JMS.
JMS Topic	Used by the Java Message Service in Publish/Subscribe communications. See Also JMS.
JMS Topic Connection Factory	Used by the Java Message Service in Publish/Subscribe communications for creating JMS Connections. See Also JMS.
JMS Transport	A pluggable transport that enables the sending of SOAP messages using the Java Message Service. See Also JMS.
JNDI	The Java Naming and Directory Interface; provides support for the common features of naming services including COS (Common Object Services), DNS (Domain Name System), LDAP (Lightweight Directory Access Protocol), and NIS (Network Information System). See Also LDAP.
JNDI Lookup	A lookup based on a unique JNDI name that returns an object bounded in the JNDI namespace. See Also JNDI.
JNDI Property	To use a specific implementation of JNDI, JNDI properties might be required to be set in the environment. See Also JNDI.

JSSE	The Java Secure Socket Extension - a set of Java packages that enable secure Internet communications. It implements a Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols and includes functionality for data encryption, server authentication, message integrity, and optional client authentication. Using JSSE, developers can provide for the secure passage of data between a client and a server running any application protocol (such as HTTP, Telnet, NNTP, and FTP) over TCP/IP.
JTA	The Java Transaction API.
Kerberos	A system developed at the Massachusetts Institute of Technology that uses passwords and symmetric cryptography (DES) to implement a ticket-based, peer-entity authentication service and an access control service distributed in a client-server network environment.
Key	Short for Cryptographic Key - an input parameter that varies the transformation performed by a cryptographic algorithm.
Key Entry	An entry in the key store consisting of an alias, a cryptographic key, and a certificate chain. See Also Alias, Key Store, Key, Certificate Chain.
Key Store	A Systinet Server component responsible for management of key entries. See Also Key Entry.
LDAP	Lightweight Directory Access Protocol (RFC-1777) - a client-server protocol that supports basic use of the directory servers, that is, database servers or other systems that provide information (such as digital certificates or CRL) about an entity whose name is known. See Also Certificate, CRL.
Library Package	Java class packages that provide their classes to other deployed packages. Java classes deployed in Systinet Server are normally accessible only inside their own packages.
Literal Serialization	Serialization driven only by XML Schema-type definitions.
Load-Balancing	Distributing processing and communications activity evenly across a computer network so that no single device is overwhelmed.
Local Name	A local part (without namespace) of a Qname. See Also Qualified Name.
Message	Data plus meta-information indicating how it is to be routed and handled. An example of a message is a SOAP message or transport-level message.
Message Processing	The process through which a message is processed by interceptors, serializers, and deserializers.
MIME	Multipurpose Internet Mail Extensions - a standard for sending data with attachments. This standard is set out in RFCs 2045, 2046, 2047, and 2048.
MOM	Message Oriented Middleware. An integration paradigm based on asynchronous message exchange. See Also Gateway.
Multipart Content	Content encoded in accordance with the MIME specification.

Namespace	Namespaces are typically established to distinguish between multiple interpretations of a single token or phrase. For example, a "nut" in the "food" namespace is something to eat, while in the "hardware" namespace something to fasten to a bolt (something you would not want to attempt with a "food:nut" and vice-versa). In XML, it can be thought of as a collection of names, identified by a URI reference [RFC2396], that are used in XML documents.
NetBeans	An open source platform primarily used for development of Java applications; it has evolved into a Tools Platform. The commercial and branded version of NetBeans is a product called Sun ONE Studio (formerly Sun Forte For Java). See Also Sun ONE Studio.
OASIS	Organization for the Advancement of Structured Information Standards (http://www.oasis-open.org) - an international, not-for-profit consortium that designs and develops industry standard specifications for interoperability based on XML.
Orion Application Server	An Application Server available from IronFlare AB of Sweden.
Out Parameter	A parameter that is sent from the server to the client.
Outgoing Message	A message sent out during Systinet Server runtime. On the client side, this message is called a request; on the server side, it is a response.
Package Manager (in Systinet Developer)	A part of a Systinet Server representation/Client Profile that is responsible for management of deployment/client packages. It also lets you view the installed packages and their Web services.
Package, Client Package	See Client Package.
Package, Library Package	See Library Package.
Package, Server Package	See Server Package.
Permission	An action that can be performed on a particular resource by a specific principal or role.
PDP- Policy Decision Point	A logical entity that is responsible for authorizing or denying access to services and/or resources.
Ping Service	A ping service is a System Web service that can be used as a lightweight method for determining whether a Systinet Server is running.
PKCS	The Public-Key Cryptography Standards are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography.
PKI	Public-Key Infrastructure - a system of certification authorities (and, optionally, other supporting servers and agents) that perform some set of certificate management, archive management, key management, and token management functions for a community of users in an application of asymmetric cryptography. See Also Certification Authority.
PEP - Policy Enforcement Point	A logical entity that enforces policy decisions.
POP, POP3	Post Office Protocol - a protocol for retrieval of email messages from mail servers. See Also POP3 server.

POP3 server	A mail server that supports the POP3 protocol from retrieval of email messages. See Also POP, POP3.
Port	A part of WSDL that binds an endpoint address and its interface.
PortType	Part of a WSDL document that describes the interface of a service. See Also WSDL.
Principal	An entity whose identity can be authenticated. A principal can represent any entity, such as an in individual, a corporation, or a login id.
Protected Store	A Systinet Server component consisting of a user store and key store. See Also User Store, Key Store.
Proxy Host	The host name of a proxy server.
Proxy Port	Port number of a proxy server.
Proxy, dynamically generated	A Java object that acts as a proxy to a Web service. Invoking methods on this object results in a SOAP request and response exchange with the Web service.
Public Cloud	A Universal Business Registry where businesses can describe and publish their web services to the general public. See Also UBR.
Publisher Assertion	A structure that allows you to emphasize a relationship between two Business Entities. See Also Universal Description, Discovery and Integration.
QName	See Qualified Name.
Qualified Name	A name that consists of a namespace and a unique name from that namespace. See Also Namespace.
Receiver	A referenceable entity that accepts messages. This can be overseen as a Web service, an asynchronous endpoint, or a stub/proxy that accepts a response.
Reference	A reference to data that are defined in another part of the message. An example might be a reference to the next MIME part of a message or a reference to repeated Java objects.
Reliability	The ability of messages to be delivered regardless of software component, system, or network failures. See Also WS-ReliableMessaging.
Remote Debugging (in Systinet Developer)	Debugging of Web services that are deployed to a remote Systinet Server. In Systinet Server for Java Developer, you can place a breakpoint into your Web service source code, switch-on Remote Debugging Support for the Systinet Server and debug this Web service remotely even when it is running on a remote machine.
Remote Server	In Systinet Developer, you have a list of Systinet Servers that you can work with. You can register any running Systinet Server into this list so you can work with it (remotely manage this server, deploy Web services to this server etc.).
REST	REpresentational State Transfer is an architectural module used to implement networked IT systems. The modeling of communication between components is similar

	to that used by HTTP. The main distinguishing features of this model relate to resources.
Return Value	A single value returned from a service.
Role	A category that applies to a set of principals.
RFC	An IETF Request For Comments (see http://www.ietf.org/rfc) - usually a standard or a recommendation.
RPC	Remote Procedure Call - an extension of a common procedure call used inside one application to span multiple processes running on multiple hosts.
RPC/Encoded	One possible SOAP message encoding, indicating that the message format is logically given by the XML schema present in the WSDL. The physical representation of the message is given by the encoding of the message. See Also WSDL.
SAML	Security Assertions Markup Language - an XML framework for exchanging security information over the Internet. SAML enables disparate security services systems to interoperate. It resides within a system's security mechanisms to enable exchange of identities and entitlements with other services.
Scalability	How well a system can adapt to increased demands. For example, a scalable network system would be one that can start with just a few nodes, but easily expand to thousands of nodes.
Schema Type	Defines the type of a part of XML data.
Security Manager	The component of the Systinet Server responsible for security management.
Security Mechanism	A mechanism that implements a security function. Some examples of security mechanisms are authentication exchange, checksum, digital signature, encryption, and traffic padding.
Security Provider	A provider for particular security mechanism(s). See Also Security Mechanism.
Sender	An entity that sends messages.
Sequence Owner	A load balancer node that handles all the messages in a WS-RM reliable managing sequence. The reliable message sequence corresponds to a load balancer session. See Also WS-ReliableMessaging.
Serialization	The process by which binary objects are written into a structured stream; for example, when Java objects are written into a SOAP message.
Serializer	A class that writes a Java object into a SOAP message.
Server Package	The package that holds all the service-related files. See Also Deployment Package, Deployment.
Service Class	The implementation class of the Web service.
Service Endpoint	A single endpoint of a service instance with an associated path and additional configuration (such as header processors, serializers, etc.).

Service Instance	A service class instance registered in Systinet Server for Java.
Service Lookup	See Web Service Lookup.
Service Manager	A component of the Systinet Server that is responsible for management of deployed Web services.
Service State	The current state of a service instance; for example, Offline, Starting, Running, Stopping, Stopped.
Service, Asynchronous Java Service	A Web service implemented in Java that returns the results of an invocation in an asynchronous manner.
Service, Java Service	A Web service implemented in Java that handles the messages using Java types representation of their content.
Service, Raw Service	A Service written in Java that handles the messages using a low-level transport message API.
Service, XML Service	A Service written in Java that handles the messages using the low-level SOAP Message API.
Servlet	The basic part of Java Servlet Technology.
Servlet Container	A container application that allows servlets to run. See Also Servlet.
Single Login	A system of applications, where a principal (user) authenticates with one system entity (called identity provider) and has that authentication honored by other system entities (called service providers or partners). See Also SSO (Single Sign-On).
SMTP	Simple Mail Transfer Protocol - a protocol for sending email messages between servers. Most email systems that send mail over the Internet use SMTP to send messages from one server to another; the messages can then be retrieved with an email client using either POP or IMAP. In addition, SMTP is generally used to send messages from a mail client to a mail server.
SMTP Server	A mail server that supports the SMTP protocol for email transfer. See Also SMTP.
SOAP	Simple Object Access Protocol - a lightweight protocol based on XML for the exchange of information in a decentralized, distributed environment.
SOAP Body	The part of a SOAP message that contains the actual data. See Also SOAP.
SOAP Digital Signature	The W3C document SOAP Security Extensions: Digital Signature specifies the syntax and processing rules for a SOAP header entry to carry digital signature information within a SOAP 1.1 Envelope. See Also SOAP, SOAP Header, SOAP Envelope, XML Signature.
SOAP Envelope	The root element of a SOAP message. It contains exactly one body sub-element and optionally one header sub-element. See Also SOAP.

SOAP Fault	Used to return errors that occur during the routing/processing of a SOAP message. See Also SOAP.
SOAP Fault-Actor	Part of a SOAP Fault. It provides information about who/what caused the fault. See Also SOAP Fault.
SOAP Fault-Code	Part of a SOAP Fault. It provides an numeric identification of the fault. See Also SOAP Fault.
SOAP Fault-Detail	Part of a SOAP Fault that provides more details about the fault. In Systinet Server for Java, this element usually contains a server stack trace. See Also SOAP Fault.
SOAP Header	The part of soap message that contains metadata (for example, authentication information or instance identification) of the message. See Also SOAP Body.
SOAP Message	A message encoded in accordance with the SOAP specification. See Also SOAP.
SOAP with Attachments	Binding for a SOAP message to be carried within a MIME multipart/related message in such a way that the processing rules for the SOAP 1.1 message are preserved. See Also SOAP.
SOAPSpy	A SOAP message-tracking tool that scans communication between the client and sever. The communication is visually displayed. You can also manually change and send the messages. See Also SOAP.
SOS	See Sun ONE Studio.
SPKM	Simple Public Key Mechanism - a security mechanism specified by the IETF in RFC-2025.
SQL Statement	A statement of the Structured Query Language.
SSJ	Abbreviation for Systinet Server for Java™.
SSL	The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols were designed to help protect the privacy and integrity of data while it is transferred across a network. The Internet Engineering Task Force (IETF) standard called Transport Layer Security (TLS) is based on SSL. See Also TLS.
SSO (Single Sign-On)	A system that enables a user to access multiple computer platforms or application systems after being authenticated only once. See Also SAML, Kerberos.
Static Invocation	Constructing a request at compile time. Calling an operation via a proxy procedure.
Stub	A statically-generated service interface, which in turn dynamically generates the proxy during runtime.
Subject	A grouping of related information for a single entity, such as a person. Such information includes the Subject's identities, as well as its security-related attributes (passwords and cryptographic keys, for example).

	See Also Identity.
Sun ONE Studio	Sun ONE Studio (formerly Sun Forte For Java) is an IDE for development of Java applications. It is a branded and commercial version of NetBeans. Systinet Developer for Sun ONE Studio is a plug-in that can be used with SOS and lets developers develop Web service-based applications in the IDE. See Also NetBeans.
Systinet Developer	A product of Systinet Corporation that lets developers create, test, debug, and manage Web services using their favorite IDE. Systinet Developer is a plug-in that enhances IDEs such as Sun Microsystems Sun ONE Studio, Borland JBuilder, and IBM Eclipse.
Systinet Server for Java Application Directory	A directory to which the WASP_HOME parameter points. See Also Deployment, WASP_HOME.
Systinet Server for Java Root URL	The URL where the Systinet Server runs. The Global URL of the Web service running on Systinet Server is <Systinet Server for Java Root URL> + <path of the Service Endpoint>.
System Web Services	Web services can be categorized into three groups: System, Application, and Utility Web services. System Web services facilitates fundamental functions such as service deployment, administration and security settings management.
Target Namespace	In WSDL, XML Schema, or a deployment descriptor document, the namespace into which the content of the document is placed.
TLS	Transport Layer Security protocol. Its primary goal is to provide privacy and data integrity between two communicating applications. The first version of TLS is described in RFC-2246. See Also SSL.
tModel	A structure that takes the form of keyed metadata (data about data). In a general sense, the purpose of a tModel within the UDDI registry is to provide a reference system based on abstraction. Among the roles that a tModel plays in UDDI is the ability to provide and to describe compliance with a specification or concept to a taxonomy, for example.
Transport	A component of the Systinet Server that is responsible for transferring messages to a Web service using particular transport protocol.
Transport Message	A message accessible via Systinet Server Transport API. See Also Message.
Transport Repository	A repository of all Systinet Server transports.
Trusted Certificate Entry	An entry managed by the key store that represents a trusted certificate or certificate chain. See Also Key Store, Certificate Chain.
UBR	Universal Business Registry (also known as Public Cloud) - a set of UDDI Registries that form a global distributed registry of information about Web services. Note that UBR nodes (members of the Public Cloud) are run by Microsoft, IBM, SAP, HP, and NNTP. They replicate the content of Public Cloud.
UDDI	See Universal Description, Discovery and Integration.

UDDI Green Pages	UDDI accepts and organizes three types of information into three broad categories: White, Yellow, and Green Pages. Green Pages hold the technical information about services that are exposed by the business, including references and interfaces to the services a company can deliver.
UDDI Inquiry Port	Every UDDI Registry implementation provides two ports with which you can interact: inquiry and publishing. The inquiry port allows you to browse and search information that is published to a UDDI Registry.
UDDI node	The UDDI node is a collection of Web services, each of which implements the APIs in a UDDI API set, and that are managed according to a common set of policies. Typically, a node consists of at least an implementation of the Inquiry, the Publication, and the Custody and Ownership Transfer API sets; often a node will implement additional API sets such as Subscription and Replication.
UDDI Operator	A UDDI Operator is a role of a person who sets node policy and runs a node. There is exactly one operator for a given node.
UDDI Publishing Port	Every UDDI Registry implementation provides two ports with which you can interact with: inquiry and publishing. The publishing port allows you to publish information about your Web services.
UDDI Registry	A UDDI Registry is an implementation of the UDDI specification that allows Web service vendors to register information about the Web services they offer so that others can find them.
UDDI White Pages	UDDI accepts and organizes three types of information into three broad categories: White, Yellow, and Green Pages. White Pages include address, contact, and known identifiers.
UDDI Yellow Pages	UDDI accepts and organizes three types of information into three broad categories: White, Yellow, and Green Pages. Yellow Pages include industrial categorizations based on standard taxonomies.
Undeployment	Undeployment is a process of uninstalling deployed packages from the Systinet Server. See Also Deployment.
Universal Description, Discovery and Integration	UDDI is a specification for distributed Web-based information registries of Web services.
Updatable Policy	A Systinet Server component responsible for management of access control lists. See Also ACL.
URI	Uniform Resource Identifier - the generic term for all types of names and addresses that refer to objects on the World Wide Web. A URL is one kind of URI.
URL	Uniform Resource Locator - the global address of documents and other resources on the World Wide Web. The first part of the address indicates what protocol to use and the second part specifies the IP address or the domain name where the resource is located.
User	Any person who interacts directly with a computer system. Note that 'users' do not typically include 'operators,' 'system programmers,' 'technical control officers,' 'system security officers,' and other system support personnel.

User Group	A named collection of user identifiers. See Also User.
User Property	In the context of Systinet Server for Java, a user attribute that can be stored in the user store. See Also User Store.
User Store	A Systinet Server component responsible for management of user (security) properties, such as passwords and certificates.
Utility Web services	Web services can be categorized into three groups: System, Application, and Utility Web. A Utility Service typically provides commonly required functionality utilized by any Application Web service. It provides an easy way for developers to reuse common functions to produce more reliable code and reduce redundancy.
UUID	Universally Unique Identifier as used in http://www.ietf.org/ recommendations or drafts.
WAR File	A format for compressing files, similar to a JAR file. Web applications that may be deployed to an application server are often compressed into WAR files. See Also JAR File.
WASP, WASP Server for Java	The former name of Systinet Server for Java™.
WASP_HOME	The directory where the Systinet Server distribution is installed.
WaspPackager Tool	A part of Systinet Server for Java that creates deployment packages that can be deployed to Systinet Servers or client packages that are used for Web service Clients. See Also Deployment, Deployment Package, Client Package.
Web Service	Loosely coupled software components delivered over Internet standard technologies.
Web Service Client	An application that uses Web services.
Web Service Debugger (in Systinet Developer)	A special kind of Sun ONE Studio Debugger Type that must be used for debugging Web service clients. This Debugger Type ensures the correct initialization of the client part of Systinet Server for Java.
Web Service Executor (in Systinet Developer)	A special kind of Sun ONE Studio Executor that must be used for running Web service Clients. This Executor ensures the correct initialization of the client part of Systinet Server for Java.
Web Service Lookup	A process through which a remote Web service is bound to a Java interface. The result of this process is a Java stub for the Web service.
Web Services Description Language Utility (wsdl.exe)	Used to generate code for XML Web service clients and XML Web services using ASP.NET from WSDL contract files and XSD schemas.
WSDL	An XML-based language that describes an interface of a Web service plus information on how to call the Web service and where to find it.
WSDL Compiler	The previous name for WSDL2Java, a Systinet Server tool that converts a WSDL document into Java code.
WSDL Compiler tool	See WSDL Compiler.

WSDL Compiler Web service	Former name of the WSDL2Java Web service, a utility Web service that offers SOAP access to the WSDL2Java tool used for the generation of Java source files from a WSDL document.
WSDL Operation	Part of a WSDL Document representing the interface of an operation that can be invoked on a Web service.
WSDL Port	Part of a WSDL Document that binds the endpoint of a service with an interface.
WSDL Service	Part of WSDL Document that specifies the set of endpoints that define one logical service.
WS-Addressing	A protocol that provides transport-neutral mechanisms to address Web services and messages. Specifically, WS-Addressing defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. It enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner. For more information, please see the WS-Addressing specification. [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-addressing.asp]
WS-Eventing	Specification which describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages. For more information, please see the WS-Eventing specification. [http://msdn.microsoft.com/webservices/community/workshops/default.aspx?pull=/library/en-us/dnglobspec/html/ws-eventing.asp]
WS-Policy	The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service. WS-Policy defines a base set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements, preferences, and capabilities. For more information, please see the WS-Policy specification. [http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-policy.asp]
WS-ReliableMessaging	A protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures. Is used in conjunction with other specifications and application-specific protocols within the SOAP [SOAP] and WSDL [WSDL] extensibility model. The draft version of this protocol was known as WS-Reliability. For more information, please see the WS-ReliableMessaging specification. [http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-reliablemessaging.asp]
WS-RM	See WS-ReliableMessaging.
WS-Security	WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. It enables the user to encrypt and/or sign individual SOAP messages.

Systinet Server for Java provides an implementation of OASIS' working draft 13 [http://www.oasis-open.org]. It is based on a Systinet-modified version of Apache XML-Security package 1.0.4 [http://xml.apache.org/security].

For more information, please see the WS-Security specification. [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp]

X.509

Part of the ITU-T X.500 specification that defines a framework to provide and support data origin authentication and peer entity authentication services, including formats for X.509 public-key certificates, X.509 attribute certificates, and X.509 CRLs.
See Also CRL.

XKMS

The XML Key Management Specification - a specification designed to extend the public key infrastructure (PKI) model by using XML to provide new levels of ease and interoperability when implementing secure applications.
See Also PKI, XML.

XML

eXtensible Markup Language - a W3C-sponsored format for structured documents and data, used mostly on the Web.

XML Canonicalization

A method for generating a physical representation, the canonical form, of an XML document that accounts for permissible changes or variations in syntax. It is a reduction of a document to a standard minimal form useful, among other things, for document or structure comparisons. Except for limitations regarding a few unusual cases, if two documents have the same canonical form, then the two documents are logically equivalent within the given application context.

XML Encryption

A standard that specifies the process for encrypting data and representing the result in an XML document. The data may be an XML element, or XML element content, or any arbitrary data (including an XML document).
See Also XML, XML Signature.

XML protocol

A communication or messaging protocol based on XML.

XML Schema

A means for defining the structure, content and semantics of XML documents through XML itself. It defines a richer set of data types - such as booleans, numbers, dates and times, and currencies - than the more traditional DTD. XML Schemas make it easier to validate documents based on namespaces. It is defined in the W3C's XML Schema Working Group.

XML Signature

A way of providing integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.
See Also XML, XML Encryption.

XPath

A language for addressing parts of an XML document. See XPath 1.0 [http://www.w3c.org/TR/xpath] and XPath 2.0 [http://www.w3c.org/TR/2004/WD-xpath20-20041029/].
See Also XSLT, XQuery, Content Based Routing (CBR).

XQuery

A query language able to express queries across data structured as XML. The result of an XQuery program is also XML. XQuery can be viewed as a transformation language. See XQuery 1.0 [http://www.w3c.org/TR/2004/WD-xquery-20041029/].
See Also XPath.

XSLT

A language for transforming XML documents to other XML documents or more generally any text output. Its expressive power is greater than XQuery. Hence it is more universal. See XSLT 1.0 [http://www.w3c.org/TR/xslt] and XSLT 2.0 [http://www.w3c.org/TR/xslt20/].
See Also XPath, XQuery.